Dottorato di Ricerca in Informatica
Università di Bologna, Padova, Venezia

Specification and Analysis of Stochastic Real-Time Systems

Mario Bravetti

12 Febbraio 2002

Coordinatore:
Prof. Özalp Babaoğlu

Tutore:
Prof. Roberto Gorrieri

Abstract

A formal approach for the design and analysis of concurrent systems is proposed which integrates two different orthogonal aspects of time: (i) the aspect of probabilistic-time, concerning the probabilistic quantification of durations of system activities via exponential probability distributions and the evaluation of system performance through the analysis of Continuous Time Markov Chains (CTMCs), and (ii) the aspect of real-time, concerning the expression of time constraints and the verification of exact time properties. We show that these two aspects, that led to the introduction of different specification paradigms called Markovian process algebras and timed automata, respectively, can be expressed in an integrated way by a single language, namely a process algebra capable of expressing both probabilistic durations and time bounds.

In the first part of the thesis we present some fundamental results about process algebras with durational actions that will be used in the rest of the thesis. In particular we present three techniques for expressing the semantics of durational actions in process algebra (called ST semantics in the literature), where actions are seen as a combination of start and termination events. The first technique, called the static name technique, allow us to decide weak ST bisimulation equivalence over a wide class of recursive processes; the second technique, called the dynamic name technique, allow us to decide and axiomatize weak ST bisimulation equivalence over the same class of processes, at the price of an increased size of the semantic models of processes; while the third technique, called the stack technique, allows us to keep deciding and axiomatizing weak ST bisimulation equivalence when we introduce an action refinement operator, at the price that names for actions are lost in the semantic models of processes.

In the second part of the thesis we present a stochastic real-time process algebra which solves the problem of integrating timed automata and Markovian process algebras, by expressing activities with generally distributed durations, which are given an ST semantics. The introduction of such a language, called calculus of Interactive Generalized Semi-Markov Processes (IGSMPs), besides giving the opportunity of modeling and analyzing concurrent systems accounting for both probabilistic-time and real-time aspects in an integrated way, offers to the modeler a great expressive power making it feasible to produce detailed specifications of real systems. IGSMP specifications can be directly analyzed through standard discrete event simulation and by means of new techniques that we introduce: minimization via a notion of bisimulation based equivalence which abstracts from internal system activities and derivation of the underlying performance model in the form of a Generalized Semi-Markov Process (GSMP). We also present formal techniques for compositionally deriving, from an IGSMP specification, a pure real-time model (called Interactive Timed Automaton), by considering the support of general distributions, and a pure probabilistic-time model (called Interactive

Weighted Markov Chain), by approximating general distribution with phase-type distributions (combinations of exponential distributions), which are guaranteed to be consistent, in that represent different aspects of the same initial system specification, and can be analyzed by exploiting standard techniques and existing software tools. The latter technique is particularly significant in that: (i) it shows process algebra to provide exactly the machinery necessary for approximating GSMPs with CTMCs through phase-type distributions, and (ii) it shows ST semantics to be the adequate semantics for generally distributed time in that approximation of activity durations with phase-type distributions is a form of action refinement. We conclude this part of the thesis by solving the issue of dealing with weakly guarded recursion in interactive timed processes: we present a complete axiomatization over finite-state weakly guarded interactive timed processes.

In the third part of the thesis we present an alternative methodology to integrate fixed and probabilistic durations in a single language, which is based on discrete time, as in Discrete Time Markov Chains (DTMCs), instead of continuous time (as in CTMCs and GSMPs). Such an approach expresses the behavior of processes by means of generative-reactive transition systems: an extension of DTMCs with action types expressing multi-way process synchronization through a mixture of the classical generative and reactive models of probability. Moreover, differently from classical discrete time process algebras, where parallel processes are executed in synchronous locksteps, it allows processes with (possibly) different mean advancing speeds to be modeled, hence processes whose actions have a different mean duration. As a further result we show that, when evaluating steady state based performance measures, such an approach provides an exact solution even if advancing speeds are considered to be exact instead of probabilistic, i.e. actions have a fixed duration, without incurring the state space explosion problem which arises with standard synchronous approaches. We show the effectiveness of this approach by presenting a case study on multi-path routing.

The purpose of the fourth part of the thesis is to integrate the two approaches presented in the second and third part into a single language and to furthermore add expressive features, such as multi-level priorities and rewards, in order to make it suitable for modeling and analyzing real systems. We solve this issue by first developing $EMPA_{\mathrm{gr}}$, an extension of the existing algebra Extended Markovian Process Algebra (EMPA) with the generative-reactive synchronization mechanism introduced in the third part of the thesis. $EMPA_{\mathrm{gr}}$ is a Markovian process algebra capable of representing both discrete and continuous time systems, where we can express: activities with exponentially distributed durations (which are not considered when doing discrete time specifications), activities with a discrete duration (which are assumed to take time zero in continuous time), probabilities, multi-level priorities and the generative-reactive synchronization mechanism. Then we extend the process algebra $EMPA_{\mathrm{gr}}$ with the capability of expressing performance measures at the algebraic level by attaching bonus and yield rewards to actions. Finally we develop the algebra Generalized Semi-Markovian Process Algebra ($GSMPA$) which further extends the expressive power of $EMPA_{\mathrm{gr}}$ to activities with a generally distributed duration.

For all the process algebras that we introduce we present a notion of bisimulation equivalence (which abstracts from internal actions for the algebras of the second part of the thesis) which is shown to be a congruence for all the algebra operators and (apart from GSMPA) a sound axiomatization which is shown to be complete over finite-state algebra processes.

Acknowledgements

First of all, I would like to thank my Ph.D. supervisor Prof. Roberto Gorrieri. I'm really grateful to him for the valuable discussions and the always insightful and competent suggestions and comments. Moreover, I thank him for having been an important guide during my graduate studies and for the financial support and travel opportunities he gave me. I'm also very grateful to Marco Bernardo and Alessandro Aldini for the strict and gratifying collaboration on the extensions to the stochastic process algebra EMPA and on the investigation of generative-reactive probabilistic systems, respectively. Certainly this piece of work would have not been possible without the cooperation and moral support of Roberto, Marco and Alessandro.

I would like to thank Prof. Jane Hillston and Prof. Joost-Pieter Katoen for their kind willingness to review my Ph.D. thesis. They had to do a lot of work in a very short time, thus making it possible for me to get the Ph.D. degree in the year 2002. Thanks a lot to both of them.

I'm also grateful to many people which I met during conferences, schools and visits. Discussions and remarks I got from them have often inspired and influenced the contents of this thesis. In particular, I would like to explicitely thank: Holger Hermanns for the many animated (and often noisy!) discussions filled up with genuine scientific passion, Arend Rensink for the conversations on ST semantics during his visits to Bologna, Gerald Lüttgen for his remarks and comments in the light of his considerable knowledge about priority in process algebra, and Rance Cleaveland for the cooperation during my short visit to the State University of New York at Stony Brook. A special thank also to Roberto Segala for the uncountable philosophical discussions on computer science carried out while on leave together for some conference or school, to the entire PAPM community which constituted the "environment" in which I started and grew up as a researcher (I remember with great emotion my first talk in English in Enschede at PAPM '97), and to Pierpaolo Degano, who, with a comment during a competition for a Ph.D. position, made me realize that the semantics that I had in mind for general distributions in process algebra was actually the ST semantics.

Many thanks to all my colleagues in Cesena and Bologna who made pleasant going to work every day. In particular, I would like to acknowledge the friendship of Nadia Busi and Gianluigi Zavattaro, who made of every break a big fun and always took care of making me aware of what was happening around me while I was, as usual, absorbed by research work. Moreover, thanks to Paola Salomoni for having been such a pleasant office-mate and for the help she gave me in getting acquainted with the academic environment of Cesena, to Vittorio Ghini for the wise pieces of advice he gave me when I had some trouble or had to take some important decision, and to Irene Schena for always being so joyful and ready to speak about everything.

Last but not least, I would like to thank all my friends, my parents and my family.

This thesis is dedicated to Elisa, who gave me the most beautiful and fulfilling moments of these years, who closely supported me day by day, and who caused colors never to leave the world around me.

The Expressive Sheila's Poem

To Torey With Much
"Love"

And the rest came
They tried to make me lough
They played games with me
Some games for fun and some
For real and for keeps.
And then they went away
Leaving me in the ruins of the games
Not knowing which were for fun
And which were for keeps and
Leaving me alone with the echoes of
Laughter that wasn't mine.

Then you came
With your funny way of being
Not quite human
And you made me cry.
And you didn't seem to care if I did
You just said the games are over
And waited
Until all my tears turned into
Joy.
With much love,
Sheila

The expressive poem of Sheila (about 11 years old) dedicated to her teacher Torey Hayden, from the book "One Child" by Torey L. Hayden.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Thanks to the success of personal communication and calculation technologies, of computer networks and of multimedia global systems (WEB), the role played by distributed systems and communication protocols in modern society is getting more and more prominent in every day life. However, an adequate comprehension of the phenomena that such complex systems may exhibit, as e.g. coordination problems or problems arising from a bad system performance, that turn out to be important from a practical viewpoint, has not gone along with their development. A rather unfortunate consequence of the absence of an adequate theory for such systems is that the software produced does not offer enough correctness guarantees, a problem which is particularly severe whenever human lives may depend from software correctness. Formal methods are assuming a more and more important role in the design of software which is critical for such systems, especially because they offer a strong theoretical basis for verifying their correctness and estimate their performance. Moreover they offer instruments for early system prototyping that may contribute to reduce the time of software development, helping to detect possible design errors in the early stages of production and avoiding the high costs arising from detecting them at a late stage.

The development of a software product requires, similarly as the other industrial products, an engineering cycle which is mainly composed of an initial specification stage and of a successive implementation stage. By using formal methods, it is possible to develop non-ambiguous specifications of systems which are analyzable with automated software tools. In particular, it is possible to verify the correctness of such specifications with respect to a certain set of requirements or to carry out a preliminary analysis of system performance, so to detect the design decisions that give the best results. The main objective of this thesis is to develop new techniques for the formal verification of real-time properties and the evaluation of performance of distributed systems and communication protocols.

1.1 Modeling and Analyzing Concurrent Systems via Stochastic Time and Real-Time

The importance of considering the behavior of concurrent systems with respect to time during their design process has been widely recognized [107, 87, 7, 19, 49, 90, 139, 39, 69, 99, 113, 44, 36, 11, 133, 132, 131].

In particular two different approaches for expressing and analyzing time properties of systems have been developed which are based on formal description paradigms.

A first approach is devoted to the *evaluation of the performance* of concurrent systems (see e.g. [107, 87, 19, 138, 99]). According to this approach the time spent by a system in a certain activity is expressed probabilistically through a distribution of duration. Performance measures of systems can then be evaluated via mathematical or simulative techniques. This approach has led to the definition of *stochastic process algebras*, an extension of standard process algebras where a distribution of duration is associated with each action of a process. In most cases, as in [19], the expressiveness of such algebras is limited to exponential distributions of time, because this causes the passage of time to be "memoryless". As a consequence it is possible to completely avoid explicitely representing durations in semantic models. Moreover the limitation to *exponential distributions* allows for a straightforward transformation of the semantic model of a system into a *Continuous Time Markov Chain* (CTMC), a stochastic process which is easily mathematically analyzable for deriving performance measures. For this reason they are called *Markovian process algebras*. It is worth noting that the limitation imposed over durations is very strong because not even deterministic (fixed) durations can be expressed.

A second approach concentrates on the aspect of *real-time*, i.e. the expression of time constraints and the verification of exact time properties (see [11, 133, 132, 131] and the references therein). By this approach the parts of the system that are critical from the viewpoint of time bounds can be validated during the design phase through techniques such as e.g. *model checking* [11]. In this view *timed automata* have been developed by extending standard labeled transition systems with the representation of time by means of *clocks*. The time value assumed by a clock in a timed automata increases as time passes. In timed automata we have transitions representing the setting of a clock with a certain time value and transitions which can be executed provided that clocks satisfy a certain time constraint (see e.g. [11, 133, 132, 131]).

1.2 Integrating Stochastic Time and Real-Time via General Distributions

In the following we present the basic idea for integrating stochastic time and real-time in a single specification paradigm by means of generally distributed probabilistic time that we introduced in [32], we discuss how to develop a specification language expressing generally distributed time, and we present an integrated approach for the specification and analysis of Stochastic Real-Time concurrent/distributed systems.

1.2.1 The Basic Idea

The different aspects of time expressed by the Stochastic Time and Real-Time approaches can be seen as being *orthogonal*.

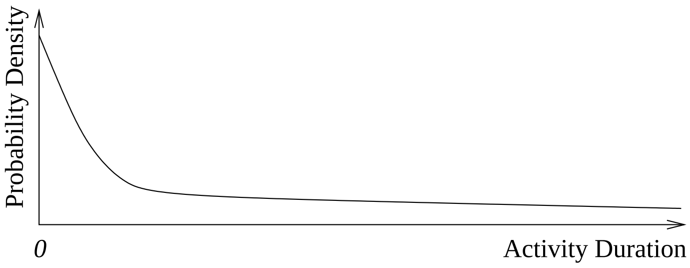


Figure 1.1: Stochastic Time (Markovian) Activity

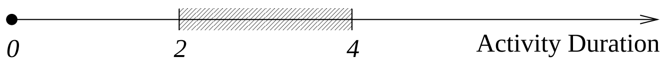


Figure 1.2: Real-Time Activity

According to the first approach the possible values for the duration of an activity are quantified through probabilistic (exponential) distributions, but no time constraint is expressible: all duration values are possible with probability greater than zero. In Fig. 1.1 we depict the probability density for the duration values of an activity with an exponentially distributed duration.

According to the second approach some interval of time is definable for doing something, but the actual time the system spends in-between interval bounds is expressed non-deterministically. For instance, in Fig. 1.2 we depict an activity whose duration must be between 2 and 4 time units. Note that activities with a deterministic (fixed) duration are expressed when interval bounds coincide. For instance, in Fig. 1.3 we depict an activity whose duration is certainly 3.

A specification paradigm capable of expressing both aspects of time should be able of expressing both time constraints and a probabilistic quantification for the possible durations which satisfy such constraints. We obtain such an expressive power by considering stochastic models capable of expressing *general probability distributions* for the duration of activities. In this way time constraints are expressible via probability distribution functions that associate probability greater than zero only to time values that are possible according to the constraints. Technically, the set of possible time values for the duration of an activity is given by the *support* of the associated duration distribution. This idea of deriving real-time constraints from

Figure 1.3: Activity with a Fixed Duration

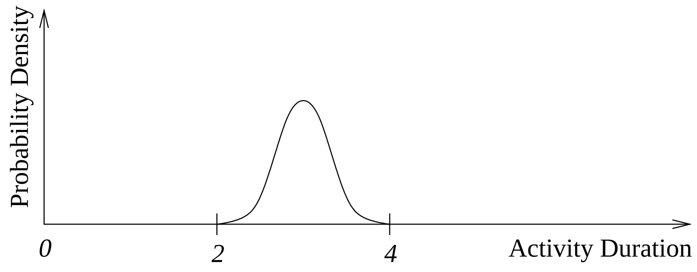


Figure 1.4: Generally Distributed Activity

distribution supports, that we have introduced in [32], was subsequently applied also in [50] and [67]. For instance, in Fig. 1.4 we depict an activity with a distribution whose support is the interval of Fig. 1.2. Note that with this approach we can also represent deterministic durations via trivial distribution functions that give all the probability to a single value of time.

Representing the real-time and probabilistic-time in a single specification paradigm allows us to model a concurrent system more precisely by expressing and analyzing the relationships between the two aspects of time. Moreover, the capability of expressing general distributions gives the possibility of producing much more realistic specifications of systems. System activities which have an uncertain duration could be represented probabilistically by more adequate distributions than exponential ones (e.g. Gaussian distributions or experimentally determined distributions).

1.2.2 Process Algebras with Generally Distributed Time

A formalism expressing systems with generally distributed delays should originate from probabilistic models which are well-founded from the viewpoint of probability theory. In particular it is well-known that the behavior of systems employing generally distributed durations which can be executed in parallel are stochastic process called *Generalized Semi-Markov Process* (GSMP). A GSMP describes the temporal behavior of a system by using *elements*, which act similarly as clocks of a Timed Automata. In particular the temporal delays in the evolution of a system are represented by clocks (elements) whose duration is determined by an associated generally distributed random variable. In this way the temporal behavior of the system is "guided" by the events of start and termination of clocks (elements).

The most natural way to develop a specification paradigm for concurrent systems with generally distributed delays is, therefore, to consider a model with clocks (similar to a Timed Automaton), where each clock has an associated general probability distribution of duration. Moreover in the model *clock start* and *clock termination* events must be expressed. We consider this approach using clocks for modeling systems with generally distributed delays to be convenient with respect to other approaches in the literature (see Chapter 6) since it makes it relatively simple to turn models into GSMPs or Timed Automata for analyzing system properties. Based on this idea, in Chapter 6 we introduce Interactive Generalized Semi-Markov

Processes (IGSMPs) as models which give the possibility of specifying systems as the parallel composition of subsystems described by GSMPs (along the lines of [99], where the same problem is attacked for Continuous Time Markov Chains). An IGSMP represents the behavior of a component through a set of clocks with an associated duration distribution, by employing both standard action transitions, representing the interactive behavior of the component, and clock start transitions and clock termination transitions, representing, in analogy to Timed Automata, the timed probabilistic behavior of the component. Clock start transitions are labeled with a clock name and a *weight* [150] and represent the event of start of a temporal delay whose probabilistic duration is given by the distribution associated with the clock. Weights are used to express probabilistic choices when several clock start transitions can be performed in a system state. Clock termination transitions are labeled with a clock name and represent the event of termination of the corresponding temporal delay.

As far as the problem of developing a process algebra with generally distributed delays is concerned, the fundamental issue is to understand how to define the operational semantics and the semantic reasoning, e.g. the definition of an adequate notion of bisimulation based congruence, for such an algebra.

In [39] we recognized that the problem of developing a semantics for a process algebra with generally distributed delays is essentially the problem of representing the execution of a temporal delay as the combination of a start and a termination event in such a way that pair of events derived from the execution of a clock are somehow related, e.g. by deriving a clock name for the delay. This exactly corresponds to consider classical *ST semantics* [81] for delays. In particular clock names are obtained if we use a technique for expressing ST semantics based on names and not, e.g., on pointers [52]. The entire first part of the thesis (Chapter 3) is dedicated to results concerning the decidability and axiomatizability of ST semantics and ST weak bisimulation in process algebras, which are used several times in the rest of the thesis. In particular we introduce three methods for expressing ST semantics: one based on static names, one based on dynamic names and one based on a stack technique. The semantics of the calculus of IGSMPs is based on the dynamic name technique, which allow equivalence between processes to be established by a simple variant of the notion of observational congruence of [122].

Another important issue to address when developing a semantics for a process algebra with generally distributed delays is how to interpret a choice "$\_ + \_$" between delays. Our claim (introduced in [41]) is that, while in the "Markovian world" the most natural solution is to adopt a *race policy* where the choice is solved in favor of the delay which terminates first, when we consider general distributions a convenient solution is to adopt a *preselection policy* where, first one of the delays is selected according to a probabilistic choice, and then the selected delay is executed. A justification for this claim can be found in the difference between the structure of GSMPs, where probabilistic choices are explicitely represented, and CTMCs, where probabilistic choices are implicitly expressed through races between exponential delays. On the other hand, supposed that we consider a process algebra where temporal delays with distribution f are represented by actions $<f, w>$, a preselection policy where delays are probabilistically chosen according to weights w is

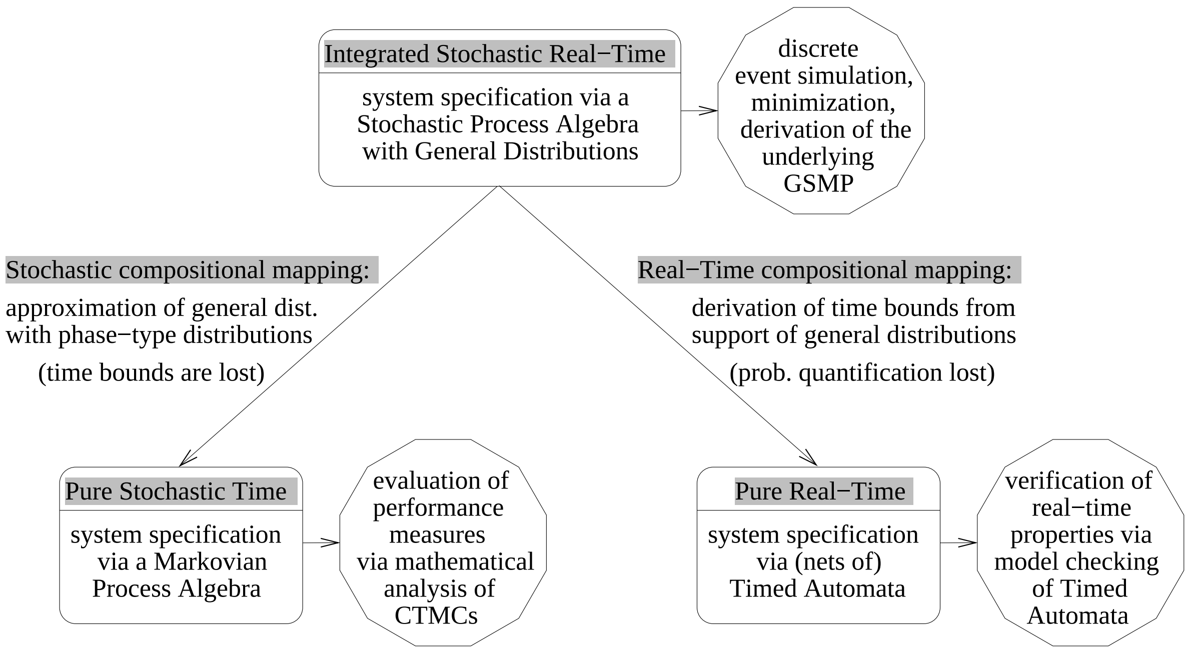


Figure 1.5: Stochastic Real-Time Integrated Approach

naturally obtained with ST semantics by associating the weight w with the transition representing the delay start. Based on this ideas in Chapter 7 we define a process algebra for deriving IGSMPs (we call it "calculus of IGSMPs" in analogy to [99]), which is simply an extension of a standard process algebra with the new action prefix $<f, w>$.

1.2.3 An Integrated Approach

In Fig. 1.5 we show how process algebra with generally distributed time can offer the possibility of an integrated approach for the modeling and analysis of Stochastic Real-Time concurrent/distributed systems. In particular in the second part of the thesis we will realize this integrated approach in the case of the calculus of IGSMPs. IGSMP specifications can be directly analyzed through standard discrete event simulation (see e.g. [82]) and by means of new techniques that we introduce: minimization via a notion of bisimulation based congruence which abstracts from internal system activities, and derivation of the underlying performance model in the form of a GSMP for IGSMPs which are complete both from the interactive and from the performance viewpoints. The performance of a GSMP can then be evaluated by using simulative or mathematical methodologies. *Exact mathematical analysis* can be carried out, in some cases, by employing the notion of *insensitivity* [121]: in an insensitive GSMP the duration distributions can be replaced with exponential distributions with the same mean, so the problem reduces to solve a CTMC.

Besides the possibility of performing combined analysis, we also present (in Chapter 8) formal techniques for compositionally deriving, from an IGSMP specification:

(i) A pure stochastic time (Markovian) specification in the form of a term of the calculus of *Interactive Weighted Markov Chains* (IWMCs), a probabilistic extension of Interactive Markov Chains [99] introduced in Chapter 4, by approximating general distributions with combinations of exponential distributions (the so called *phase-type* distributions). A consequence of this transformation is that all duration values for delays get probability greater than 0. Hence the information about time constraints (related to the real-time behavior of the system) is lost.

(ii) A pure real-time specification in the form of a net (a parallel composition) of *Interactive Timed Automata* (ITA), a variant of Timed Automata [11, 132] introduced in Chapter 5, by considering the support of general distributions, i.e. the set of time values that are given probability (density) greater than 0, and by turning probabilistic choices into non-deterministic choices. As a consequence the information related to the probabilistic-time behavior of the system is lost.

Interactive Weighted Markov Chains are basically an extension of Continuous Time Markov Chains with *action transitions*, representing the ability of the process to interact with other processes, and *probabilistic transitions*, representing probabilistic choices internally performed by the process. In particular Interactive Weighted Markov Chains extend Interactive Markov Chains of [99] with the capability of representing probabilistic choices through transitions labeled with *weights* [150]. From a complete IWMC specification a CTMC can be straightforwardly derived which can be analyzed with standard mathematical techniques and software tools (see e.g. [147]) for deriving performance measures of the system.

Interactive Timed Automata are a variant of classical Timed Automata [11, 132], where action executions, events enabled on the basis of clock constraints and clock reset events are expressed by means of separate transitions. The advantage of ITA with respect to existing Timed Automata is that action transitions can be dealt with separately from time-related transitions, hence making it easy to define, e.g., a notion of weak bisimulation. They can be analyzed via model checking for verifying real-time properties of systems with existing techniques and software tools (see e.g. [154, 116]).

The IWMC and the ITA obtained from an IGSMP specification are guaranteed to be consistent, in that represent different aspects of the same initial system specification. The technique leading to the derivation of the IWMC is particularly significant in that: (*i*) it shows process algebra to provide exactly the machinery necessary for approximating GSMPs with CTMCs through phase-type distributions, and (*ii*) it shows ST semantics to be the adequate semantics for generally distributed time in that approximation of activity durations with phase-type distributions is a form of action refinement.

Finally, we face the problem of dealing with “Zeno” processes (technically, weakly guarded processes) in interactive timed process algebras, so that our integrated approach can be extended also to these kind of processes. This problem is solved in Chapter 9 where we present a complete axiomatization over finite-state weakly guarded interactive timed processes.

1.3 Integrating Fixed and Probabilistic Durations via Discrete Time

An alternative methodology to integrate fixed and probabilistic durations in a single language is presented in the third part of the thesis (Chapter 10). Such a methodology is based on discrete time, i.e. time is not considered to be continuous (as in CTMCs and GSMPs) but is represented by a sequence of discrete steps as in Discrete Time Markov Chains (DTMCs). In particular, we make use of a discrete time process algebra expressing probabilistic external/internal choices, multi-way synchronization, and processes with different advancing speeds.

Such an algebra is obtained by employing a probabilistic asynchronous parallel operator (similar to that considered in [13]) and a CSP-like generative-reactive synchronization mechanism (similar to that considered in [155]). In particular, the semantics of processes is expressed by means of generative-reactive transition systems: an extension of DTMCs with action types expressing multi-way process synchronization through a mixture of the classical generative and reactive models of probability [80]. Moreover, differently from existing discrete time process algebras (see e.g. [112, 150, 118]), where parallel processes are executed in synchronous locksteps, the parallel operator that we adopt allows processes with different probabilistic advancing speeds (mean number of actions executed per time unit) to be modeled. Therefore we can specify different processes by means of actions with a different mean duration.

In Chapter 10 we show that, when evaluating steady state based performance measures which are expressible by attaching rewards to the actions of a process algebraic specification, the measure obtained with the algebra is exact even if advancing speeds of processes are considered not to be probabilistic, i.e. actions have a fixed duration. This result gives the possibility of analyzing systems which employ both probabilistic and fixed durations without resorting to generally distributed durations. Such analysis is very simple to be conducted in that, thanks to the usage of generative-reactive transition systems, the semantic model of a complete system is always a DTMC. Moreover, even if we restrict the attention to fixed durations only, the modeling technique presented in Chapter 10 is also very advantageous with respect to a classical synchronous approach, in that it avoids the explosion of the system state space when actions with largely different durations are employed in the system specification. The expressiveness of the calculus is shown in Chapter 10 by means of a non-trivial case study on multi-path routing which employs both fixed and probabilistic durations and actions with largely different durations (ranging from from half a microsecond to 20 milliseconds).

Finally it is worth noting that the axiomatization technique that we present in Chapter 10 (which is based on the introduction of an auxiliary operator) solves the open problem of cleanly axiomatizing action restriction in the generative model.

1.4 Gaining Expressivity: Modeling and Analyzing Real Systems

As a final issue, in the fourth part of the thesis we face the problem of developing a specification language which integrates the two approaches presented in the previous two sections (one based on continuous time and the other one based on discrete time) and which is endowed with additional expressive features, such as multi-level priorities and rewards, so that it is expressive enough to model real systems. This is confirmed by the modeling experience with the process algebra Extended Markovian Process Algebra (EMPA) [23, 8, 9, 28, 19], which has shown the necessity of considering all the mechanisms above in order to develop relevant case studies.

We solve this problem by first developing, in Chapter 11, $EMPA_{\rm gr}$, an extension of EMPA with the generative-reactive synchronization mechanism introduced in Chapter 10. $EMPA_{\rm gr}$ is a Markovian process algebra capable of representing both discrete and continuous time systems, where we can express: activities with exponentially distributed durations (which are not considered when doing discrete time specifications), activities with a discrete duration (which are assumed to take time zero in continuous time), probabilities, multi-level priorities and the generative-reactive synchronization mechanism. With respect to EMPA, $EMPA_{\rm gr}$ solves the open problem of obtaining equivalence compositionality in the presence of all these features, thus allowing for an efficient analysis of system specifications. Theoretically speaking, we argue that, when abandoning the classical nondeterministic setting by considering the features above, a natural solution is to break the symmetry of the roles of the processes participating in a synchronization. We accomplish this by distinguishing between master actions – the choice among which is carried out generatively according to their priorities/probabilities or exponentially distributed durations – and slave actions – the choice among which is carried out reactively according to their priorities/probabilities – and by imposing that a master action can synchronize with slave actions only. We show that such an asymmetric cooperation mechanism is natural and easy to understand by introducing the novel *cooperation structure model* which represents the structure of synchronizations in a system state. Then, we prove that the synchronization mechanism in $\mathrm{EMPA}_{\rm gr}$ is correct w.r.t. the cooperation structure model. As far as the proof of congruence of Markovian bisimulation equivalence is concerned, it is worth noting that, in Chapter 11, we introduce a new proof technique for showing congruence w.r.t. recursion in Markovian process algebras, which repairs some inaccuracies in the proofs previously proposed in the literature.

Then, in Chapter 12, we extend $EMPA_{\rm gr}$ with the capability of expressing performance measures at the algebraic level by attaching bonus and yield rewards to actions. In particular we show that the standard technique of expressing stationary and transient performance measures as weighted sums of state probabilities and transition frequencies through rewards can be imported into the process algebra framework. Technically speaking, if we denote by $n \in \mathbb{N}$ the number of performance measures of interest, we define a family of process algebras called $\mathrm{EMPA}_{\mathrm{gr}_n}$ which extend $EMPA_{\rm gr}$ by attaching sequences of yield and bonus rewards of length n to actions. We subsequently face the problem of defining a notion of equivalence for $\mathrm{EMPA}_{\mathrm{gr}_n}$ in the bisimulation style which is a congruence with respect to all the operators and which equates $\mathrm{EMPA}_{\mathrm{gr}_n}$

terms possessing the same functional, probabilistic, prioritized and exponentially timed behavior as well as the same performance measure values. We solve this problem through successive conceptual improvements, by subsequently defining three notions of equivalence satisfying the requirements above, each coarser than the previous one. First, we show how to define an equivalence which, whenever possible, aggregates yield rewards, but does not manipulate bonus rewards. Then, we show how to define a Markovian bisimulation congruence that aggregates bonus rewards as well. Finally, we show how to define a Markovian bisimulation congruence that allows yield rewards and bonus rewards to be used interchangeably up to suitable correcting factors, aiming at the introduction of a normal form for rewards. More precisely, we demonstrate that this is possible in the continuous time case, while it is not possible in the discrete time case because compositionality is lost.

Finally, in Chapter 13, we develop the algebra Generalized Semi-Markovian Process Algebra ($GSMPA$) which further extends the expressive power of $EMPA_{\mathrm{gr}}$ to activities with a generally distributed duration, which are given an ST semantics by means of the static name technique presented in Chapter 3. The fact of considering static names instead of dynamic names (as in IGSMPs) gives us the possibility of obtaining smaller initial semantic models for algebraic specifications and of decoupling the specification of system behavior from the specification of action durations which is convenient when dealing with general distributions. Moreover, we introduce the capability of representing contemporaneous termination of generally distributed durations in semantic models (situation that must instead be avoided in IGSMPs), thus having complete freedom when using general distributions in a system specification. The price to pay for using a static name technique instead of a dynamic name technique is a complex notion of bisimulation equivalence for GSMPA processes (due to the contemporaneous presence of probabilistic choices and static names).

1.5 Thesis Outline

This thesis is organized in parts as follows:

- Before the first part of the thesis we have Chapter 2 which contains some background about process algebras and stochastic processes taken from [122, 109, 14, 29, 136, 137, 95, 97, 117, 115, 149, 147, 144, 106, 121, 65, 93, 92, 94, 129, 110, 143, 91, 130].

- The first part of the thesis (including Chapter 3 only) contains the results concerning ST semantics in classical process algebras. In Chapter 3 we present three techniques for expressing ST semantics (namely the static name, the dynamic name and the stack technique) and we show that we can decide and axiomatize weak ST bisimulation over a wide class of processes which include recursion and action refinement. The material presented in this chapter has been published in [43].

- The second part of the thesis (Chapters 4-9), which contains the material regarding interactive timed modeling paradigms, is structured as follows:

– Chapter 4 introduces the calculus of Interactive Weighted Markov Chains (IWMCs) which is an extension of the calculus of Interactive Markov Chains [99] with probabilistic choices. We show an adapted notion of observational congruence to be a congruence for all the operators of the calculus and we present a sound axiomatization which is complete for finite-state algebra processes.

– Chapter 5 introduces Interactive Timed Automata (ITA) which are a variant of classical timed automata [11, 132], where action executions, events enabled on the basis of clock constraints and clock reset events are expressed by means of separate transitions. We introduce parallel composition and hiding of ITA and an adapted notion of weak bisimulation matching clock names of ITA which is based on the static name technique presented in Chapter 3. Moreover, aiming at defining a semantics for ITA, we introduce Interactive Prioritized Timed Transition Systems (IPTTSs) and we define parallel composition and hiding of IPTTSs and an adapted notion of weak bisimulation over IPTTSs. We then present a semantics which maps ITA onto IPTTSs which is shown to preserve equivalence and to be compositional with respect to both parallel composition and hiding.

– Chapter 6 introduces Interactive Generalized Semi-Markov Processes (IGSMPs). We start by defining Well-Named IGSMPs. We introduce parallel composition and hiding of Well-Named IGSMPs and an adapted notion of weak bisimulation over Well-Named IGSMPs which is based on the dynamic name technique presented in Chapter 3. Then, we show how to derive a performance model (a GSMP) from IGSMP which is complete both for interaction and for performance. Moreover, aiming at defining a semantics for IGSMPs, we introduce Interactive Stochastic Timed Transition Systems (ISTTSs) and we define parallel composition and hiding of ISTTSs and an adapted notion of weak bisimulation over ISTTSs. We then present a semantics which maps IGSMPs onto ISTTSs which is shown to preserve equivalence and to be compositional with respect to both parallel composition and hiding. Such a semantics is claimed, e.g. as opposed to the semantics presented in [66], to handle correctly non-deterministic choices in systems with generally-distributed durations. The material presented in this chapter has been published in [34] and [44].

– Chapter 7 introduces the calculus of Interactive Generalized Semi-Markov Processes which produces IGSMPs as semantic models of terms and where parallel composition and hiding are defined as in Chapter 6. We show an adapted notion of observational congruence (defined starting from weak bisimulation over IGSMPs introduced in Chapter 6) to be a congruence for all the operators of the calculus and we present a sound axiomatization which is complete for finite-state algebra processes. The material presented in this chapter has been published in [44].

– Chapter 8 presents the two formal mappings from IGSMPs to IWMCs and ITA. ITA are obtained as the pure real-time model of an IGSMP by considering the support of generally distributed durations and by turning probabilistic choices into non-deterministic choices. IWMCs are obtained as the pure Markovian model of an IGSMP by approximating generally distributed durations in IGSMPs with phase-type distributions. Some preliminary ideas on which this chapter is based have been published in [32].

  - Chapter 9 faces the problem of dealing with weakly guarded recursion in interactive timed processes. In particular we develop a complete axiomatization over finite-state weakly guarded interactive timed processes. The material presented in this chapter has been published in [45] and is taken also from [46].

- The third part of the thesis (including Chapter 10 only) contains the material concerning the algebra of discrete time generative-reactive processes. In Chapter 10 we introduce the syntax and semantics of the generative-reactive calculus with advancing speeds, we show an adapted notion of probabilistic bisimulation equivalence to be a congruence for all the operators of such a calculus and we present a sound axiomatization which is complete for finite-state algebra processes. Moreover we show that, when evaluating steady state based performance measures, such an approach provides an exact solution even if advancing speeds are considered to be exact instead of probabilistic, and we present a non-trivial case study on multi-path routing. The material presented in this chapter has been published in [33].

- The fourth part of the thesis (Chapters 11-13), which is dedicated to integrating the approaches presented in the second and third part of the thesis into a single language and to furthermore add expressive features, is structured as follows:

  - Chapter 11 introduces $EMPA_{\mathrm{gr}}$, an extension of the existing algebra Extended Markovian Process Algebra (EMPA) with the generative-reactive synchronization mechanism introduced in Chapter 10. We define the syntax and semantics $\mathrm{EMPA}_{\mathrm{gr}}$, we show an adapted notion of probabilistic bisimulation equivalence to be a congruence for all the operators of such a calculus and we present a sound axiomatization which is complete for finite-state algebra processes. Moreover, we show the generative-reactive synchronization mechanism at the basis of $\mathrm{EMPA}_{\mathrm{gr}}$ to be natural and easy to understand by introducing the novel cooperation structure model, representing the structure of process synchronizations in a system state, and by proving that the synchronization mechanism in $\mathrm{EMPA}_{\mathrm{gr}}$ is correct w.r.t. such a model. The material presented in this chapter has been published in [36] and [38] and is also taken from [37].

  - Chapter 12 introduces $\mathrm{EMPA}_{\mathrm{gr}_n}$, which extends $EMPA_{\mathrm{gr}}$ with the capability of expressing n performance measures at the algebraic level by attaching bonus and yield rewards [110] to actions. After defining the syntax and semantics of $\mathrm{EMPA}_{\mathrm{gr}}$, we introduce three extensions of the notion of bisimulation equivalence introduced in Chapter 11 (each coarser than the previous one) accounting for yield and bonus rewards; we show that they are all congruences for $\mathrm{EMPA}_{\mathrm{gr}_n}$ which preserve the specified performance measures; and, for each of them, we give a sound axiomatization which is complete over finite $\mathrm{EMPA}_{\mathrm{gr}_n}$ terms. The material presented in this chapter has been published in [21] and in [22].

- Chapter 13 introduces the process algebra Generalized Semi-Markovian Process Algebra (GSMPA) which further extends $EMPA_{\mathrm{gr}}$ to activities with a generally distributed duration by using ST semantics for such actions. After defining the syntax and semantics of GSMPA, we introduce a notion of bisimulation equivalence over GSMPA process, which is based on the static name technique presented in Chapter 3, and we show it to be a congruence with respect to all the operators of GSMPA. The material presented in this chapter has been published in [39] and is also taken from [40].

- In the fifth part of the thesis (including Chapter 14 only) we report some concluding remark concerning ongoing and future work.

- The thesis ends with the following four appendices including technical machinery.

  - Appendix A includes technical machinery for Chapter 3: the definition of strong guarded processes for the two process algebras considered in Chapter 3, the proof of equivalence of the dynamic name and static name techniques, and the proof of equivalence of the stack and static name techniques.
  - Appendix B includes technical machinery for Chapter 11: the proof of consistency of the $EMPA_{\mathrm{gr}}$ semantics with the cooperation structure model, the proof of congruence of $EMPA_{\mathrm{gr}}$ bisimulation equivalence, and the proof of soundness and completeness of its axiomatization.
  - Appendix C includes technical machinery for Chapter 12: the proof of congruence of the Markovian bisimulation equivalence which aggregates yield rewards and the proof of congruence of the Markovian bisimulation equivalence which aggregates yield and bonus rewards.
  - Appendix D includes technical machinery for Chapter 13: the proof of congruence of GSMPA bisimulation equivalence with respect to all the operators of the algebra and with respect to recursive definitions.

Chapter 2

Background

In this chapter we provide the background which is necessary to understand the remainder of the thesis. This chapter is organized as follows. In Sect. 2.1 we present process algebras. In Sect. 2.2 we introduce continuous time and discrete time Markov chains, semi-Markov processes and Generalized semi-Markov processes.

2.1 Process Algebras

In this section we recall the theory of process algebras and we introduce a simple process algebra based on a combination of the operators of well known process algebras such as CCS [122], CSP [109] that will serve as a basis for the development of the process calculi introduced in the rest of the thesis. In Sect. 2.1.1 we present the operators of the process algebra. In Sect. 2.1.2 we introduce rooted labeled transition systems as they are used as semantic models for algebraic terms. In Sect. 2.1.3 we provide the operational semantics for the process algebra. Finally, in Sect. 2.1.4 we recall the notion of bisimulation equivalence together with its equational and logical characterizations.

2.1.1 Algebraic Operators

Process algebras [122, 109, 14, 29] are algebraic languages which support the compositional description of concurrent systems and the formal verification of their properties. The basic elements of any process algebra are its actions, which represent activities carried out by the systems being modeled, and its operators (among which a parallel composition operator), which are used to compose algebraic descriptions. In this section we introduce a simple process algebra based on a combination of the operators of CCS [122] and CSP [109].

Let Act be a set of *actions* ranged over by $a, b, \dots$ which contains a distinguished action τ denoting *unobservable* activities. Furthermore, let $Const$ be a set of *constants* ranged over by $A, B, \dots$ and $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of *action relabeling functions* ranged over by $\varphi, \varphi', \dots$.

Definition 2.1 *The set $\mathcal{L}$ of process terms is generated by the following syntax*

$$E ::= \underline{0} \mid a.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq Act - \{\tau\}$*. The set* $\mathcal{L}$ *will be ranged over by* $E, F, \ldots$*.* ■

The *null term* "$\underline{0}$" is the term that cannot execute any action.

The *action prefix operator* "$a.\_$" denotes the sequential composition of an action and a term. Term $a.E$ can execute action a and then behaves as term E.

The *abstraction operator* "$\_/L$" make actions unobservable. Term E/L behaves as term E except that each executed action a is hidden, i.e. turned into τ, whenever $a \in L$. This operator provides a means to encapsulate or ignore information.

The *relabeling operator* "$\_[\varphi]$" changes actions. Term $E[\varphi]$ behaves as term E except that each executed action a becomes $\varphi(a)$. This operator provides a means to obtain more compact algebraic descriptions, since it enables the reuse of terms in situations demanding the same functionality up to action names.

The *alternative composition operator* "$\_ + \_$" expresses a nondeterministic choice between two terms. Term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is executed.

The *parallel composition operator* "$\_ \parallel_S \_$" expresses the concurrent execution of two terms according to the following synchronization discipline: two actions can synchronize if and only if they have the same observable type in S, which becomes the resulting type. Term $E_1 \parallel_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S if the requirement above is met.

In order to avoid ambiguities, we assume the binary operators to be left associative and we introduce the following operator precedence relation: abstraction = relabeling > action prefix > alternative composition > parallel composition. Parentheses can be used to alter associativity and precedence. Moreover, given a term E, we denote by $sort(E)$ the set of actions occurring in the action prefix operators of E.

Finally, let partial function $Def : Const \twoheadrightarrow \mathcal{L}$ be a set of *constant defining equations* of the form $A \triangleq E$. As an example of application of operators and defining equations, we consider a two place buffer, which can be modeled as the parallel composition of two communicating places as follows:

$$\begin{array}{rcl} Buffer & \triangleq & Place_1 \parallel_{\{move\}} Place_2 \\ Place_1 & \triangleq & accept.move.Place_1 \\ Place_2 & \triangleq & move.release.Place_2 \end{array}$$

The first place accepts items from the outside and moves them to the second place, while the second place receives items from the first place (through the synchronization on $move$) and gives them to the outside.

It is worth noting that recursive definitions like "$Place_1 \triangleq accept.move.Place_1$" can also be denoted by means of a *recursion operator* "$recX.\_$", where the meaning of "$recX.E$" is the same as that of "$A \triangleq E\{A/X\}$", with "$E\{A/X\}$" being the term obtained from E by replacing A for X. Hence, for example the behavior of $Place_1$ would be equivalently represented by $recX.accept.move.X$. Usually, the use of constants is suitable when doing specifications, while the use of operator "$recX.\_$" can be preferred when dealing with axiomatizations (see Sect. 2.1.4). The two constructs are shown to be completely equivalent in [122].

In order to guarantee the correctness of recursive definitions, we restrict ourselves to terms that are closed and guarded w.r.t. Def, i.e. those terms such that each constant occurring in them has a defining equation and appears in the context of an action prefix operator. This rules out undefined constants, meaningless definitions such as $A \triangleq A$, and infinitely branching terms such as $A \triangleq a.\underline{0} \,\|_{\emptyset}\, A$ whose executable actions cannot be computed in finite time. Let us denote by "$\_ \equiv \_$" the syntactical equality between terms and by "$\_\, st\, \_$" the relation subterm-of.

Definition 2.2 *The term $E\langle A := E'\rangle$ obtained from $E \in \mathcal{L}$ by replacing each occurrence of A with E', where $A \triangleq E' \in Def$, is defined by induction on the syntactical structure of E as follows:*

$$
\begin{array}{rcl}
\underline{0}\langle A := E'\rangle & \equiv & \underline{0} \\
(a.E)\langle A := E'\rangle & \equiv & a.E\langle A := E'\rangle \\
E/L\langle A := E'\rangle & \equiv & E\langle A := E'\rangle/L \\
E[\varphi]\langle A := E'\rangle & \equiv & E\langle A := E'\rangle[\varphi] \\
(E_1 + E_2)\langle A := E'\rangle & \equiv & E_1\langle A := E'\rangle + E_2\langle A := E'\rangle \\
(E_1 \,\|_S\, E_2)\langle A := E'\rangle & \equiv & E_1\langle A := E'\rangle \,\|_S\, E_2\langle A := E'\rangle \\
B\langle A := E'\rangle & \equiv & \begin{cases} E' & \text{if } B \equiv A \\ B & \text{if } B \not\equiv A \end{cases}
\end{array}
$$
■

Definition 2.3 *The set of terms obtained from $E \in \mathcal{L}$ by repeatedly replacing constants by the right hand side terms of their defining equations in Def is defined by*

$$
Subst_{Def}(E) = \bigcup_{n \in \mathbb{N}} Subst^n_{Def}(E)
$$

where

$$
Subst^n_{Def}(E) = \begin{cases} \{E\} & \text{if } n = 0 \\ \{F \in \mathcal{L} \mid F \equiv G\langle A := E'\rangle \wedge G \in Subst^{n-1}_{Def}(E) \wedge A\, st\, G \wedge A \triangleq E' \in Def\} & \text{if } n > 0 \end{cases}
$$
■

Definition 2.4 *The set of constants occurring in $E \in \mathcal{L}$ w.r.t. Def is defined by*

$$
Const_{Def}(E) = \{A \in Const \mid \exists F \in Subst_{Def}(E).\, A\, st\, F\}
$$
■

Definition 2.5 *A term $E \in \mathcal{L}$ is closed and guarded w.r.t. Def if and only if for all $A \in Const_{Def}(E)$*

- *A is equipped in Def with defining equation $A \triangleq E'$, and*
- *there exists $F \in Subst_{Def}(E')$ such that, whenever an instance of a constant B satisfies $B\, st\, F$, then the same instance satisfies $B\, st\, a.G\, st\, F$.*

We denote by $\mathcal{G}$ the set of terms in $\mathcal{L}$ that are closed and guarded w.r.t. Def. ■

2.1.2 Rooted Labeled Transition Systems

In this section we present the definition of labeled transition system together with some related notions [136]. These mathematical models, which are essentially state transition graphs, are commonly adopted when defining the semantics for a process algebra in the operational style [137].

Definition 2.6 *A rooted labeled transition system (LTS) is a tuple*

$$(S, U, \longrightarrow, s_0)$$

where:

- S *is a set whose elements are called states.*
- U *is a set whose elements are called labels.*
- $\longrightarrow \subseteq S \times U \times S$ *is called transition relation.*
- $s_0 \in S$ *is called the initial state.* ■

In the graphical representation of a LTS, states are drawn as black dots and transitions are drawn as arrows between pairs of states with the appropriate labels; the initial state is pointed to by an unlabeled arrow.

Below we recall two notions of equivalence defined for LTSs. The former, isomorphism, relates two LTSs if they are structurally equal. This is formalized by requiring the existence of a label preserving relation which is bijective, i.e. a bijection between the two state spaces such that any pair of corresponding states have identically labeled transitions toward any pair of corresponding states. The latter equivalence, bisimilarity, is coarser than isomorphism as it relates also LTSs which are not structurally equal provided that they are able to simulate each other. This is formalized by requiring the existence of a label preserving relation between the two state spaces which is not necessarily bijective.

Definition 2.7 *Let* $Z_k = (S_k, U, \longrightarrow_k, s_{0k})$, $k \in \{1, 2\}$, *be two LTSs.*

- Z_1 *is isomorphic to* Z_2 *if and only if there exists a bijection* $\beta : S_1 \longrightarrow S_2$ *such that:*
  - $\beta(s_{01}) = s_{02}$*;*
  - *for all* $s, s' \in S_1$ *and* $u \in U$

$$s \xrightarrow{u}_1 s' \iff \beta(s) \xrightarrow{u}_2 \beta(s')$$

- Z_1 *is bisimilar to* Z_2 *if and only if there exists a relation* $\mathcal{B} \subseteq S_1 \times S_2$ *such that:*
  - $(s_{01}, s_{02}) \in \mathcal{B}$*;*
  - *for all* $(s_1, s_2) \in \mathcal{B}$ *and* $u \in U$
    - * *whenever* $s_1 \xrightarrow{u}_1 s'_1$, *then* $s_2 \xrightarrow{u}_2 s'_2$ *and* $(s'_1, s'_2) \in \mathcal{B}$*;*
    - * *whenever* $s_2 \xrightarrow{u}_2 s'_2$, *then* $s_1 \xrightarrow{u}_1 s'_1$ *and* $(s'_1, s'_2) \in \mathcal{B}$. ■

2.1.3 Operational Semantics

The semantics of our example process algebra is defined in this section following the structured operational approach [137]. This means that inference rules are given for each operator which define an abstract interpreter for the language. More precisely, the semantics is defined through the transition relation $\longrightarrow$ which is the least subset of $\mathcal{G} \times Act \times \mathcal{G}$ satisfying the inference rules in Table 2.1. Such rules, which formalize the meaning of each operator informally given in Sect. 2.1.1, yield LTSs where states are in correspondence with terms and transitions are labeled with actions. Given a term E, the outgoing transitions of the state corresponding to E are generated by proceeding by induction on the syntactical structure of E applying at each step the appropriate semantic rule until an action prefix operator is encountered or no rule can be used. This can be done in finite time because of the restriction to closed and guarded terms.

As an example, when computing the transitions for the state associated with *Buffer*, we apply first of all one of the inference rules for the parallel composition operator, say the first one. Then $Place_1$ is examined and the axiom for the action prefix operator is applied thus determining *accept* as executed action and $move.Place_1$ as term associated with the derivative state. The inference is concluded by returning to *Buffer* and determining *accept* as executed action (since $accept \notin \{move\}$) and $move.Place_1 \parallel_{\{move\}} Place_2$ as term associated with the derivative state. This is the only outgoing transition for the considered state since initially applying one of the other inference rules for the parallel composition operator results in a violation of their side conditions.

We recall that the abstraction operator, the relabeling operator, and the parallel composition operator are called static operators because they appear also in the derivative term of the consequence of the related semantic rules. By contrary, the action prefix operator and the alternative composition operator are classified as dynamic operators. We say that a term is sequential if every occurrence of static operators is within the scope of an action prefix operator. It is worth noting that the LTS underlying $E \in \mathcal{G}$ is finite if all of the subterms of terms in $Subst_{Def}(E)$ whose outermost operator is static contains no recursive constants.

Definition 2.8 *The operational interleaving semantics of $E \in \mathcal{G}$ is the LTS*

$$[\![E]\!] = (S_E, Act, \longrightarrow_E, E)$$

where:

- *S_E is the least subset of $\mathcal{G}$ such that:*
  - *$E \in S_E$;*
  - *if $E_1 \in S_E$ and $E_1 \xrightarrow{a} E_2$, then $E_2 \in S_E$.*
- *$\longrightarrow_E$ is the restriction of $\longrightarrow$ to $S_E \times Act \times S_E$.* ■

$$a.E \xrightarrow{a} E$$

$$\frac{E \xrightarrow{a} E'}{E/L \xrightarrow{a} E'/L} \text{ if } a \notin L \qquad \frac{E \xrightarrow{a} E'}{E/L \xrightarrow{\tau} E'/L} \text{ if } a \in L$$

$$\frac{E \xrightarrow{a} E'}{E[\varphi] \xrightarrow{\varphi(a)} E'[\varphi]}$$

$$\frac{E_1 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'} \qquad \frac{E_2 \xrightarrow{a} E'}{E_1 + E_2 \xrightarrow{a} E'}$$

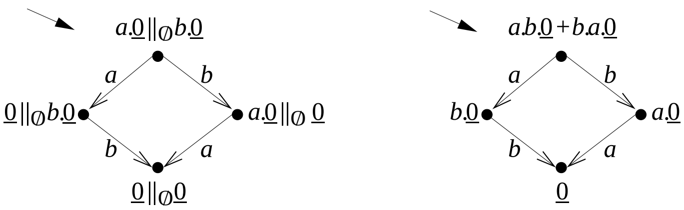
$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1' \parallel_S E_2} \text{ if } a \notin S \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1 \parallel_S E_2'} \text{ if } a \notin S$$

$$\frac{E_1 \xrightarrow{a} E_1' \quad E_2 \xrightarrow{a} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a} E_1' \parallel_S E_2'} \text{ if } a \in S$$

$$\frac{E \xrightarrow{a} E'}{A \xrightarrow{a} E'} \text{ if } A \triangleq E$$

Table 2.1: Operational semantics for the example process algebra

We talk about interleaving semantics [122] because every parallel computation is represented in the LTS by means of alternative sequential computations obtained by interleaving actions executed by concurrent processes. As an example, note that $[\![a.\underline{0} \parallel_{\emptyset} b.\underline{0}]\!]$ is isomorphic to $[\![a.b.\underline{0} + b.a.\underline{0}]\!]$:



2.1.4 Bisimulation Equivalence

In this section we define a notion of equivalence for the example process algebra and we provide its equational and logical characterizations. The purpose of such an equivalence is to relate those terms representing systems which, though structurally different, behave the same from the point of view of an external observer. Following [122], the equivalence is given in the bisimulation style.

Definition 2.9 *A relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a strong bisimulation if and only if, whenever $(E_1, E_2) \in \mathcal{B}$, then for all $a \in Act$:*

- *whenever $E_1 \xrightarrow{a} E_1'$, then $E_2 \xrightarrow{a} E_2'$ and $(E_1', E_2') \in \mathcal{B}$;*
- *whenever $E_2 \xrightarrow{a} E_2'$, then $E_1 \xrightarrow{a} E_1'$ and $(E_1', E_2') \in \mathcal{B}$.* ■

The union of all the strong bisimulations can be shown to be an equivalence relation which coincides with the largest strong bisimulation. Such a relation, denoted $\sim$, is defined to be the *strong bisimulation equivalence* and can be shown to be a congruence w.r.t. all the operators as well as recursive definitions. As an example, if $E_1 \sim E_2$ then $E_1 \,\|_S\, F \sim E_2 \,\|_S\, F$ for all F and S. This allows for the compositional manipulation of algebraic terms: given a term, replacing one of its subterms with a bisimilar subterm does not alter the whole meaning because the newly generated term is still bisimilar to the original one.

The effect of $\sim$ can be better seen through its equational characterization. Given that $\sim$ is a congruence, we present in Table 2.2 a set $\mathcal{A}$ of axioms for the set of nonrecursive terms in $\mathcal{G}$ on which a deductive system $Ded(\mathcal{A})$ can be built which satisfies reflexivity ($E = E$), symmetry ($E_1 = E_2 \Longrightarrow E_2 = E_1$), transitivity ($E_1 = E_2 \wedge E_2 = E_3 \Longrightarrow E_1 = E_3$), and substitutivity ($E_{1,1} = E_{2,1} \wedge \ldots \wedge E_{1,n} = E_{2,n} \Longrightarrow op(E_{1,1}, \ldots, E_{1,n}) = op(E_{2,1}, \ldots, E_{2,n})$ for each n-ary operator op of the algebra) [95]. It can be shown that the axiomatization above, which basically allows terms to be rewritten in such a way that only action prefix and alternative composition operators occur, is sound and complete w.r.t. $\sim$, which means that in $\mathcal{A}$ it can be proved $E_1 = E_2$ if and only if $E_1 \sim E_2$. We observe that axiom $\mathcal{A}_{11}$, also known as the expansion law, is the equational characterization of the notion of interleaving.

It is worth noting that *equivalence checking* is one of the major verification techniques in the process algebra setting. Given a term representing a system, another simpler term is considered which is taken to be a specification of the system and the purpose is to verify whether the given term is equivalent to the specification. As an example, the two place buffer may be specified as follows by considering the possible sequences of actions

$$\begin{array}{rcl}
BufferSpec & \triangleq & accept.BufferSpec_1 \\
BufferSpec_1 & \triangleq & move.(accept.BufferSpec_2 + release.BufferSpec) \\
BufferSpec_2 & \triangleq & release.BufferSpec_1
\end{array}$$

and it can be shown that $Buffer \sim BufferSpec$, which means that the parallel composition of two interacting places described by *Buffer* complies with the specification. In the case of a similar technique known as

$$
\begin{array}{lrcl}
(\mathcal{A}_1) & (E_1 + E_2) + E_3 & = & E_1 + (E_2 + E_3) \\
(\mathcal{A}_2) & E_1 + E_2 & = & E_2 + E_1 \\
(\mathcal{A}_3) & E + \underline{0} & = & E \\
(\mathcal{A}_4) & E + E & = & E \\
\\
(\mathcal{A}_5) & \underline{0}/L & = & \underline{0} \\
(\mathcal{A}_6) & (a.E)/L & = & \begin{cases} a.E/L & \text{if } a \notin L \\ \tau.E/L & \text{if } a \in L \end{cases} \\
(\mathcal{A}_7) & (E_1 + E_2)/L & = & E_1/L + E_2/L \\
\\
(\mathcal{A}_8) & \underline{0}[\varphi] & = & \underline{0} \\
(\mathcal{A}_9) & (a.E)[\varphi] & = & \varphi(a).E[\varphi] \\
(\mathcal{A}_{10}) & (E_1 + E_2)[\varphi] & = & E_1[\varphi] + E_2[\varphi] \\
\\
(\mathcal{A}_{11}) & \sum\limits_{i \in I_1} a_i.E_i \parallel_S \sum\limits_{i \in I_2} a_i.E_i & = & \sum\limits_{j \in J_1} a_j.(E_j \parallel_S \sum\limits_{i \in I_2} a_i.E_i) + \\
& & & \sum\limits_{j \in J_2} a_j.(\sum\limits_{i \in I_1} a_i.E_i \parallel_S E_j) + \\
& & & \sum\limits_{k \in K_1} \sum\limits_{h \in H_k} a_k.(E_k \parallel_S E_h) + \\
& & & \sum\limits_{k \in K_2} \sum\limits_{h \in H_k} a_k.(E_h \parallel_S E_k) \\
\\
& \text{where } I_1 \cap I_2 & = & \emptyset \\
& J_1 & = & \{i \in I_1 \mid a_i \notin S\} \\
& J_2 & = & \{i \in I_2 \mid a_i \notin S\} \\
& K_1 & = & \{k \in I_1 \mid a_k \in S \wedge \exists h \in I_2.\, a_h = a_k\} \\
& K_2 & = & \{k \in I_2 \mid a_k \in S \wedge \exists h \in I_1.\, a_h = a_k\} \\
& H_k & = & \begin{cases} \{h \in I_2 \mid a_h = a_k\} & \text{if } k \in K_1 \\ \{h \in I_1 \mid a_h = a_k\} & \text{if } k \in K_2 \end{cases}
\end{array}
$$

Table 2.2: Axioms for strong bisimulation equivalence

preorder checking, the specification is still a term but it is treated as the minimal requirement to be met, owing to the fact that it can contain don't care points. Finally, we cite a different verification technique known as *model checking*, which is based on formalizing safety and liveness properties by means of modal or temporal logic formulas and verifying whether a given term satisfies such formulas. As an example, the syntax of Hennessy-Milner logic [97] formulas is given by

$$\Phi ::= tt \mid f\!f \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid [a]\Phi \mid \langle a \rangle \Phi$$

and their semantics is given by the following satisfaction relation

$$\begin{array}{rcl}
E & \models & tt \\
E & \not\models & f\!f \\
E \models \neg\Phi & \iff & E \not\models \Phi \\
E \models \Phi_1 \wedge \Phi_2 & \iff & E \models \Phi_1 \wedge E \models \Phi_2 \\
E \models \Phi_1 \vee \Phi_2 & \iff & E \models \Phi_1 \vee E \models \Phi_2 \\
E \models \langle a \rangle \Phi & \iff & \exists E'.\, E \xrightarrow{a} E' \wedge E' \models \Phi \\
E \models [a]\Phi & \iff & \forall E'.\, E \xrightarrow{a} E' \Longrightarrow E' \models \Phi
\end{array}$$

As far as modal operators are concerned, $E \models \langle a \rangle \Phi$ means that it is possible for E to execute a and evolve into a state where Φ is satisfied, while $E \models [a]\Phi$ means that if E executes a then the derivative term necessarily satisfies Φ. The reason why we have recalled this particular modal logic is that it is the logical characterization of $\sim$, i.e. it can be proved that $E_1 \sim E_2$ if and only if E_1 and E_2 satisfy the same subset of Hennessy-Milner logic formulas.

We conclude by observing that bisimulation equivalence treats τ actions in the same way as observable actions. Since τ actions cannot be seen by an external observer, a less strong notion of bisimulation equivalence called *weak bisimulation equivalence* ($\approx$) can be set up which abstracts from τ actions and still is a congruence w.r.t. the parallel composition operator [122]. Instead of giving the definition of such an equivalence, we only provide the axioms to add to those in Table 2.2 in order to obtain its equational characterization:

$$\begin{array}{rcl}
\tau.E & = & E \\
a.\tau.E & = & a.E \\
E + \tau.E & = & \tau.E \\
a.(E_1 + \tau.E_2) + a.E_2 & = & a.(E_1 + \tau.E_2)
\end{array}$$

Since any congruence w.r.t. the parallel composition operator is useful for minimization purposes, it turns out that $\approx$ is more convenient than $\sim$. As an example, moving items from the first place to the second place in the two place buffer can be regarded as an internal action, so the buffer may be specified as follows by considering the possible sequences of observable actions

$$\begin{array}{rcl}
\mathit{BufferSpec}' & \stackrel{\Delta}{=} & \mathit{accept}.\mathit{BufferSpec}'_1 \\
\mathit{BufferSpec}'_1 & \stackrel{\Delta}{=} & \mathit{accept}.\mathit{BufferSpec}'_2 + \mathit{release}.\mathit{BufferSpec}' \\
\mathit{BufferSpec}'_2 & \stackrel{\Delta}{=} & \mathit{release}.\mathit{BufferSpec}'_1
\end{array}$$

and it can be shown that $\mathit{Buffer}/\{\mathit{move}\} \approx \mathit{BufferSpec}'$ with $\mathit{BufferSpec}'$ having fewer states than $\mathit{BufferSpec}$.

2.2 Stochastic Processes

In this section we recall the theory of Stochastic Processes. In Sect. 2.2.1 we propose the mathematical model given by probabilistically rooted labeled transition systems which provide a graph theoretic representation of Markov chains and semi-Markov processes. In Sect. 2.2.2 and 2.2.3 we present continuous time and discrete time Markov chains, respectively. In Sect. 2.2.4 we introduce the notion of ordinary lumping. In Sect. 2.2.5 we present semi-Markov processes. In Sect. 2.2.6 we present generalized semi-Markov processes. In Sect. 2.2.7 we present phase-type distributions. In Sect. 2.2.8 we explain how to specify performance measures via reward structures. Finally, in Sect. 2.2.9 we recall queueing systems.

2.2.1 Probabilistically Rooted Labeled Transition Systems

In this section we propose an extension of LTSs where the initial state is replaced by a probability mass function which determines for each state the probability that it is the initial one. Moreover, a holding time function is added in order to distinguish among vanishing states, tangible states, and absorbing states. Vanishing states have zero sojourn time and the labels of their outgoing transitions are the execution probabilities of the transitions themselves. Tangible states have a positive sojourn time and the labels of their outgoing transitions are the execution rates of the transitions themselves. Absorbing states have an infinite sojourn time because they have no outgoing transitions. By means of this interpretation, we are able to represent discrete time Markov chains (where tangible states are absent), continuous time Markov chains (where vanishing states are absent), and semi-Markov processes (where both vanishing and tangible states coexist).

Definition 2.10 *A probabilistically rooted labeled transition system (p-LTS) is a tuple*

$$(S, \mathbb{R}_+, \longrightarrow, P, H)$$

where:

- *S and $\longrightarrow$ are defined as for a LTS.*
- *$P : S \longrightarrow \mathbb{R}_{[0,1]}$, called initial state probability function, is such that $\sum_{s \in S} P(s) = 1$.*
- *$H : S \longrightarrow \{v, t, a\}$, called holding time function, is such that* [1]

$$\begin{array}{lcll} H(s) & = & t & \text{if } \sum\{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s' \in S.\, s \xrightarrow{\lambda} s' \,|\!\} \neq 1 \\ H(s) & \in & \{t, v\} & \text{if } \sum\{\!|\, \lambda \in \mathbb{R}_+ \mid \exists s' \in S.\, s \xrightarrow{\lambda} s' \,|\!\} = 1 \\ H(s) & = & a & \text{if } \nexists s' \in S.\, \nexists \lambda \in \mathbb{R}_+ .\, s \xrightarrow{\lambda} s' \end{array}$$

■

[1] $\{\!|$ and $|\!\}$ denote multiset delimiters.

In the graphical representation of a p-LTS, states and transitions are drawn as in a LTS and each state is labeled with its initial state probability (unless it is zero) as well as its holding time (unless it is clear from the context).

The notions of equivalence for p-LTSs (p-isomorphism and p-bisimilarity) carry over from the corresponding notions for LTSs. In particular, p-bisimilarity is developed according to [117], so it considers two p-LTSs to be equivalent if any pair of corresponding states have the same aggregated label to reach the same equivalence class of states. We recall that, given an equivalence relation $\mathcal{R}$ over set S, the set of equivalence classes of S w.r.t. $\mathcal{R}$ is denoted $S/\mathcal{R}$.

Definition 2.11 *Let $Z_k = (S_k, \mathbb{R}_+, \longrightarrow_k, P_k, H_k)$, $k \in \{1, 2\}$, be two p-LTSs.*

- *Z_1 is p-isomorphic to Z_2 if and only if there exists a bijection $\beta : S_1 \longrightarrow S_2$ such that:*
  - *for all $s \in S_1$*

$$\begin{aligned} P_1(s) &= P_2(\beta(s)) \\ H_1(s) &= H_2(\beta(s)) \end{aligned}$$

  - *for all $s, s' \in S_1$ and $\lambda \in \mathbb{R}_+$*

$$s \xrightarrow{\lambda}_1 s' \iff \beta(s) \xrightarrow{\lambda}_2 \beta(s')$$

- *Z_1 is p-bisimilar to Z_2 if and only if there exists a relation $\mathcal{B} \subseteq S_1 \times S_2$ with reflexive, symmetric and transitive closure $\mathcal{B}' \subseteq (S_1 \cup S_2) \times (S_1 \cup S_2)$ such that:*
  - *for all $C \in (S_1 \cup S_2)/\mathcal{B}'$*

$$\sum_{s \in C \cap S_1} P_1(s) = \sum_{s \in C \cap S_2} P_2(s)$$

  - *whenever $(s_1, s_2) \in \mathcal{B}$, then*

$$H_1(s_1) = H_2(s_2)$$

  - *whenever $(s_1, s_2) \in \mathcal{B}$, then for all $C \in (S_1 \cup S_2)/\mathcal{B}'$*

$$\sum \{\!| \lambda \in \mathbb{R}_+ \mid \exists s_1' \in C \cap S_1 .\, s_1 \xrightarrow{\lambda}_1 s_1' |\!\} = \sum \{\!| \lambda \in \mathbb{R}_+ \mid \exists s_2' \in C \cap S_2 .\, s_2 \xrightarrow{\lambda}_2 s_2' |\!\} \quad \blacksquare$$

2.2.2 Continuous Time Markov Chains

A Markov chain is a stochastic process with discrete state space which is characterized by the memoryless property: the probability of being in a certain state at a given instant depends only on the current state instead of the whole history of visited states. Below we consider the continuous time case.

Definition 2.12 *A continuous time Markov chain (CTMC) [115] is a continuous time stochastic process* $X = \{X(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *with discrete state space* S_X *such that, for each* $n \in \mathbb{N}_+$, $i_0, \dots, i_{n-1}, i_n \in S_X$, $t_0, \dots, t_{n-1}, t_n \in \mathbb{R}_{[0,\infty[}$ *where* $t_0 < \dots < t_{n-1} < t_n$, *it turns out*

$$\Pr\{X(t_n) = i_n \mid X(t_{n-1}) = i_{n-1} \wedge \dots \wedge X(t_0) = i_0\} = \Pr\{X(t_n) = i_n \mid X(t_{n-1}) = i_{n-1}\} \qquad \blacksquare$$

Definition 2.13 *Let* X *be a CTMC.*

- *The transition matrix of* X *from time* $t \in \mathbb{R}_{[0,\infty[}$ *to time* $t' > t$ *is matrix* $\mathbf{P}_X(t, t')$ *defined by*

$$\mathbf{P}_X(t, t') = [\Pr\{X(t') = j \mid X(t) = i\}]_{i,j \in S_X}$$

- *The infinitesimal generator of* X *at time* $t \in \mathbb{R}_{[0,\infty[}$ *is matrix* $\mathbf{Q}_X(t)$ *defined by*

$$\mathbf{Q}_X(t) = [q_{i,j}(t)]_{i,j \in S_X} = \lim_{\Delta t \to 0} \frac{\mathbf{P}_X(t, t + \Delta t) - \mathbf{I}}{\Delta t}$$

  where $\mathbf{I}$ *is the identity matrix.*

- X *is a homogeneous CTMC (HCTMC) if and only if its infinitesimal generator is independent of the time.*

- *The state probability distribution function of* X *at time* $t \in \mathbb{R}_{[0,\infty[}$ *is vector* $\boldsymbol{\pi}_X(t)$ *defined by*

$$\boldsymbol{\pi}_X(t) = [\Pr\{X(t) = i\}]_{i \in S_X}$$

- *The stationary probability distribution function of* X *is vector* $\boldsymbol{\pi}_X$ *defined by*

$$\boldsymbol{\pi}_X = \lim_{t \to \infty} \boldsymbol{\pi}_X(t) \qquad \blacksquare$$

In the continuous time case, the exponential distribution ($F(t) = 1 - e^{-\lambda \cdot t}$ for all $t \in \mathbb{R}_+$, with $\lambda \in \mathbb{R}_+$ called the rate) is the only distribution enjoying the memoryless property. Therefore, given a HCTMC X, the rates of its transitions can be interpreted as the rates of the exponential distributions describing the durations of the transitions themselves, and it can be proved that the sojourn time of state i is exponentially distributed with rate $\sum_{j \in S_X - \{i\}} q_{i,j}$ and the execution probability of the transition from state i to state j (given that the current state is i) is $q_{i,j} / \sum_{k \in S_X - \{i\}} q_{i,k}$. Moreover, if the state space is finite, it can be shown that the state probability distribution function of X at time $t \in \mathbb{R}_{[0,\infty[}$ is given by

$$\boldsymbol{\pi}_X(t) = \boldsymbol{\pi}_X(0) \cdot e^{\mathbf{Q}_X \cdot t}$$

while the stationary probability distribution function of X can be determined (if the transitions of X result in a strongly connected graph) by solving the global balance equation

$$\begin{aligned} \boldsymbol{\pi}_X \cdot \mathbf{Q}_X &= \mathbf{0} \\ \sum_{i \in S_X} \pi_i &= 1 \end{aligned}$$

Suitable numerical techniques are available for both equation systems [147].

The HCTMC X can equivalently be represented by means of the p-LTS

$$(S_X, \mathbb{R}_+, \{(i, q_{i,j}, j) \in S_X \times \mathbb{R}_+ \times S_X\}, \boldsymbol{\pi}_X(0), H_X)$$

where

$$H_X(i) = \begin{cases} t & \text{if } \exists j \in S_X \,.\, q_{i,j} > 0 \\ a & \text{otherwise} \end{cases}$$

2.2.3 Discrete Time Markov Chains

In the discrete time case, passing from one state to another is taken to be one time step.

Definition 2.14 *A discrete time Markov chain (DTMC) [115] is a discrete time stochastic process* $X = \{X_n \mid n \in \mathbb{N}\}$ *with discrete state space* S_X *such that, for each* $n \in \mathbb{N}_+$, $i_0, \ldots, i_{n-1}, i_n \in S_X$, *it turns out*

$$\Pr\{X_n = i_n \mid X_{n-1} = i_{n-1} \wedge \ldots \wedge X_0 = i_0\} = \Pr\{X_n = i_n \mid X_{n-1} = i_{n-1}\} \qquad \blacksquare$$

Definition 2.15 *Let* X *be a DTMC.*

- *The transition matrix of* X *at step* $n \in \mathbb{N}$ *is matrix* $\mathbf{P}_X(n)$ *defined by*

$$\mathbf{P}_X(n) = [\Pr\{X_{n+1} = j \mid X_n = i\}]_{i,j \in S_X}$$

- X *is a homogeneous DTMC (HDTMC) if and only if its transition matrix is independent of the time.*

- X *is an aperiodic DTMC if and only if every state of the DTMC is not periodic. A state of a DTMC is periodic if there exists* $\gamma > 1$ *(called the period of the state) such that the number of transitions traversed when going from* i *to itself is of type* γ, 2γ,3γ,...

- *The state probability distribution function of* X *at step* $n \in \mathbb{N}$ *is vector* $\boldsymbol{\pi}_X(n)$ *defined by*

$$\boldsymbol{\pi}_X(n) = [\Pr\{X_n = i\}]_{i \in S_X}$$

- *The stationary probability distribution function of X is vector $\boldsymbol{\pi}_X$ defined by*

$$\boldsymbol{\pi}_X = \lim_{n \to \infty} \boldsymbol{\pi}_X(n)$$

- *The time-averaged stationary probability distribution function of X is vector $\boldsymbol{\pi}_X$ defined by*

$$\boldsymbol{\pi}_X = \lim_{n \to \infty} \frac{1}{(n+1)} \cdot \sum_{m=0}^{n} \boldsymbol{\pi}_X(m)$$

■

In the discrete time case, the geometric distribution ($P(n) = p^n \cdot (1-p)$ for all $n \in \mathbb{N}$, with $p \in \mathbb{R}_{]0,1[}$ called the parameter) is the only distribution enjoying the memoryless property. Therefore, given a HDTMC X, it can be proved that the sojourn time of state i is geometrically distributed with parameter $p_{i,i}$ and the execution probability of the transition from state i to state j (given that the current state is i) is $p_{i,j}$ itself. Moreover, if the state space is finite, it can be shown that the state probability distribution function of X at step $n \in \mathbb{N}_+$ is given by

$$\boldsymbol{\pi}_X(n) = \boldsymbol{\pi}_X(0) \cdot \mathbf{P}_X^n$$

while the stationary probability distribution function of X can be determined (if the transitions of X result in a strongly connected graph and X is aperiodic) by solving the global balance equation

$$\begin{aligned} \boldsymbol{\pi}_X \cdot \mathbf{P}_X &= \boldsymbol{\pi}_X \\ \sum_{i \in S_X} \pi_i &= 1 \end{aligned}$$

Suitable numerical techniques are available for both equation systems [147]. If X is periodic instead of aperiodic, the formula above computes the time-averaged stationary probability distribution function of X.

The HDTMC X can equivalently be represented by means of the p-LTS

$$(S_X, \mathbb{R}_{]0,1]}, \{(i, p_{i,j}, j) \in S_X \times \mathbb{R}_{]0,1]} \times S_X\}, \boldsymbol{\pi}_X(0), H_X)$$

where

$$H_X(i) = \begin{cases} v & \text{if } \exists j \in S_X . p_{i,j} > 0 \\ a & \text{otherwise} \end{cases}$$

2.2.4 Ordinary Lumping

The ordinary lumping [144] is an aggregation method that allows an exact analysis of a MC to be carried out on a smaller stochastic process which still is a MC. Exact analysis refers to the fact that, whenever the stationary probability distribution function of the original MC exists, the stationary probability of each macrostate of the lumped MC is the sum of the stationary probabilities of the original states it contains. Though quite helpful, this aggregation should be avoided when it may cause information loss, e.g. as a consequence of merging together states having different meanings w.r.t. a given performance measure.

Definition 2.16 *Let* $(S_k, \mathbb{R}_+, \longrightarrow_k, P_k, H_k)$ *be the p-LTS representing MC* X_k, $k \in \{1,2\}$. X_2 *is an ordinary lumping of* X_1 *if and only if* S_2 *is a partition of* S_1 *such that:*

- *for all* $C \in S_2$
$$\sum_{s \in C} P_1(s) = P_2(C)$$
- *for all* $C \in S_2$ *and* $s \in C$
$$H_1(s) = H_2(C)$$
- *for all* $C, C' \in S_2$ *and* $s, s' \in C$
$$\sum \{\!| \lambda \in \mathbb{R}_+ \mid \exists s'' \in C'.\, s \xrightarrow{\lambda}_1 s'' |\!\} = \sum \{\!| \lambda \in \mathbb{R}_+ \mid \exists s'' \in C'.\, s' \xrightarrow{\lambda}_1 s'' |\!\}$$
- *for all* $C, C' \in S_2$
$$C \xrightarrow{\lambda}_2 C' \iff \lambda = \sum \{\!| \mu \in \mathbb{R}_+ \mid \exists s \in C.\, \exists s' \in C'.\, s \xrightarrow{\mu}_1 s' |\!\} \qquad \blacksquare$$

It is easily seen that, if a MC X' is an ordinary lumping of a MC X, then the p-LTSs underlying X and X' are p-bisimilar via the relation that associates each state of X with the state of X' that contains it.

2.2.5 Semi-Markov Processes

A semi-Markov process [106] is a generalization of a DTMC where the times between transitions are allowed to be random variables which depend on the current, and possibly the next, state. A CTMC can be regarded as a special case of such a stochastic process.

Definition 2.17 *A semi-Markov process (SMP) is a stochastic process* $Z = \{Z(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *with discrete state space* S_Z *defined by*
$$Z(t) = Y_{N(t)}$$
where, given a matrix of transition distributions $\mathbf{Q}_Z(t) = [q_{i,j}(t)]_{i,j \in S_Z}$ *defined over* $\mathbb{R}_{[0,\infty[}$ *and a vector of initial state probabilities* $\mathbf{a}_Z = [a_k]_{k \in S_Z}$, *we have that:*

- $(X, Y) = \{(X_n, Y_n) \mid n \in \mathbb{N}\}$ *is a two dimensional stochastic process such that:*
  - $\Pr\{Y_0 = k\} = a_k$ *for all* $k \in S_Z$*;*
  - $\Pr\{Y_{n+1} = j \wedge X_n \leq t \mid Y_n = i \wedge X_{n-1} = t_{n-1} \wedge \ldots \wedge X_0 = t_0 \wedge Y_0 = i_0\} = \Pr\{Y_{n+1} = j \wedge X_n \leq t \mid Y_n = i\} = q_{i,j}(t)$ *for all* $i, j \in S_Z$ *and* $t \in \mathbb{R}_{[0,\infty[}$*;*

here Y_n is the state of Z at the epoch of its n-th transition while X_n is the elapsed time between the n-th and $(n+1)$-th transitions;

- $N = \{N(t) \mid t \in \mathbb{R}_{[0,\infty[}\}$ *is a stochastic process defined by*

$$N(t) = \sup\{n \in \mathbb{N} \mid S_n \leq t\}$$

  where $S = \{S_n \mid n \in \mathbb{N}\}$ is a stochastic process such that:

  - $S_0 = 0$;
  - $S_n = \sum_{k=0}^{n-1} X_k$ *for all* $n \in \mathbb{N}_+$;

  here S_n is the epoch of the n-th transition while $N(t)$ is the number of transitions that occur by time t.

Y is called the embedded MC of Z and its transition matrix is given by $\mathbf{P}_Y = [q_{i,j}(\infty)]_{i,j \in S_Z}$. ■

A SMP thus evolves as follows. Transitions occur from state to state according to a DTMC. When the process is in state i, the next state visited is j with probability $p_{i,j}$. When the successive states are i and j, the distribution function of the time to make the transition is $F_{i,j}(t) = q_{i,j}(t)/p_{i,j} = \Pr\{X_n \leq t \mid Y_n = i \wedge Y_{n+1} = j\}$. We denote by $\mathbf{h}(t) = [h_i(t)]_{i \in S_Z}$ the vector of the holding time distribution functions, where $h_i(t) = \sum_{j \in S_Z} q_{i,j}(t) = \Pr\{X_n \leq t \mid Y_n = i\}$.

Definition 2.18 *Let Z be a SMP.*

- *The transition matrix of Z from time $t \in \mathbb{R}_{[0,\infty[}$ to time $t' > t$ is matrix $\mathbf{P}_Z(t,t')$ defined by*

$$\mathbf{P}_Z(t,t') = [\Pr\{Z(t') = j \mid Z(t) = i\}]_{i,j \in S_Z}$$

- *Z is a homogeneous SMP (HSMP) if and only if its transition matrix is independent of the time.*

- *The state probability distribution function of Z at time $t \in \mathbb{R}_{[0,\infty[}$ is vector $\boldsymbol{\pi}_Z(t)$ defined by*

$$\boldsymbol{\pi}_Z(t) = [\Pr\{Z(t) = i\}]_{i \in S_Z}$$

- *The stationary probability distribution function of Z is vector $\boldsymbol{\pi}_Z$ defined by*

$$\boldsymbol{\pi}_Z = \lim_{t \to \infty} \boldsymbol{\pi}_Z(t)$$ ■

Whenever the stationary probability distribution function of a HSMP Z exists, it can be determined by first solving

$$\begin{array}{rcl} \boldsymbol{\pi}_Y \cdot \mathbf{P}_Y & = & \boldsymbol{\pi}_Y \\ \sum\limits_{i \in S_Z} \pi_{Y,i} & = & 1 \end{array}$$

and then letting

$$\pi_{Z,i} = \frac{\pi_{Y,i} \cdot \bar{h}_i}{\sum_{k \in S_Z} \pi_{Y,k} \cdot \bar{h}_k} \tag{2.1}$$

where $\bar{h}_i$ is the average sojourn time in state i.

In this thesis we consider HSMPs where the times between transitions are allowed to be only exponentially distributed or zero [2] and depend only on the current state. Moreover, all the outgoing transitions of a given state must be of the same type. A HSMP Z of this kind can be represented by means of the p-LTS

$$(S_Z, \mathbb{R}_+, \{(i, r_{i,j}, j) \in S_Z \times \mathbb{R}_+ \times S_Z\}, \boldsymbol{\pi}_Z(0), H_Z)$$

where $r_{i,j}$ is the rate or the probability of the transition from state i to state j depending on whether it is exponentially timed or immediate, while

$$H_Z(i) = \begin{cases} v & \text{if } \bar{h}_i = 0 \\ t & \text{if } \bar{h}_i \in \mathbb{R}_+ \\ a & \text{if } \bar{h}_i = \infty \end{cases}$$

2.2.6 Generalized Semi-Markov Processes

The class of Generalized Semi-Markov Processes (GSMPs), introduced by Matthes (1962) in [121], extends the expressiveness of the class of semi-Markov process by introducing the possibility to have multiple *active elements* in a state, each with a different duration distribution of its life. When a active element terminates (*it dies*) a state change occurs and the other elements continue their life in the following state, thus carrying out their residual duration.

Definition 2.19 A *generalized semi-Markov process (GSMP)* is a stochastic process defined on a set of states $\{s \mid s \in \mathcal{S}\}$ as follows. In each state s there is a set of active elements $ElSt(s)$ taken from a set El, that decay at the rate $Dec(e, s)$, $e \in El$. The set El is partitioned into two sets El' ed El^* with $El = El' \cup El^*$. If $e \in El'$ the element e has an exponentially distributed lifetime, if instead $e \in El^*$ it has an arbitrarily distributed lifetime. Whenever in a state s an advancing element e dies, the process moves to the state $s' \in \mathcal{S}$ with probability $P(s, e, s')$.
The following constraints must be satisfied.

- When the process moves from a state to another, no more than one element of El^* can be born or die contemporaneously.
- The active elements of El^* that do not die in a transition keep their residual duration. ■

[2] If we are interested in the stationary distribution of HSMPs only, this is not a limiting condition, since every HSMP can be represented in this way by preserving average sojourn times in states (see formula 2.1 and the notion of insensitivity for semi-Markov processes in Sect. 2.2.6).

The described process is not Markovian since the lifetimes of elements are arbitrarily distributed. It is not Markovian even in the instants of state change in that an element can continue its life from previous states and its residual duration depends on the lifetime it already spent in the past. The standard technique of including supplementary variables in the representation of a process state [65] allows us to retrieve the Markov property. The state space is extended in the following way: a state of the process is described by a pair $(s, \mathbf{y})$, where $s \in \mathcal{S}$ and $\mathbf{y}$ is a vector that includes, for each element of $ElSt(s)$ belonging to El^*, either its residual lifetime or the lifetime it already spent in the past. Both methods achieve the desired goal. Therefore the process assumes partially discrete (states of $\mathcal{S}$) and partially continuous (domain of $\mathbf{y}$) state space.

Because of their complexity, GSMPs are mainly employed in the framework of performance modeling with the purpose to apply simulation techniques. Nevertheless some previous work (see [121, 93, 92, 94]) has been done on the investigation of exact analysis techniques for the evaluation of the steady state probability of GSMPs. For this purpose the concept of *insensitivity* has been introduced.

Definition 2.20 A GSMP is said to be *insensitive* with respect to a given set of elements if its steady state probability depends only on the mean of the lifetime distributions of such elements and not on the "shape" of the distributions. ■

As a consequence the equilibrium behavior of an insensitive GSMP can be studied by replacing general distributions with any other distributions with the same mean. Exponential distributions are normally used for this purpose in order to reduce the GSMP to a CTMC that can be easily analyzed. It is worth noting that, by observing the formula for the calculation of the steady state probability of a SMP, it easily derives that such class of processes is insensitive to the distribution of the sojourn time in a state.

Conditions for the insensitivity of a GSMP have been studied by Matthes in [121].

Theorem 2.1 The following statements are equivalent for GSMPs:

- The process is insensitive with respect to the active elements of El^*. That is, the general distributions of the lifetimes of the elements of El^* can be replaced by any other distributions with the same mean, and yet the process still retains the same equilibrium distribution.
- When all active elements of El^* are assumed to be negative exponentially distributed, the flux out of each state due to the death of an element of El^* is equivalent to the flux into that state due to the birth of that element. ■

The second statement describes the *insensitivity balance equations* for each element. In SMPs the insensitivity balance equations are identical to the global balance equations (the equations used to determine the steady state distribution of the SMP) because in each state there is a single element which is born when the state is entered and which dies when the state is left. Therefore it follows immediately that any SMP is insensitive.

In the case of insensitivity we have the following theorems.

Theorem 2.2 Consider an insensitive GSMP whose state space has been extended with supplementary variables. The *equilibrium density* [3] is the same no matter if the state space is extended with the residual lifetimes or with spent lifetimes. ■

Theorem 2.3 Consider an insensitive GSMP whose state space has been extended with supplementary variables. The *equilibrium density* of being in state $(s, \mathbf{y})$ is:

$$\pi(s, \mathbf{y}) = \pi(s) \prod_{e \in ElSt(s) \cap El^*} \mu_e \cdot (1 - PDF_e(y_e))$$

where $\pi(s)$ is the equilibrium distribution of the purely Markovian form of the GSMP (where each distribution is replaced by the equivalent exponential one), $PDF_e(t)$ is the distribution of the lifetime of element e, y_e is the variable associated with e, and $(\mu_e)^{-1}$ is the mean of the distribution of e. ■

A *GSMP* can be represented by a tuple:

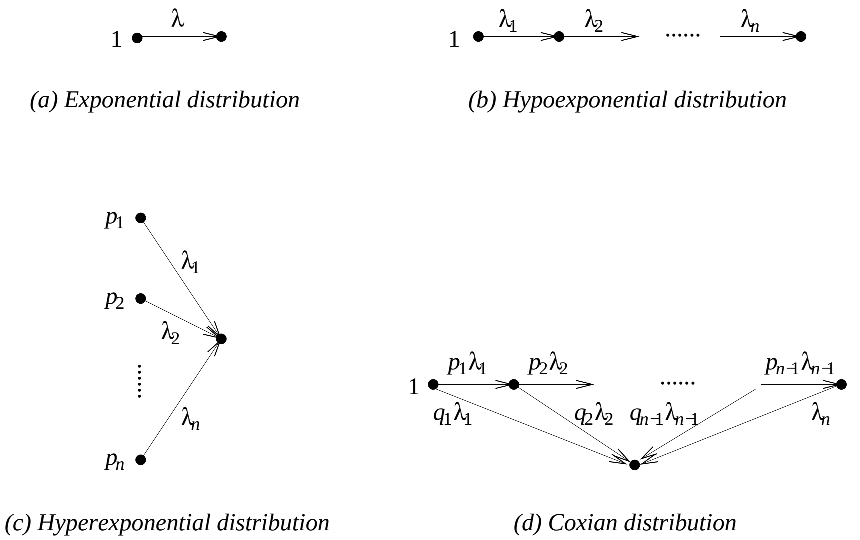
$$(\mathcal{S}, El, ElPDF, ElSt, Dec, \longrightarrow, Pr, P_{init})$$

where:

- $\mathcal{S}$ is the set of the states of the *GSMP*.
- El is the set of the elements of the *GSMP*.
- $ElPDF : El \longrightarrow PDF^+$ is a function that associates with each element the distribution of its lifetime.
- $ElSt : \mathcal{S} \longrightarrow El$ is a function that associates with each state the set of its active elements.
- $Dec : El \times \mathcal{S} \longrightarrow\!\!\!\!\!\!\!\circ\;\; \mathbb{R}^+$ is a partial function that associates a decay rate with each active element of each state. Dec is partial because for each $s \in \mathcal{S}$ it is defined only for the (e, s) such that $e \in ElSt(s)$.
- $\longrightarrow \subseteq \mathcal{S} \times El \times \mathcal{S}$ is a relation that represents the transitions between the states of the *GSMP*. They are labeled by the element $e \in El$ that terminates. We include only transitions for which $Pr(s, e, s') > 0$.
- Pr is a function that associates a (non zero) probability with each transition of the *GSMP* (relation $\longrightarrow$). The meaning of Pr is: if in s an element e terminates, with probability $Pr(s, e, s')$ the process moves into state s'. For what we said in the previous item, Pr is never zero over its domain, whilst it is considered as zero outside.
- $P_{init} : \mathcal{S} \longrightarrow \mathbb{R}_{[0,1]}$ is a function that associates with each state the probability that it is the initial state.

Note that, given a tuple defining a *GSMP*, the sets El' and El^* are derived in the following way:

[3] Since the state space is now partially continuous we speak of "density", instead of "distribution".



(a) Exponential distribution *(b) Hypoexponential distribution*

(c) Hyperexponential distribution *(d) Coxian distribution*

Figure 2.1: Phase-Type Distributions

- $El' = \{e \in El \mid \exists \lambda : ElPDF(e) = Exp(\lambda)\}$
- $El^* = \{e \in El \mid \nexists \lambda : ElPDF(e) = Exp(\lambda)\}$

where $Exp(\lambda)$ is the exponential distribution with rate λ.

2.2.7 Phase-Type Distributions

A *phase-type distribution* [129] is a continuous distribution function describing the time to absorption in a finite state HCTMC having exactly one absorbing state. Well known examples of phase type distributions are the exponential distribution, the hypoexponential distribution, the hyperexponential distribution, and the Coxian distribution, which are characterized in terms of time to absorption in a finite state HCTMC with an absorbing state as depicted in Fig. 2.1. In Fig. 2.1(a) we show the HCTMC corresponding to an exponential distribution with rate $\lambda \in \mathbb{R}_+$; in Fig. 2.1(b) the HCTMC corresponding to an n-stage hypoexponential distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$; in Fig. 2.1(c) the HCTMC corresponding to an n-stage hyperexponential distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$ and branching probabilities $p_i \in \mathbb{R}_{]0,1]}$, $1 \leq i \leq n$, where $\sum_{i=1}^{n} p_i = 1$; finally in Fig. 2.1(d) the HCTMC corresponding to an n-stage Coxian distribution with rates $\lambda_i \in \mathbb{R}_+$, $1 \leq i \leq n$, and branching probabilities $p_i, q_i \in \mathbb{R}_{]0,1]}$ where $p_i + q_i = 1$, $1 \leq i \leq n-1$.

As we will see more extensively in the second part of the thesis, phase-type distributions make the limitation to exponential distributions inherent of HTCMCs less restrictive from a modeling viewpoint, as they can be used to describe or approximate many probability distributions frequently occurring in practice.

2.2.8 Reward Structures

Reward structures permit us to specify and derive measures for system models represented by SMPs.

Definition 2.21 *A reward structure [110] for a SMP is composed of:*

- *A yield function $y_{i,j}(t)$ expressing the rate at which reward is accumulated at state i t time units after i was entered when the successor state is j.*
- *A bonus function $b_{i,j}(t)$ expressing the reward awarded upon exit from state i and subsequent entry into state j given that the holding time in state i was t time units.* ■

Since the generality of this structure is difficult to fully exploit due to the complexity of the resulting solution, the analysis is usually simplified by considering yield functions that do not depend on the time nor the successor state, as well as bonus functions that do not depend on the holding time of the previously occupied state: $y_{i,j}(t) = y_i$ and $b_{i,j}(t) = b_{i,j}$.

Several performance measures can be calculated by exploiting rewards. According to the classifications proposed in [143, 91], we have *instant-of-time measures*, expressing the gain received at a particular time instant, and *interval-of-time (or cumulative) measures*, expressing the overall gain received over some time interval. Both kinds of measures can refer to stationary or transient state. In the following, we shall concentrate on instant-of-time performance measures.

In the stationary case, instant-of-time performance measures quantify the long run gain received per unit of time. Given yield rewards y_i and bonus rewards $b_{i,j}$ for a certain SMP, the corresponding stationary performance measure is computed as:

$$\sum_i y_i \cdot \pi_i + \sum_i \sum_j b_{i,j} \cdot \phi_{i,j} \tag{2.2}$$

where π_i is the stationary probability of state i and $\phi_{i,j}$ is the stationary frequency with which the transition from state i to state j is traversed.

- In the case of a homogeneous semi-Markov reward process (HSMRP) with transition matrix of the embedded DTMC $\mathbf{P}$ (whose stationary probabilities are denoted π') and with average holding times $\mathbf{h}$ restricted to be either exponentially distributed or constant zero, the stationary frequency $\phi_{i,j}$ is given by the stationary frequency with which state i is entered (i.e. the ratio of its stationary probability to its average holding time) multiplied by the probability with which the transition from state i to state j is traversed given that the current state is i:

$$\phi_{i,j} = \frac{\pi'_i}{\sum_k \pi'_k \cdot h_k} \cdot p_{i,j}$$

- In the case of a homogeneous continuous time Markov reward chain (HCTMRC) with infinitesimal generator $\mathbf{Q}$, the stationary frequency $\phi_{i,j}$ is given by the stationary probability of state i multiplied by the rate of the transition from state i to state j (as the probability of traversing such a transition given that the current state is i is the ratio of its rate to the reciprocal of the average holding time of i):

$$\phi_{i,j} = \pi_i \cdot q_{i,j}$$

- In the case of a homogeneous discrete time Markov reward chain (HDTMRC) with transition matrix $\mathbf{P}$, the stationary frequency $\phi_{i,j}$ is given by the stationary probability of state i (as a transition from state i is executed one time unit after entering i) multiplied by the probability with which the transition from state i to state j is traversed given that the current state is i:

$$\phi_{i,j} = \pi_i \cdot p_{i,j}$$

In the transient state case, instant-of-time performance measures quantify the gain received at a given time instant. Given yield rewards y_i and bonus rewards $b_{i,j}$ for a certain SMP, the corresponding transient state performance measure is computed as:

$$\sum_i y_i \cdot \pi_i(t) + \sum_i \sum_j b_{i,j} \cdot \phi_{i,j}(t) \tag{2.3}$$

where $\pi_i(t)$ is the probability of being in state i at time t and $\phi_{i,j}(t)$ is the the transient frequency with which the transition from state i to state j is traversed at time t, which is computed in the same way as $\phi_{i,j}$ with $\pi_i(t)$ in place of π_i.

As a graph theoretic representation of HSMRPs, HCTMRCs, and HDTMRCs, we propose an extension of p-LTSs where bonus rewards are added to transition labels and a yield reward function over states is introduced.

Definition 2.22 *A probabilistically rooted labeled transition system with rewards (pr-LTS) is a tuple*

$$(S, \mathbb{R}_+ \times \mathbb{R}, \longrightarrow, P, H, Y)$$

where:

- *S, $\longrightarrow$, P, and H are defined as for a p-LTS.*

- *$Y : S \longrightarrow \mathbb{R}$ is called the yield reward function.* ■

Definition 2.23 *Let $Z_k = (S_k, \mathbb{R}_+ \times \mathbb{R}, \longrightarrow_k, P_k, H_k, Y_k)$, $k \in \{1, 2\}$, be two pr-LTSs.*

- *Z_1 is pr-isomorphic to Z_2 if and only if there exists a bijection $\beta : S_1 \longrightarrow S_2$ such that:*

– *for all* $s \in S_1$

$$\begin{aligned} P_1(s) &= P_2(\beta(s)) \\ H_1(s) &= H_2(\beta(s)) \\ Y_1(s) &= Y_2(\beta(s)) \end{aligned}$$

– *for all* $s, s' \in S_1$, $\lambda \in \mathbb{R}_+$, *and* $r \in \mathbb{R}$

$$s \xrightarrow{\lambda,r}_1 s' \iff \beta(s) \xrightarrow{\lambda,r}_2 \beta(s')$$

- *Z_1 is pr-bisimilar to Z_2 if and only if there exists a relation $\mathcal{B} \subseteq S_1 \times S_2$ with reflexive, symmetric and transitive closure $\mathcal{B}' \subseteq (S_1 \cup S_2) \times (S_1 \cup S_2)$ such that:*

  – *for all* $C \in (S_1 \cup S_2)/\mathcal{B}'$

$$\sum_{s \in C \cap S_1} P_1(s) = \sum_{s \in C \cap S_2} P_2(s)$$

  – *whenever* $(s_1, s_2) \in \mathcal{B}$, *then*

$$\begin{aligned} H_1(s_1) &= H_2(s_2) \\ Y_1(s_1) &= Y_2(s_2) \end{aligned}$$

  – *whenever* $(s_1, s_2) \in \mathcal{B}$, *then for all* $r \in \mathbb{R}$ *and* $C \in (S_1 \cup S_2)/\mathcal{B}'$

$$\sum\{\!| \lambda \in \mathbb{R}_+ \mid \exists s_1' \in C \cap S_1 .\, s_1 \xrightarrow{\lambda,r}_1 s_1' |\!\} = \sum\{\!| \lambda \in \mathbb{R}_+ \mid \exists s_2' \in C \cap S_2 .\, s_2 \xrightarrow{\lambda,r}_2 s_2' |\!\} \quad \blacksquare$$

The relationship between ordinary lumping with rewards [130] and pr-bisimilarity is the same as the relationship between ordinary lumping and p-bisimilarity reported at the end of Sect. 2.2.4.

2.2.9 Queueing Systems

A *queueing system (QS)* [115] is an abstract model largely used for evaluating the performance of computer and communication systems through the computation of measures such as system throughput, resource utilization, and user response time. A QS is a service center, composed of a waiting queue and a given number of servers, which provides a certain service to a population of customers according to a given discipline. As an example, we can think of a computer system as a QS, where the central unit can be viewed as the server and the various devices can be viewed as the customers.

In this thesis, we shall be mainly concerned with QSs $M/M/n/q/m$ with arrival rate λ and service rate μ, which are defined as follows (see Fig. 2.2):

1. The customer arrival process is Markovian with rate λ.

2. The customer service process is Markovian with rate μ.
3. There are n independent servers.
4. There is a FIFO queue with $q - n$ seats. When missing, parameter q denotes an unbounded queue.
5. There are m independent customers. When missing, parameter m denotes an unbounded population of customers.

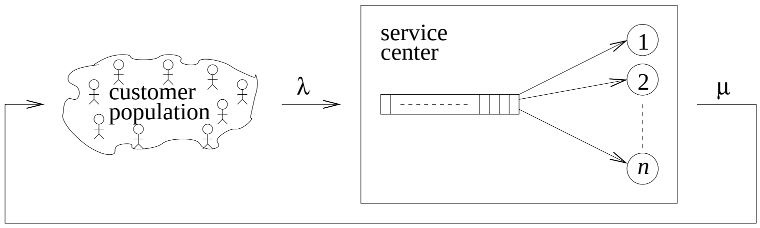


Figure 2.2: Structure of a QS

As an example, consider a QS $M/M/1$ with arrival rate λ and service rate μ. The stochastic process underlying this QS is a HCTMC. If state s_i represents the situation in which i customers are in the system, then state s_0 has a transition toward s_1 labeled with λ while state s_i, $i \in \mathbb{N}_+$, has a transition toward s_{i+1} labeled with λ and a transition toward s_{i-1} labeled with μ. It can be shown that this HCTMC admits stationary probability distribution function if and only if $\lambda < \mu$, in which case for all $i \in \mathbb{N}$

$$\pi_i = (1 - \lambda/\mu)(\lambda/\mu)^i$$

From this distribution we can compute the mean number of customers in the system as

$$\sum_{i=0}^{\infty} i \cdot \pi_i = \frac{(\lambda/\mu)}{(1 - \lambda/\mu)}$$

while the average waiting time in the system is derived by Little's law as the value above divided by λ

$$\frac{(1/\mu)}{(1 - \lambda/\mu)}$$

Moreover, the utilization of the server is given by

$$\sum_{i=1}^{\infty} \pi_i = 1 - \pi_0 = \lambda/\mu$$

The presence of this background section is justified by the fact that QSs will be often used as examples throughout the thesis.

Part I

ST Semantics

Chapter 3

Deciding and Axiomatizing Weak ST Bisimulation

In this chapter we address the problem of deciding and axiomatizing weak ST bisimulation for process algebras with recursion and action refinement. ST bisimulation is a bisimulation based equivalence which expresses the execution of an action as the combination of the two interdependent events of action start and action termination and which supports the operation of action refinement. Since in the rest of the thesis we will use ST semantics for expressing the behavior of timed formalisms with durational activities, this chapter introduces basic results that will be used several times throughout the thesis.

Due to the complex nature of bisimulation equivalences which extend the ordinary notion of bisimulation by expressing some form of history dependence (like ST bisimulation), it turned out to be problematic to decide them over non trivial classes of recursive systems. Moreover, to the best of our knowledge, the problem of axiomatizing them over such classes of systems has never been previously solved.

In this chapter we solve this problem in the case of weak ST bisimulation as follows. We first consider the basic process algebra with CSP multiway synchronization and recursion introduced in Chapter 2 and we show that a simple technique based on static names is sufficient to decide weak ST bisimulation over processes which are finite state according to the standard interleaving semantics. Then we introduce a different technique based on dynamic names and on the new idea of compositional level-wise renaming of actions (which produces semantic models via Structural Operational Semantics such that weak ST bisimulation can be established through standard weak bisimulation) and we show that it can be applied to decide and axiomatize weak ST bisimulation over the same class of processes. Finally we introduce a third technique based on pointers, updated according to a pseudo stack discipline, which preserves the possibility of deciding and axiomatizing weak ST bisimulation also when an action refinement operator is considered.

3.1 Introduction

ST semantics, originally defined in [81] over Petri Nets, is one of the most studied non-interleaving semantics. The main reason is that ST semantics is the least informative semantics that is a congruence for action refinement [78, 4, 152, 83, 85, 52], thus allowing for the top-down design of concurrent systems. This semantics describes the behavior of a concurrent system in terms of activities which exhibit a duration. In particular, actions are not atomic as in standard interleaving semantics, but the execution of an action is split into the two distinguished events of action start and action termination. Between such events, other system activities may evolve. Moreover, enough information is included in semantic models so that the event of an action termination uniquely determines to which event of action start it refers to, even in the situation of auto-concurrency (i.e., multiple actions of the same type being in execution at the same time). Different implementations of the ST semantics are reported in [78, 52] and, under other names, also in [152, 4, 75].

The ST semantics has found application in several different contexts. For example the ST idea of representing durational actions has turned out to be useful to express the semantics of languages with time, e.g., [153] argues that ST bisimulation is the correct equivalence for comparing the time behavior of concurrent systems in an asynchronous setting, while in this thesis we will show that, when the duration of the activities of a concurrent system is specified by a general probability distribution, the use of ST semantics produces suitable semantic models that can be analyzed to obtain performance measures.

Besides ST bisimulation, other bisimulation equivalences expressing some form of history dependence (the fact that in ST semantics action terminations are uniquely related to action starts is a form of history dependence) have been defined in the literature: e.g. history preserving bisimulation [142, 71, 79, 72] and location bisimulation [31, 2, 127, 55]. Due to the complex nature of this kind of equivalences it turned out to be problematic to decide them in the presence of recursion (see for instance [114, 77, 76] for positive results and [111] for a negative result). Moreover to the best of our knowledge, the problem of axiomatizing them over non trivial classes of systems (e.g. classes which include most recursive systems) has never been solved. For instance, the axiom systems for ST bisimulation presented in [96, 3] only deal with nonrecursive systems.

Two main approaches have been developed in the literature for expressing history dependent bisimulations.

The first approach is based on *static names* [79, 78, 4, 2, 55]. According to this approach a unique name is statically assigned (i.e. on the basis of the syntactical structure of the process) to each different "historical element" (e.g. a started action in ST semantics) which must be referred to in labels of process computations. Such references express the history dependence. Then history dependent bisimulation is defined by making associations among static names of historical elements of different processes that are put in correspondence by the equivalence. An advantage of this approach is that it gives the opportunity to produce (as done for location semantics in [2, 55]) finite semantic models also for a wide class of recursive systems. The main drawback of this approach is that each kind of history dependent bisimulation has a different definition which deviates from standard bisimulation. As a consequence, there is no simple way to derive an axiomatization for such history dependent bisimulations by using an approach similar to that employed for standard bisimulation in process algebras with an interleaving semantics (equality of two terms would depend on the associations already established among their historical elements), and the results previously developed for standard bisimulation theory (e.g. tools for verification) cannot be directly exploited.

The second approach is based on *pointers* [71, 83, 85, 52]. According to this approach, a historical dependence is expressed in the label of a computation by a *pointer* that determines the position in the semantic model of the transition that "activated" such "historical element" (i.e. the transition representing an action start in ST semantics). This approach has the advantage that the equivalence of terms, according to a history dependent bisimulation, can be established by simply applying the standard notion of bisimulation on a particular semantic model especially constructed for that kind of history dependent bisimulation. The drawback of the techniques used in [71, 83, 85, 52] is that for most recursive systems (e.g. for $a \,\|\, recX.b.X$ in [71, 83, 52]) a semantic model with an infinite number of states is produced. As a consequence there is no easy way to decide or axiomatize history dependent equivalences between process terms which include recursion.

A further step is made in [126]. Here a technique for expressing history preserving semantics is developed, which is based on *dynamic names*. This technique combines the advantages of the first and second approach: finite semantic models are produced for a wide class of recursive systems and history preserving bisimulation is decided simply by applying the standard definition of bisimulation. The idea of the approach of [126] is to *dynamically assign* - with a *fixed rule* - a different name to each new "historical element" that becomes active, on the basis of the names of the historical elements already active; in particular, names of obsolete "historical elements" (which are no longer active) are *reused*. Note that this technique is based on names as the approach of [79, 78, 4, 2, 55], but here names are not assigned statically (i.e. at compile-time) according to their syntactical position in the initial process, they are instead computed dynamically while the system evolves (i.e. at run-time). [1] Since the method to compute new names is fixed, processes that perform equivalent computations produce the same names for "historical elements". As a consequence the history dependent bisimulation can be decided by applying standard bisimulation. Unfortunately the technique developed in [126] is not compositional, in the sense that, in order to produce the history preserving semantic model of a term, first an intermediate semantic model of the whole term must be computed and then transformed by adjusting history information in transitions. Hence also with the technique of [126], since it is not compositional, there is no easy way to produce an axiomatization for a history dependent bisimulation.

In this chapter, we first show that, considered the process algebra introduced in Chapter 2 which extends the basic algebra of [4] (which has prefix, choice and CCS [122] parallel composition) with CSP [109] multiway synchronization and recursion, a technique based on static names can be easily employed to decide weak ST bisimulation over processes which are finite state according to the standard interleaving semantics. Then we show that an axiomatization of weak ST bisimulation which is complete over such class of processes can

[1] The idea of dynamic names already appeared in [31], but in that chapter no special technique (like reuse of names) is employed in order to obtain finite models.

be naturally obtained by combining the idea of dynamic names (which allow us to produce finite semantic models for a wide class of recursive systems and to establish ST bisimulation through standard bisimulation) with the new idea, we introduce here, of *levelwise renaming* of actions which provide us with compositionality. Finally we consider a more complex process algebra which includes in addition an action refinement operator and we face the problem of deciding and axiomatizing weak ST bisimulation for such a process algebra. We solve this problem by considering a third, completely new, idea for referring to historical elements (action starts) which, with respect to the dynamic name technique, maximizes the reuse of names. The resulting technique turns out to be no longer based on (static or dynamic) names, but on a dynamic re-denomination of actions which makes the technique more similar to an approach based on pointers. The main difference with the previous techniques based on pointers is that our new technique, which is called *stack technique*, is not affected by the problem of producing infinite semantic models in the presence of recursion for a wide class of processes. We show that by combining the stack technique with the idea of levelwise renaming it is indeed possible to axiomatize weak ST bisimulation over a wide class of processes of the language with action refinement.

In the following we will explain in more details the results that we present in this chapter.

3.1.1 ST Semantics via Static Names

We start by considering a basic process algebra like that introduced in Chapter 2, called *Basic Language* (*BL*), equipped with the CCS prefix and choice operators, the CSP parallel and hiding operators and a recursion operator. [2] We show how to define ST semantics in SOS style [137] for *BL* by using static names so that finite state semantic models are obtained for processes which are finite state according to the standard interleaving semantics. This is obtained thanks to the observation that, for the purpose of defining an ST semantics, it is not necessary to assign a different static name to *every* action of a process (as done e.g. in [4]), but it is just sufficient to consider action *locations* (syntactical positions with respect to parallel operators) as their static names. Moreover with respect to [4] the definition of operational ST semantics is more complex because of multiway CSP synchronization. As we will see, the fact that visible actions can be obtained by synchronization leads to the necessity of enriching the representation of system states so that they include explicitly the information about the locations of the actions in execution (i.e. started but not terminated yet). The definition of weak ST bisimulation for static names that we employ is just that defined in [4] under the name of "weak refine bisimulation", where equivalence is established by making associations among static names of actions that are put in correspondence by the bisimulation. Note that it is correct to consider the equivalence of [4] as a definition for ST bisimulation since it has been proven in [84] to correspond to the original definition of ST equivalence [78]. We show that through our technique we can decide weak ST bisimulation over *BL* processes with a finite state interleaving semantics.

[2] To be precise, with respect to the algebra considered in Chapter 2, in *BL* we use a recursion operator "$recX.\_$", instead of process constants, to denote recursion and we do not consider a relabeling operator.

3.1.2 ST Semantics via Dynamic Names

Then we address the problem of axiomatizing weak ST bisimulation over processes of BL with a finite state interleaving semantics. Our approach is based on the observation that this can be easily done provided that we have a compositional approach for deriving semantic models which express ST semantics via dynamic names (a different name is dynamically assigned to each action that starts execution through a fixed rule and names of terminated actions are reused when new actions start). This because, similarly to [126], such semantic models are finite for processes with a finite state interleaving semantics and ST bisimulation of two processes can be decided simply by applying the standard definition of bisimulation. More precisely what we need is a structural operational semantics (SOS [137]) which allows us to derive, notably for the parallel composition operator "$\|$", the ST computations of $P \,\|\, Q$ (starts or terminations of actions with certain dynamic names) from the ST computations of P and Q. As long as we consider only terms with choice, prefix and termination operators, where ST computations are encoded in prefixes (such terms are just *normal forms*, i.e. representations of a semantic model), axiomatizing standard bisimulation is simply done with the standard axiom set developed by Milner [122]. In order to transform a general term in normal form it is just sufficient to have axioms that reflect the operational rules for the operator $\|$, i.e. they must derive from the computations of P and Q (prefixes representing starts or terminations of actions with certain dynamic names), the computations of $P \,\|\, Q$ (in the form of prefixes representing starts or terminations of actions with certain dynamic names).

Therefore the essence of the problem of developing the axiomatization is obtaining compositionality in the generation of semantic models. In order to do this it is necessary to associate in some way the dynamic names assigned to actions at the level of $P \,\|\, Q$, generated according to the fixed rule for creating new names, to the dynamic names assigned to the same actions inside P or Q (which in general are different) generated according to the same rule. In this way when a future reference to an action is made by a termination transition inside P or Q such a reference can then be re-mapped to the correct reference at the level of $P \,\|\, Q$.

We show that it is possible to do this by parameterizing in state terms each parallel operator with a mapping M which records such associations while new names are generated. The resulting technique is a *levelwise renaming technique* where actions are renamed at each structural level (e.g. from the level of P or Q to the level of $P \,\|_M\, Q$) according to such mappings.

Through this technique we define ST semantics in SOS style for BL processes via dynamic names and we show that: (i) the semantic models produced are finite for processes with a finite state interleaving semantics, (ii) standard weak bisimulation of the semantic models of two BL processes holds whenever they are weak ST bisimilar according to the definition with static names, and (iii) we can easily produce an axiomatization for weak ST bisimulation which is complete over finite state guarded processes.

3.1.3 ST Semantics via the Stack Technique

In the last part of the chapter we propose a better (even if less intuitive) technique for implementing ST semantics that is based on some sort of pointers instead of dynamic names, but still relies on our idea of levelwise renaming. The main difference with the previous technique is that, while with the dynamic name technique the reuse of names is "on demand", i.e. it is delayed until new names need to be activated, here the reuse of names is always done as soon as an "active" (started) action terminates, by changing the names of the other active actions. Hence here reuse is performed eagerly. A consequence is that the name assigned to an active action changes dynamically while other actions start and terminate, hence it assumes the flavor of a pointer. In a state of a semantic model the name of an active action (its pointer) is determined by the position of such action in a "stack" of the currently active actions. For this reason this technique is called stack technique. The stack technique produces a simpler representation for states and more compact semantic models with respect to the technique based on dynamic names.

Moreover we show that with the stack technique it is possible to solve some problems that arise with the dynamic name technique when we consider a *Refinement Language* RL which extends the language BL with an action refinement operator $P[a \rightsquigarrow Q]$ which performs the semantic refinement of all actions a executed by P to Q [78, 73, 86]. In particular with the dynamic name technique there is no easy way to define a semantics for $P[a \rightsquigarrow Q]$ so that if both P and Q are finite state processes then $P[a \rightsquigarrow Q]$ is finite state, hence to decide and axiomatize ST bisimulation for a wide class of processes containing the refinement operator.

Through the new approach, we define ST semantics in SOS style for RL processes via the stack technique and we show that: (i) the semantic models produced are finite for BL processes with a finite state interleaving semantics, (ii) standard weak bisimulation of the semantic models of two BL processes holds whenever they are weak ST bisimilar according to the definition with static names, (iii) if both P and Q are finite state processes then $P[a \rightsquigarrow Q]$ is finite state, and (iv) we can easily produce an axiomatization for weak ST bisimulation which is complete over finite state guarded processes of RL.

3.1.4 Outline of the Chapter

The chapter is organized as follows. In Sect. 3.2 we present the basic language BL. In Sect. 3.3 we present the technique for defining ST semantics of BL via static names and we recall the definition of weak ST bisimulation of [4]. In Sect. 3.4 we present the technique for defining ST semantics of BL via dynamic names and we present a complete axiomatization of weak ST bisimulation of finite state guarded BL processes. In Sect. 3.5 we analyze the drawbacks of the dynamic name technique and we introduce the refinement language RL. In Sect. 3.6 we present the stack technique for defining ST semantics of RL and we present a complete axiomatization of weak ST bisimulation of finite state guarded RL processes. In Sect. 3.7 we compare the size of the ST semantic models produced with our three techniques for a BL process. Finally, in Sect. 3.8 we report some concluding remarks.

3.2 The Basic Language

Let A be a countable set of observable action types; a, b, c range over A and S, L over the subsets of A. The set of all action types is denoted by $Act = A \cup \{\tau\}$, where τ is a distinguished type representing an internal computation. Let μ range over Act. Moreover let Var be a set of process variables ranged over by X, Y, Z. The BL terms are generated by the following syntax:

$$P ::= \underline{0} \mid X \mid \mu.P \mid P + P \mid P \|_S P \mid P/L \mid recX.P$$

"$\underline{0}$" denotes a process that cannot move. The operators "." and "+" are the CCS [122] prefix and choice. "$\|_S$" is the CSP [109] parallel operator, where synchronization over actions in S is required. "$/L$" is the hiding operator which turns the actions in L into τ actions. Finally "$recX$" denotes recursion in the usual way. We denote by BL_C the set of BL processes, i.e. closed terms of BL, and by BL_{CG} the set of strongly guarded processes of BL (see the definition of BL strongly guarded process in Appendix A). In the following, in order to avoid ambiguities, we assume the following operator precedence relation: hiding > prefix > recursion > parallel composition > choice.

3.3 ST Semantics via Static Names

The *static name* technique is based on the idea of statically assigning a name to each action occurring in a BL process P on the basis of its syntactical position in P. Names must be assigned to actions in such a way that we are guaranteed that any two actions of the same type a that can be executed simultaneously (there may exist a state in which they are both started but not terminated yet) always get different names. In this way, when an action a with a given name terminates, it is always clear which action a (among those in execution) is terminating and we preserve the correct relationship between action starts and action terminations, as required by ST semantics. We point out that, for the purpose of defining an ST semantics, it is not necessary to identify *every* action of a term as a different action (as done, e.g., in [4]). For example two consecutive actions a of a term executed by the same sequential process (with the form $a.a.P'$) cannot be concurrently executed, but are causally related. Thus the two actions cannot overlap during their execution and once an action has started it must terminate before the other one can start. Therefore even if the two actions are not distinguished by the identification mechanism the event of an action termination is still uniquely related to the event of its start (there is no ambiguity about which action is terminating). Also alternative actions a of a term executable by the same sequential process (with the form $a.P' + a.P''$) can be identified in the same way since the execution of one of them excludes the execution of the other one.

The method that we adopt for assigning names to actions is based on *locations*. The name of an action is given by its location defined as the syntactical position (left or right) with respect to the parallel operators of the term P specifying the system. Therefore in the semantic model of a BL process P the event of starting of an observable action a is represented by a transition labeled by $a^+_@$, where @ is the location of the action, while the event of termination of the action is represented by a transition labeled by $a^-_@$, where the location @ is an "identifier" which uniquely determines which action a is terminating. A consequence of using action locations as static names with respect to the approach of [4] is that we can easily deal with recursion and, since action locations are "reused" by the actions of the same sequential process, we can obtain finite semantic models also in the presence of recursion.

For the choice of the CSP operators of multiway synchronization and hiding, we are forced to deviate from the simpler approach of [4]. In particular, since visible actions a can be obtained by synchronization, we need (as we will see) to record the locations of synchronized actions under execution (started but not terminated yet) and so it is necessary to represent system states as pairs $\langle P, ex \rangle$ composed of a term P and a set ex of locations of actions in execution. Another technical difference with [4] regards the treatment of τ actions. Since τ actions are invisible (weak ST bisimulation abstracts from them), we do not have to associate names with τ actions when they are executed, as in [4]. However, differently from [4] where τ actions are executed in a single step, in our approach we represent the execution of a τ action as the combination of two "invisible events" $\dot{\tau}$ of action start and action termination. Both events are represented by $\dot{\tau}$ since they are invisible and we adopt a notion of weak ST bisimulation which abstracts from $\dot{\tau}$ transitions (instead of abstracting from τ transitions as usual). On the other hand splitting τ actions, as we do for visible actions, adheres to the intuition that the semantics of τ should be isomorphic to that of $a/\{a\}$, where the two events a^+ and a^- are separately hidden.

As we already said in the introduction, in the case of the static name technique the definition of ST equivalence (which is taken from [4]) is rather complex because it must somehow abstract from the particular names chosen for the actions of a given BL process P. The problem of name abstraction derives from the fact that the role of static names is just to distinguish actions of the same type executed in parallel by a single process P, but not to distinguish actions of the same type executed by different processes P and P'. Since the mechanism for assigning names to actions is based on syntactical positions, two terms P and P' that behave in an equivalent way may get different names for corresponding actions. For instance in the two terms $a \,\|_\emptyset\, b$ and $b \,\|_\emptyset\, a$, which should be clearly equivalent, the action a of $a \,\|_\emptyset\, b$ gets name a_l (where l stands for "left") while the action a of $b \,\|_\emptyset\, a$ gets name a_r (where r stands for "right") and similarly for actions b. In order to implement abstraction from action names the definition of weak ST bisimulation must somehow associate the names of the actions used by one process with the names of the corresponding actions used by the other one. More precisely, while checking weak ST bisimulation of two processes P and Q, we make use of an *association history* H where we record associations $(a, @', @'')$ between names of corresponding actions, where a is the type of the two actions, $@'$ is the location of the action a of P, and $@''$ is the location of the corresponding action a of Q. We add a new association $(a, @', @'')$ to H whenever P performs an $a^+_{@'}$ transition and Q performs a corresponding $a^+_{@''}$ transition. After the association $(a, @', @'')$ is established, the two processes P and Q are equivalent only if P performs the transition $a^-_{@'}$ in correspondence to the transition $a^-_{@''}$ of Q. Whenever this happens, the association $(a, @', @'')$ is removed from H. In the following we use $a:(@', @'')$ to stand for $(a, @', @'') \in H$.

3.3.1 Definition of ST Operational Semantics via Static Names

Now we show how the static name technique can be exploited to give operational ST semantics to our basic language BL. First of all, we need to extend the BL syntax to represent actions started but still to be terminated in the state terms of semantic models. The new prefix a^- represents a visible action a that still has to terminate and the new prefix $\dot{\tau}$ represents a silent action τ that still has to terminate. Therefore the metavariable for action prefix is η, which ranges over $Act \cup \{a^- \mid a \in A\} \cup \{\dot{\tau}\}$.

The set BL_{sn} of state terms of BL (where the sn stands for the static name technique) is generated by the following syntax:

$$T ::= \underline{0} \mid X \mid \eta.P \mid P + P \mid T \,\|_S\, T \mid T/L \mid recX.P$$

where $P \in BL$. We denote by $BL_{sn,C}$ the set of processes of BL_{sn}, i.e. closed terms of BL_{sn}.

In order to complete the description of states of semantic models we formally introduce global locations of actions in execution.

Definition 3.1 *The set GLoc of global locations is generated by the following syntax:*

$$@ ::= l@ \mid r@ \mid <@|@> \mid \bullet$$

■

Given a process P such that $P' \,\|_\emptyset\, P''$ is the subterm of P corresponding to its top-level parallel operator, then: an action of P in execution at global location $l@$ is located in the term P' and, inside P', as described by @; an action of P in execution at global location $r@$ is similarly located in P''; and an action of P in execution at global location $<@'|@''>$ is obtained by the synchronization of two actions, one located in the term P' at location $@'$ and the other one in the term P'' at location $@''$. An action of a process P in execution at global location $\bullet$ is an action executed by P not in the scope of any parallel operator.

Example 3.1 Consider $(a^-.P \,\|_\emptyset\, b^-.Q) \,\|_{\{b\}}\, b^-.R$. Then $ll\bullet$ is the global location of the action a in execution in the leftmost sequential process. Moreover $<r\bullet \mid \bullet>$ is the global location of the action in execution obtained by synchronizing the two actions b. ■

Note that the structure of the syntax of terms in BL_{sn} makes evident that, as we already explained, it can never happen that we have two actions in execution with the same global location since in a term of BL_{sn} each sequential process may include at most one action a^-.

We denote by ex a set of pairs $(a, @)$ representing the type and global locations of the actions in execution in a system state (actions started but not terminated yet). ex ranges over the set $Ex = \mathcal{P}(A \times GLoc)$. In the following we use $a:@$ to stand for $(a, @) \in ex$.

The states of the semantic model of a process of BL are pairs $\langle T, ex\rangle$, with $T \in BL_{sn}$ and $ex \in Ex$, where ex includes the global locations of the actions in execution in T. The set of states is denoted by

$\Sigma = BL_{sn,C} \times Ex$, ranged over by s, t. It is necessary to include ex in the representation of a system state because it records the synchronization structure of started actions. Consider for example the term $(a.\underline{0} \,\|_\emptyset\, a.\underline{0}) \,\|_{\{a\}} (a.\underline{0} \,\|_\emptyset\, a.\underline{0})$. In this term two actions may start execution before any action terminates so that we reach the state $(a^-.\underline{0} \,\|_\emptyset\, a^-.\underline{0}) \,\|_{\{a\}} (a^-.\underline{0} \,\|_\emptyset\, a^-.\underline{0})$. In this term we cannot distinguish if the two actions that have started are $a_{\langle l\bullet | l\bullet\rangle}$ and $a_{\langle r\bullet | r\bullet\rangle}$, or $a_{\langle l\bullet | r\bullet\rangle}$ and $a_{\langle r\bullet | l\bullet\rangle}$. Note that the necessity of including global locations of actions in execution in the representation of system states derives from the capability of BL of performing multiway synchronizations. Due to the CSP synchronization mechanism in BL the actions obtained by synchronization are observable and must be assigned a name (a global location). [3]

We denote by $STA_{sn} = \{a^+_@ \mid a \in A \wedge @ \in GLoc\} \cup \{a^-_@ \mid a \in A \wedge @ \in GLoc\}$, the set of observable ST actions, labeling the transitions of semantic models, where $a^+_@$ represents the starting of an action a at the global location @ and $a^-_@$ represents the termination of an action a at the global location @. γ ranges over STA_{sn}. The set of all ST actions is denoted by $STAct_{sn} = STA_{sn} \cup \{\dot{\tau}\}$, where $\dot{\tau}$ is a distinguished action denoting an internal computation. θ ranges over $STAct_{sn}$.

The operational semantics for processes of BL produces a transition system labeled over $STAct_{sn}$ whose states are pairs in Σ. The SOS operational rules are presented in Table 3.1. The following auxiliary definitions are used in Table 3.1. The function $type : STAct_{sn} \longrightarrow Act$ extracts the type of an ST action and is defined in the obvious way. The function $l : Ex \longrightarrow Ex$, which, given a set of action locations ex, evaluates the set $l(ex)$ of the lefthand components of the action locations in ex, is defined by $l(ex) = \{(a,@) \mid (a, l@) \in ex \vee \exists @'. (a, <@|@'>) \in ex\}$. Similarly the function $r : Ex \longrightarrow Ex$ is defined by $r(ex) = \{(a,@) \mid (a, r@) \in ex \vee \exists @'. (a, <@'|@>) \in ex\}$. The function $h : Ex \times \mathcal{P}(A) \longrightarrow Ex$, which, given a set of action locations ex, evaluates the set $h(ex, L)$ of the action locations in ex whose type is not in L (h stands for "hiding"), is defined by $h(ex, L) = \{(a,@) \in ex \mid a \notin L\}$.

Let us briefly comment on the operational rules.

The meaning of the rules for prefix is the following. $\langle a.P, \emptyset\rangle$, where the action a still has to be executed (hence the set of action under execution is empty), becomes $\langle a^-.P, \{a : \bullet\}\rangle$ after the event of start of action a, where a is the only action under execution. $\langle a^-.P, \{a : \bullet\}\rangle$ becomes $\langle P, \emptyset\rangle$ after the event of termination of action a (hence the set of action under execution becomes empty again). $\langle \tau.P, \emptyset\rangle$ becomes $\langle \dot{\tau}.P, \emptyset\rangle$ and then $\langle P, \emptyset\rangle$ in a similar way.

The meaning of the operational rules for the parallel operator is the following:

- A starting move $a^+_{l@}$, with $a \notin S$, is made by $\langle T \,\|_S\, U, ex\rangle$ whenever $\langle T, l(ex)\rangle$ makes an $a^+_@$ move, where $l(ex)$ computes the location of the action in execution in the lefthand component T. Symmetrically a starting move $a^+_{r@}$, with $a \notin S$, is derived from an $a^+_@$ move of U.

[3] Given that we represent a system state with a pair $\langle T, ex\rangle$, it is not strictly necessary to represent in T visible actions to be terminated as a^- instead of simply using a, since actions to be terminated are uniquely determined by ex. For clarity reasons we prefer to stick to the notation a^-.

$$\langle a.P, \emptyset\rangle \xrightarrow{a^+_\bullet} \langle a^-.P, \{a:\bullet\}\rangle \qquad \langle \tau.P, \emptyset\rangle \xrightarrow{\dot\tau} \langle \dot\tau.P, \emptyset\rangle$$

$$\langle a^-.P, \{a:\bullet\}\rangle \xrightarrow{a^-_\bullet} \langle P, \emptyset\rangle \qquad \langle \dot\tau.P, \emptyset\rangle \xrightarrow{\dot\tau} \langle P, \emptyset\rangle$$

$$\frac{\langle P, \emptyset\rangle \xrightarrow{\theta} \langle T, ex\rangle}{\langle P+Q, \emptyset\rangle \xrightarrow{\theta} \langle T, ex\rangle} \qquad \frac{\langle Q, \emptyset\rangle \xrightarrow{\theta} \langle U, ex\rangle}{\langle P+Q, \emptyset\rangle \xrightarrow{\theta} \langle U, ex\rangle}$$

$$\frac{\langle T, l(ex)\rangle \xrightarrow{a^+_@} \langle T', ex'\rangle}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^+_{l@}} \langle T' \parallel_S U, ex \cup \{a: l@\}\rangle}\ a \notin S \qquad \frac{\langle T, l(ex)\rangle \xrightarrow{\dot\tau} \langle T', ex'\rangle}{\langle T \parallel_S U, ex\rangle \xrightarrow{\dot\tau} \langle T' \parallel_S U, ex\rangle}$$

$$\frac{\langle T, l(ex)\rangle \xrightarrow{a^-_@} \langle T', ex'\rangle \quad a: l@ \in ex}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^-_{l@}} \langle T' \parallel_S U, ex - \{a: l@\}\rangle}\ a \notin S$$

$$\frac{\langle U, r(ex)\rangle \xrightarrow{a^+_@} \langle U', ex'\rangle}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^+_{r@}} \langle T \parallel_S U', ex \cup \{a: r@\}\rangle}\ a \notin S \qquad \frac{\langle U, r(ex)\rangle \xrightarrow{\dot\tau} \langle U', ex'\rangle}{\langle T \parallel_S U, ex\rangle \xrightarrow{\dot\tau} \langle T \parallel_S U', ex\rangle}$$

$$\frac{\langle U, r(ex)\rangle \xrightarrow{a^-_@} \langle U', ex'\rangle \quad a: r@ \in ex}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^-_{r@}} \langle T \parallel_S U', ex - \{a: r@\}\rangle}\ a \notin S$$

$$\frac{\langle T, l(ex)\rangle \xrightarrow{a^+_{@'}} \langle T', ex'\rangle \quad \langle U, r(ex)\rangle \xrightarrow{a^+_{@''}} \langle U', ex''\rangle}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^+_{<@'|@''>}} \langle T' \parallel_S U', ex \cup \{a: <@'|@''>\}\rangle}\ a \in S$$

$$\frac{\langle T, l(ex)\rangle \xrightarrow{a^-_{@'}} \langle T', ex'\rangle \quad \langle U, r(ex)\rangle \xrightarrow{a^-_{@''}} \langle U', ex''\rangle \quad a: <@'|@''> \in ex}{\langle T \parallel_S U, ex\rangle \xrightarrow{a^-_{<@'|@''>}} \langle T' \parallel_S U', ex - \{a: <@'|@''>\}\rangle}\ a \in S$$

$$\frac{\langle T, ex\rangle \xrightarrow{\theta} \langle T', ex'\rangle}{\langle T/L, h(ex, L)\rangle \xrightarrow{\theta} \langle T'/L, h(ex', L)\rangle}\ type(\theta) \notin L$$

$$\frac{\langle T, ex\rangle \xrightarrow{\gamma} \langle T', ex'\rangle}{\langle T/L, h(ex, L)\rangle \xrightarrow{\dot\tau} \langle T'/L, h(ex, L)\rangle}\ type(\gamma) \in L$$

$$\frac{\langle P\{recX.P/X\}, \emptyset\rangle \xrightarrow{\theta} \langle T, ex\rangle}{\langle recX.P, \emptyset\rangle \xrightarrow{\theta} \langle T, ex\rangle}$$

Table 3.1: Rules for the Static Name Technique

- A termination move $a^-_{l@}$, with $a \notin S$, made by $\langle T \parallel_S U, ex\rangle$ such that $a : l@ \in ex$ is originated from the unique move $a^-_@$ of $\langle T, l(ex)\rangle$, that exists because $a : @ \in l(ex)$. Symmetrically a termination move $a^-_{r@}$, with $a \notin S$, is derived from an $a^-_@$ move of U.
- The silent $\dot{\tau}$ moves of $\langle T \parallel_S U, ex\rangle$ are derived from the $\dot{\tau}$ moves of $\langle T, l(ex)\rangle$ and of $\langle U, r(ex)\rangle$.
- A starting move $a^+_{<@'|@''>}$, with $a \in S$, is made by $\langle T \parallel_S U, ex\rangle$ whenever $\langle T, l(ex)\rangle$ makes an $a^+_@$ move and $\langle U, r(ex)\rangle$ makes an $a^+_@$ move.
- A termination move $a^-_{<@'|@''>}$, with $a \in S$, made by $\langle T \parallel_S U, ex\rangle$ such that $a : <@'|@''> \in ex$ is originated from the unique move $a^-_{@'}$ of $\langle T, l(ex)\rangle$, that exists because $a : @' \in l(ex)$, and from the unique move $a^-_{@''}$ of $\langle U, r(ex)\rangle$, that exists because $a : @'' \in r(ex)$.

Note that in all the rules for parallel composition we look at the possible moves of $\langle T, l(ex)\rangle$ and/or $\langle U, r(ex)\rangle$ just for the purpose of establishing the possible transitions for $\langle T \parallel_S U, ex\rangle$ and derived terms T' and/or U'. Once a transition and a target term for $\langle T \parallel_S U, ex\rangle$ is derived, ex is simply updated by including (if it is a starting move) or eliminating (if it is a termination move) the action location labeling the transition.

As far as the meaning of the operational rules for the hiding operator is concerned, the ST moves of a term $\langle T/L, ex'\rangle$ are determined as follows. When $\langle T, ex\rangle$ makes a θ ST move whose type is not in L, such a move is made also by $\langle T/L, h(ex, L)\rangle$, where $h(ex, L)$ expresses the locations of the actions in execution in T/L in terms of those of T by eliminating the locations of the actions which are hidden (whose type is in L). When, instead, $\langle T, ex\rangle$ makes an observable γ ST move whose type is in L, $\langle T/L, h(ex, L)\rangle$ makes a $\dot{\tau}$ move.

The rules for the choice and the recursion operators are just standard.

In the following we will denote by Σ_R the subset of Σ of the states reachable from a process of BL_C, i.e. $\Sigma_R = \{s \in \Sigma \mid \exists P \in BL_C, k \in \mathbb{N}, \{s_i\}_{i<k}, \{\theta_i\}_{i\leq k}. \langle P, \emptyset\rangle \xrightarrow{\theta_1} s_1 \xrightarrow{\theta_2} \ldots \xrightarrow{\theta_{k-1}} s_{k-1} \xrightarrow{\theta_k} s\}$. [4]

We define the ST semantic model via static names of a process $P \in BL_C$ to be the transition system obtained by considering the states of Σ_R reachable from $\langle P, \emptyset\rangle$ according to the ST operational semantics via static names, and the transitions among such states.

Example 3.2 In Fig. 3.1 we depict the ST semantic model of $recX.a.X \parallel_\emptyset recX.a.X$ obtained by applying the static name technique. ■

The following theorem shows that our approach is indeed suitable to cope with recursive processes. We consider the interleaving semantics of a process P to be the transition system including the processes in BL_C reachable from P according to the standard operational rules of [122, 109], and the transitions among such processes.

Theorem 3.1 *The interleaving semantics of a process $P \in BL_C$ is finite state if and only if the ST semantics of P via static names is finite state.*

[4] See Chapter 13 for a direct characterization of "correct" states (states in Σ_R) not based on the operational semantics.

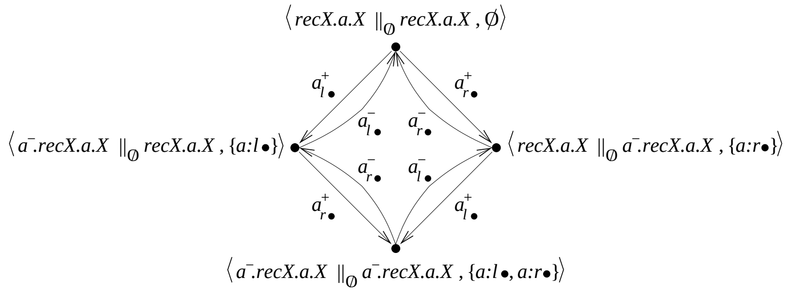


Figure 3.1: Example of Recursion with the Static Name Technique

Proof:

Proof of "$\Longrightarrow$" Let us consider the term P' obtained by syntactically substituting $\mu.\mu$ for each action μ occurring in the term P. Since the interleaving semantics of P is finite state, it easily derives that also the interleaving semantics of P' is finite state. It is easy to see that if, inside the state terms of the ST semantics of P, we syntactically replace each ST action a^- or $\dot{\tau}$ with its type and we discard the sets ex of locations of the actions in execution from states, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because the number of different locations that the actions of P may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.

Proof of "$\Longleftarrow$" Starting from the ST semantics of P, we consider only those paths where the termination transition for an action immediately follows the starting transition for that action, thus removing all transitions and states not reached in this way. The semantic model that we obtain is just the interleaving model of P where each transition is split into two transitions and a state is added between them. $\square$

In the following we will refer to a process which is finite state for interleaving and ST semantics simply with "finite state process".

Now we present a simple syntactic criterion which constitutes a sufficient condition for a process of BL_C to be finite state. In the following corollary we call "static" [122] the operators of parallel composition and hiding only.

Corollary 3.1 *Let $P \in BL_C$ be a process s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator. Then P is a finite state process.*

Note that the class of processes considered in this corollary includes strictly the class of nets of automata, i.e. terms where no static operator occurs in the scope of any recursion.

3.3.2 Weak ST Bisimulation via Static Names

As we already explained, the equivalence notion we consider over BL terms is that presented in [4]. In particular the equivalence between a pair of states is established depending on a given association history, i.e. a correspondence between their sets of actions in execution.

We denote by la a *location association*, whose elements are associations $(@', @'')$. la ranges over the set LA of partial bijections from $GLoc$ to $GLoc$. An association history H is a relation from A to $GLoc \times GLoc$ such that $\forall a \in A. H_a \in LA$, [5] i.e. H is a set including an independent location association for each different action type. We denote by $\mathcal{H}$ the set of all association histories. [6]

As usual we let $\stackrel{\theta}{\Longrightarrow}$ stand for $(\xrightarrow{\dot{\tau}})^* \xrightarrow{\theta} (\xrightarrow{\dot{\tau}})^*$, i.e. a sequence of transitions including a single θ transition and any number of $\dot{\tau}$ transitions. Moreover $\stackrel{\varepsilon}{\Longrightarrow}$ stands for $(\xrightarrow{\dot{\tau}})^*$, i.e. a possibly empty sequence of $\dot{\tau}$ transitions.

For two states s and t to be equivalent with respect to an association history H we require that:

1. H must be an association history compatible with s and t, i.e. all actions in execution (started but not terminated yet) in s and t must be recorded in the history H. Formally, an association history H is *compatible* with a pair of states $(\langle T, ex' \rangle, \langle U, ex'' \rangle)$ if and only if $\forall a \in A. \, dom(H_a) = ex'_a \wedge range(H_a) = ex''_a$.

2. Every start move of s, $s \xrightarrow{a^+_@} s'$ is matched by a corresponding move $t \stackrel{a^+_{@'}}{\Longrightarrow} t'$ of t such that s' and t' are equivalent with respect to the augmented history $H \cup \{a : (@, @')\}$. For the start moves of t we have a symmetric argument.

3. Every termination move of s, $s \xrightarrow{a^-_@} s'$ is matched by a termination move $t \stackrel{a^-_{@'}}{\Longrightarrow} t'$ of t which is consistent with the history H, i.e. $a : (@, @') \in H$, and such that s' and t' are equivalent with respect to the diminished history $H - \{a : (@, @')\}$. For the termination moves of t we have a symmetric argument.

4. Silent $\dot{\tau}$ moves must be matched as in standard weak bisimulation [122].

Formally, weak ST bisimulation is defined by means of a family of bisimulations (binary relations) $\mathcal{B}_H$, each indexed by a an association history H, as follows. First of all, let us say that a $\mathcal{H}$-indexed family of binary relations $\{\mathcal{B}_H \mid H \in \mathcal{H}\}$ over Σ_R is *symmetric* if and only if $(s,t) \in \mathcal{B}_H$ implies $(t,s) \in \mathcal{B}_{\overline{H}}$, where $\overline{H} = \{a : (@'', @') \mid a : (@', @'') \in H\}$.

[5]Given a relation R from A to B, we denote by R_a, with $a \in A$, the set $\{b \in B \mid (a,b) \in R\}$.

[6]Since we use global locations as static names for actions and, as we already observed, it can never happen that, in a state, we have two actions in execution with the same global location (even if they are of different type), it is not strictly necessary to include action types in association histories. We prefer to do this in order to adhere to the definition of ST bisimulation of [4] and for clarity reasons.

Definition 3.2 *A weak ST bisimulation family is a symmetric $\mathcal{H}$-indexed family* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ *of binary relations over Σ_R such that for each $H \in \mathcal{H}$ and $s, t \in \Sigma_R$ we have that $s\, \mathcal{B}_H\, t$ implies:*

1. *H is compatible with (s, t)*
2. $s \xrightarrow{a^+_{@}} s'$ *implies* $\exists t' \in \Sigma_R, @' \in GLoc.\ t \xRightarrow{a^+_{@'}} t' \wedge (s', t') \in \mathcal{B}_{H \cup \{a:(@,@')\}}$
3. $s \xrightarrow{a^-_{@}} s'$ *implies* $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@, @') \in H \wedge t \xRightarrow{a^-_{@'}} t' \wedge$ $(s', t') \in \mathcal{B}_{H - \{a:(@,@')\}}$
4. $s \xrightarrow{\dot{\tau}} s'$ *implies* $\exists t' \in \Sigma_R.\ t \xRightarrow{\varepsilon} t' \wedge (s', t') \in \mathcal{B}_H$ ■

Let $\{\mathbf{B_i} \mid i \in I\}$ with $\mathbf{B_i} = \{\mathcal{B}_{H,i} \mid H \in \mathcal{H}\}$ be the set of all weak ST bisimulation families. Then it is easy to verify that there exists a largest weak ST bisimulation family $\approx_{\mathbf{sn}}$ which is given by $\approx_{\mathbf{sn}} = \{\approx_{sn,H} | \approx_{sn,H} = \bigcup_{i \in I} \mathcal{B}_{H,i} \wedge H \in \mathcal{H}\}$.

Definition 3.3 *Given $P, Q \in BL_C$, P is observationally ST congruent to Q (written $P \simeq_{sn} Q$), if and only if:*

- $\langle P, \emptyset \rangle \xrightarrow{a^+_{@}} s$ *implies* $\exists t \in \Sigma_R, @' \in GLoc.\ \langle Q, \emptyset \rangle \xRightarrow{a^+_{@'}} t \wedge (s, t) \in \approx_{sn,\{a:(@,@')\}}$
- $\langle Q, \emptyset \rangle \xrightarrow{a^+_{@}} t$ *implies* $\exists s \in \Sigma_R, @' \in GLoc.\ \langle P, \emptyset \rangle \xRightarrow{a^+_{@'}} s \wedge (s, t) \in \approx_{sn,\{a:(@,@')\}}$
- $\langle P, \emptyset \rangle \xrightarrow{\dot{\tau}} s$ *implies* $\exists t \in \Sigma_R.\ \langle Q, \emptyset \rangle \xRightarrow{\dot{\tau}} t \wedge (s, t) \in \approx_{sn,\emptyset}$
- $\langle Q, \emptyset \rangle \xrightarrow{\dot{\tau}} t$ *implies* $\exists s \in \Sigma_R.\ \langle P, \emptyset \rangle \xRightarrow{\dot{\tau}} s \wedge (s, t) \in \approx_{sn,\emptyset}$ ■

The following theorem shows that $\simeq_{sn}$ is preserved by all the operators of BL, including recursion.

Theorem 3.2 *$\simeq_{sn}$ is a congruence for BL.*

Proof: The most relevant cases are those of the parallel composition operator and of the recursion operator.

As far as the parallel operator is concerned, in order to show that $\forall P', P'', Q \in BL_C, S \subseteq A$ we have that $P' \simeq_{sn} P''$ implies $P' \parallel_S Q \simeq_{sn} P'' \parallel_S Q$, it suffices to verify that, given $S \subseteq A$, the symmetric $\mathcal{H}$-indexed family

$$\begin{aligned}\mathcal{B}_H = \{(\langle T' \parallel_S U, ex' \rangle, \langle T'' \parallel_S U, ex'' \rangle) \in \Sigma_R \times \Sigma_R \mid \\ H \text{ is compatible with } (\langle T' \parallel_S U, ex' \rangle, \langle T'' \parallel_S U, ex'' \rangle) \wedge \\ Id_{r(ex')} = r(H) \wedge (\langle T', l(ex') \rangle, \langle T'', l(ex'') \rangle) \in \approx_{sn,l(H)}\}\end{aligned}$$

is a weak ST bisimulation family, where $l(H)$ is defined as

$$\begin{aligned}l(H) = \{a : (@, @') \mid a \notin S \wedge a : (l@, l@') \in H\} \cup \\ \{a : (@, @') \mid a \in S \wedge \exists @'', @''' \in GLoc.\, a : (<@|@''>, <@'|@'''>) \in H\},\end{aligned}$$

$r(H)$ is similarly defined as

$r(H) = \{a : (@, @') \mid a \notin S \wedge a : (r@, r@') \in H\} \cup$
$\{a : (@, @') \mid a \in S \wedge \exists @'', @''' \in GLoc.\, a : (<@''|@>, <@'''|@'>) \in H\}$,
and Id_{ex} is defined as $Id_{ex} = \{a : (@, @) \mid a : @ \in ex\}$.

As far as the recursion operator is concerned, in order to show that for all $P', P'' \in BL$ containing at most the variable X free we have that $P' \simeq_{sn} P''$ (meaning $\forall\, Q \in BL_C .\, P'\{Q/X\} \simeq_{sn} P''\{Q/X\}$, as usual [122]) implies $recX.P' \simeq_{sn} recX.P''$, it suffices to verify that the family (one for each $ex \in Ex$) of binary relations over Σ_R

$\mathcal{B}_{Id_{ex}} = \{(\langle T\{recX.P'/X\}, ex\rangle, \langle T\{recX.P''/X\}, ex\rangle) \in \Sigma_R \times \Sigma_R \mid T \in BL_{sn}\}$

is such that, if $(s, t) \in \mathcal{B}_{Id_{ex}}$ then:

- $s \xrightarrow{a^+_@} s'$ implies $\exists t', t'' \in \Sigma_R .\, t \xRightarrow{a^+_{@'}} t' \wedge s' \; \mathcal{B}_{Id_{ex \cup \{a:@\}}} \; t'' \; \approx_{sn, Id_{ex} \cup \{a:(@,@')\}} \; t'$
- $s \xrightarrow{\dot{\tau}} s'$ implies $\exists t', t'' \in \Sigma_R .\, t \xRightarrow{\dot{\tau}} t' \wedge s' \; \mathcal{B}_{Id_{ex}} \; t'' \; \approx_{sn, Id_{ex}} \; t'$
- $s \xrightarrow{a^-_@} s$ implies $\exists t' \in \Sigma_R .\, t \xrightarrow{a^-_@} t' \wedge s' \; \mathcal{B}_{Id_{ex - \{a:@\}}} \; t'$

This is done by induction on the depth of the inference by which a transition $s \xrightarrow{\theta} s'$ is inferred, similarly to [122]. It is easy to see that (similarly to the notion of "bisimulation up to $\approx$" of [122]) the three conditions above (together with the three symmetric conditions for the moves of t) imply that for all $ex \in Ex$ we have $\mathcal{B}_{Id_{ex}} \subseteq \approx_{Id_{ex}}$. Hence, by taking $T \equiv X$ and $ex = \emptyset$, the conditions above imply that $recX.P' \simeq_{sn} recX.P''$.
□

From Theorem 3.1 it easily derives the decidability of observational ST congruence over processes possessing a finite state interleaving semantics.

Theorem 3.3 *Observational ST congruence is decidable over processes of BL_C which possess a finite state interleaving semantics.*

Proof: Since the ST semantic model via static names of BL_C processes with a finite state interleaving semantics is finite state, observational ST congruence can be simply decided via a partition refinement algorithm similar to that of [135]. In particular given $P, Q \in BL_C$ we establish if $P \simeq_{sn} Q$ as follows. First of all we explicitly compute weak transitions among the states of the static name ST semantics of P, and similarly for Q. Then we evaluate $\approx_{sn,H}$ for each association history $H \in \mathcal{H}$ such that there exist a state s of the semantics of P and a state t of the semantics of Q such that H is compatible with (s, t). This is done by modifying partition refinement algorithm of [135] as follows. We consider the states in the semantic models of P and Q to be distinguished and for each association history $H \in \mathcal{H}$ satisfying the condition above we consider a partition over the states of P and Q which are compatible with H. Therefore we deal with a set of partitions parameterized over H, each divided into a set of classes C_H. At the beginning of the algorithm each partition consist of a single class C_H containing all the states. Then the algorithm proceeds similarly to [135] by iteratively refining all the classes currently considered (independently on the partition they belong to) until the $\mathcal{H}$-indexed family of partitions is a weak ST bisimulation family. □

3.4 ST Semantics via Dynamic Names

The *dynamic name* technique is based on the idea of dynamically assigning, during the evolution of the system, a new name to each action that starts execution, on the basis of the names assigned to the actions already started. *Names* are indexes $i \in \mathbb{N}^+$ that distinguish actions of the same type. In particular the event of starting of an observable action a is represented in semantic models by a transition labeled by a_i^+ where i is the minimum index not already used by the other actions a that have started but not terminated yet. This rule for computing indexes guarantees that names are reused and that finite models can be obtained also in the presence of recursion. The termination of the action is simply represented by a transition labeled by a_i^-, where the "identifier" i uniquely determines which action a is terminating. Since the method to compute the index for a starting action is fixed, it turns out that actions of processes that perform the same execution traces of actions get the same names. As a consequence, contrary to the static name technique, ST bisimilarity can simply be checked by applying standard bisimilarity to the semantic models of processes.

Moreover the dynamic name technique we introduce here allows us to dynamically assign names to actions, according to the rule formerly described, via SOS semantics (hence in a *compositional way*) through the idea of *levelwise renaming*. In order to obtain compositionality, it is necessary to determine, e.g. in the case of the parallel composition operator, the computations of $P \,\|\, Q$ from the computations of P and Q. This is done by parameterizing in state terms each parallel operator with a mapping M. For every action a started by $P \,\|_{S,M}\, Q$, M records the association between the name a_i, generated according to the fixed rule above for identifying a at the level of $P \,\|_{S,M}\, Q$, and the name a_j (which in general is different from a_i), generated according to the same rule for identifying the same action a inside P (or Q). In this way when, afterwards, such an action a terminates in P (or Q) the name a_j can be re-mapped to the correct name a_i at the level of $P \,\|_{S,M}\, Q$, by exploiting the information included in M. In M the action a of $P \,\|_{S,M}\, Q$ which gets index i is uniquely identified by expressing the unique name j it gets in P or in Q and the "location" of the process that executes it: *left* if P, *right* if Q. Such an association is represented inside M by the triple (a, i, loc_j) with $a \in A$, indexes $i, j \in \mathbb{N}^+$ and location $loc \in Loc = \{l, r\}$, where "l" stands for left and "r" for right. In the following we use $a\!:\!(i, loc_j)$ to stand for $(a, i, loc_j) \in M$.

3.4.1 Definition of ST Operational Semantics via Dynamic Names

Now we show how the dynamic name technique can be exploited to give operational ST semantics to our basic language *BL*. We need a richer syntax to represent the states of semantic models. We denote by $STA_{dn} = \{a_i^+ \mid a \in A \wedge i \in \mathbb{N}^+\} \cup \{a_i^- \mid a \in A \wedge i \in \mathbb{N}^+\}$ (where the *dn* stands for the dynamic name technique), the set of observable ST actions, where a_i^+ represents the beginning of the action a identified with i and a_i^- represents the ending of the action a identified with i. γ ranges over STA_{dn}. [7] The set of all

[7] For simplicity and reuse, we usually denote the meta-variable for a set, which is the version with dynamic names of a set already used in Sect. 3.3 (e.g. STA_{dn} corresponds to STA_{sn}), by the same name used in the case of static names.

ST actions is denoted by $STAct_{dn} = STA_{dn} \cup \{\dot{\tau}\}$. θ ranges over $STAct_{dn}$ and η ranges over $Act \cup STAct_{dn}$. We denote by ia an *index association*, whose elements are associations (i, loc_j). ia ranges over the set IA of partial bijections from $\mathbb{N}^+$ to $Loc \times \mathbb{N}^+$. A mapping M is a relation from A to $\mathbb{N}^+ \times (Loc \times \mathbb{N}^+)$ such that $\forall a \in A.\, M_a \in IA$, i.e. M is a set including an independent index association for each different action type.

The set BL_{dn} of state terms of BL is generated by the following syntax: [8]

$$T ::= \underline{0} \mid X \mid \eta.T \mid T + T \mid T \parallel_{S,M} T \mid T/L \mid recX.T$$

We denote by $BL_{dn,C}$ the set of processes, i.e. closed terms, of BL_{dn}. We consider the operators "$\parallel_S$" occurring in a BL process P as being "$\parallel_{S,\emptyset}$" when P is regarded as a state.

$$\frac{T \xrightarrow{\theta} T'}{T + U \xrightarrow{\theta} T'} \qquad \frac{U \xrightarrow{\theta} U'}{T + U \xrightarrow{\theta} U'}$$

$$\frac{T \xrightarrow{\theta} T'}{T/L \xrightarrow{\theta} T'/L}\ type(\theta) \notin L \qquad \frac{T \xrightarrow{\gamma} T'}{T/L \xrightarrow{\dot{\tau}} T'/L}\ type(\gamma) \in L$$

$$\frac{T\{recX.T/X\} \xrightarrow{\theta} T'}{recX.T \xrightarrow{\theta} T'}$$

Table 3.2: Standard Rules

The semantics of state terms in $BL_{dn,C}$ produces a transition system labeled over $STAct_{dn}$. The operational rules for the operators "$+$", "$/L$" and "$recX$" are the standard ones and are presented in Table 3.2. The operational rules for the operators "." and "$\parallel_{S,M}$" are presented in Table 3.3. In Tables 3.2 and 3.3 we use function $type : STAct_{dn} \longrightarrow Act$, defined in the obvious way.

The meaning of the rules for computing the starting moves a_j^+ of $T \parallel_{S,M} U$ for observable actions $a \notin S$ is the following. When T performs a_i^+ then a new index $n(M_a)$ is determined for identifying the action a at the level of "$\parallel_{S,M}$" and the new association $(n(M_a), l_i)$ is added to M_a. Function n computes the new index by choosing the minimum index not used by the other actions a already in execution: $n(ia) = min\{k \mid k \notin dom(ia)\}$, where $ia \in IA$. Symmetrically for a move a_i^+ of U.

The meaning of the rules for computing the termination moves a_j^- of $T \parallel_{S,M} U$ for observable actions $a \notin S$ is the following. When T performs a_i^- the action of type a with index j associated to l_i in M terminates at the level of "$\parallel_{S,M}$". Symmetrically for a move a_i^- of U.

[8] The syntax of state terms is quite expressive and, as we will see, is not limited to terms reachable from a process of BL. For instance it can express terms like $a_1^+.a_1^-.\underline{0}$ or $a_1^-.\underline{0} + a_2^-.\underline{0}$ which are not reachable from a BL process, but, as we will show, are needed to axiomatize ST weak bisimulation.

$$a.T \xrightarrow{a_1^+} a_1^-.T \qquad \tau.T \xrightarrow{\dot{\tau}} \dot{\tau}.T \qquad \theta.T \xrightarrow{\theta} T$$

$$\frac{T \xrightarrow{a_i^+} T'}{T \,\|_{S,M}\, U \xrightarrow{a^+_{n(M_a)}} T' \,\|_{S,M\cup\{a:(n(M_a),l_i)\}}\, U} \; a \notin S \qquad \frac{T \xrightarrow{\dot{\tau}} T'}{T \,\|_{S,M}\, U \xrightarrow{\dot{\tau}} T' \,\|_{S,M}\, U}$$

$$\frac{U \xrightarrow{a_i^+} U'}{T \,\|_{S,M}\, U \xrightarrow{a^+_{n(M_a)}} T \,\|_{S,M\cup\{a:(n(M_a),r_i)\}}\, U'} \; a \notin S \qquad \frac{U \xrightarrow{\dot{\tau}} U'}{T \,\|_{S,M}\, U \xrightarrow{\dot{\tau}} T \,\|_{S,M}\, U'}$$

$$\frac{T \xrightarrow{a_i^-} T' \quad a:(j,l_i) \in M}{T \,\|_{S,M}\, U \xrightarrow{a_j^-} T' \,\|_{S,M-\{a:(j,l_i)\}}\, U} \; a \notin S$$

$$\frac{U \xrightarrow{a_i^-} U' \quad a:(j,r_i) \in M}{T \,\|_{S,M}\, U \xrightarrow{a_j^-} T \,\|_{S,M-\{a:(j,r_i)\}}\, U'} \; a \notin S$$

$$\frac{T \xrightarrow{\gamma} T' \quad U \xrightarrow{\gamma} U'}{T \,\|_{S,M}\, U \xrightarrow{\gamma} T' \,\|_{S,M}\, U'} \; \mathit{type}(\gamma) \in S$$

Table 3.3: Rules for the Dynamic Name Technique

The rule for computing the synchronizing moves γ (a_j^+ or a_j^-) of $T \,\|_{S,M}\, U$, where $\mathit{type}(\gamma) \in S$ (i.e. a is in S), requires that the two synchronizing actions have the same index and produces an action with that index. Note that:

- since actions of a given type $a \in S$ are numbered independently from actions of other types,
- since the rule for generating new indexes for actions a starting in T and U is the same, and
- since actions of type a are required to start and terminate in T and U at the same time and with the same index,

then the set of indexes of actions a in execution in T and U is always the same and it is never possible for T and U to start actions with different indexes.

We will use the following shorthand notation to denote a set of triples $a : (i, loc_j)$ referring to the same action type a. Given an index association $ia \in IA$ we will let $a : ia$ stand for the set $\{a : (i, loc_j) \mid (i, loc_j) \in ia\}$. For instance we will use $a : \{(1, l_1), (2, r_1)\}$ to stand for $\{a : (1, l_1), a : (2, r_1)\}$.

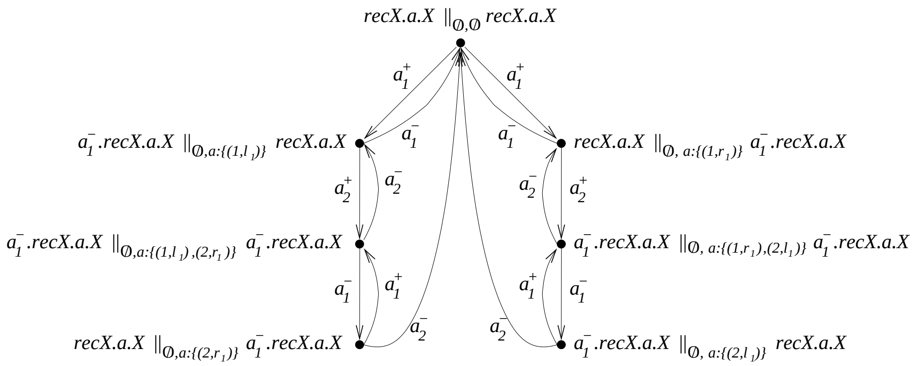


Figure 3.2: Example of Recursion with the Dynamic Name Technique

Example 3.3 In Fig. 3.2 we depict the ST semantic model of $recX.a.X \parallel_{\emptyset} recX.a.X$ obtained by applying the dynamic name technique. ■

Similarly to the static name technique, the following theorem shows that our approach is suitable to cope with recursive processes.

Theorem 3.4 *The interleaving semantics of a process $P \in BL_C$ is finite state if and only if the ST semantics of P via dynamic names is finite state.*

Proof:

Proof of "$\Longrightarrow$" As in the proof of Theorem 3.1 we consider the finite state term P' obtained by syntactically substituting $\mu.\mu$ for each action μ occurring in the term P. If, inside the state terms of the ST semantics of P, we syntactically replace each ST action with its type and we discard the sets M in the operators $\parallel_{S,M}$, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because, due to the policy of using the minimum index available for a starting action, the maximum index an action may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.

Proof of "$\Longleftarrow$" The same as in the proof of Theorem 3.1. □

As a consequence the simple syntactic criterion for BL processes that are guaranteed to be finite state we considered in Sect. 3.3.1 (processes P s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator), which strictly includes the class of nets of automata, is valid also for the dynamic name technique.

3.4.2 Weak ST Bisimulation via Dynamic Names

The equivalence notion we consider over terms of BL_{dn}, denoted by $\simeq_{dn}$, is the standard notion of *observational congruence* extended to open terms [122], where the alphabet of visible actions is STA_{dn} and hidden actions are $\dot{\tau}$ actions.

We have the following theorem of consistency with the static name ST semantics.

Theorem 3.5 *Given two processes P and Q of BL_C we have that $P \simeq_{dn} Q$ if and only if $P \simeq_{sn} Q$.*

Proof: See Appendix B. □

From Theorems 3.5 and 3.2 it immediately derives that $\simeq_{dn}$ is preserved by all the operators of BL, including recursion.

Corollary 3.2 *$\simeq_{dn}$ is a congruence for BL.* ■

As far as decidability of $\simeq_{dn}$ is concerned, obviously (since we rely on the standard notion of observational congruence) $\simeq_{dn}$ is decidable over processes of BL_C which possess a finite-state interleaving semantics. This can be seen also as a corollary of Theorems 3.5 and 3.3.

3.4.3 Axiomatization of Weak ST Bisimulation via Dynamic Names

Now we will show that with the dynamic name technique it is easy to define an axiomatization for $\simeq_{dn}$ which is complete over finite state processes of BL_{CG} by following an approach similar to the axiomatization of standard bisimulation for process algebras with a structural operational interleaving semantics.

The axiom system $\mathcal{A}_{BL}$ for $\simeq_{dn}$ on BL_{dn} terms is formed by the standard axioms presented in Table 3.4 and by the axioms of Table 3.5, which are specific for the dynamic name technique. "$\lfloor\!\lfloor$" and "$|$" denote the left merge and synchronization merge operators [18, 1], respectively.

The axiom (Par) is the standard one except that when the position of processes T and U is exchanged we must invert left and right inside M. More precisely, the inverse $\overline{M}$ of a mapping M is defined by $\overline{M} = \{a : (i, r_j) \mid a : (i, l_j) \in M\} \cup \{a : (i, l_j) \mid a : (i, r_j) \in M\}$. Axioms $(LM5)$ and $(LM6)$ just reflect the operational rules of the parallel operator for an independent move of the lefthand process. As far as axiom $(Rec2)$ is concerned, we define a variable X to be sequential in a term if each free occurrence of X in that term is in the scope of operators $\theta.T$, $T' + T''$ and $recX.T$ only. Moreover we assume the standard definition of [122] for strong guardedness of sequential variables.

In Table 3.6 we present the operational rules for "$\lfloor\!\lfloor_{S,M}$" and "$|_{S,M}$", which are obtained by applying levelwise renaming of actions (as for the parallel operator) to operational rules similar to those considered in [1]. In particular the definition of the operational rule for "$|_{S,M}$" allows for actions "$\dot{\tau}$" to be skipped [1], as reflected by axiom $(SM5)$.

Theorem 3.6 *The axioms of $\mathcal{A}_{BL}$ are sound for $\simeq_{dn}$ over BL_{dn} terms.*

Proof: A simple consequence of the soundness of the axiomatization of [1]. □

Once we have defined a sequential state to be a state that can be built using only $\underline{0}$, X and operators $\theta.T$, $T' + T''$, $recX.T$, and assumed the standard definition of [122] for strong guardedness of sequential states, we have the following theorem.

$(A1)$	$T+U = U+T$		$(A2)$	$(T+U)+R = T+(U+R)$
$(A3)$	$T+T = T$		$(A4)$	$T+\underline{0} = T$
$(Tau1)$	$\theta.\dot{\tau}.T = \theta.T$		$(Tau2)$	$T+\dot{\tau}.T = \dot{\tau}.T$
$(Tau3)$	$\gamma.(T+\dot{\tau}.U)+\gamma.U = \gamma.(T+\dot{\tau}.U)$			
$(LM1)$	$(T+U) \mathbin{\|\!\lfloor}_{S,M} R = T \mathbin{\|\!\lfloor}_{S,M} R + U \mathbin{\|\!\lfloor}_{S,M} R$			
$(LM2)$	$\underline{0} \mathbin{\|\!\lfloor}_{S,M} T = \underline{0}$			
$(LM3)$	$(\gamma.T) \mathbin{\|\!\lfloor}_{S,M} U = \underline{0}$	$type(\gamma) \in S$		
$(LM4)$	$(\dot{\tau}.T) \mathbin{\|\!\lfloor}_{S,M} U = \dot{\tau}.(T \parallel_{S,M} U)$			
$(SM1)$	$T \mid_{S,M} U = U \mid_{S,M} T$			
$(SM2)$	$(T+U) \mid_{S,M} R = T \mid_{S,M} R + U \mid_{S,M} R$			
$(SM3)$	$\underline{0} \mid_{S,M} T = \underline{0}$			
$(SM4)$	$(\gamma.T) \mid_{S,M} (\gamma'.U) = \underline{0}$	$type(\gamma) \notin S \ \vee \ \gamma \neq \gamma'$		
$(SM5)$	$(\dot{\tau}.T) \mid_{S,M} U = T \mid_{S,M} U$			
$(SM6)$	$(\gamma.T) \mid_{S,M} (\gamma.U) = \gamma.(T \parallel_{S,M} U)$	$type(\gamma) \in S$		
$(Hi1)$	$(T+U)/L = T/L + U/L$			
$(Hi2)$	$\underline{0}/L = \underline{0}$			
$(Hi3)$	$(\theta.T)/L = \theta.(T/L)$	$type(\theta) \notin L$		
$(Hi4)$	$(\gamma.T)/L = \dot{\tau}.(T/L)$	$type(\gamma) \in L$		
$(Rec1)$	$recX.T = T\{recX.T/X\}$			
$(Rec2)$	$U = T\{U/X\}$ implies $U = recX.T$	if X is sequential and strongly guarded in T		

Table 3.4: Standard Axioms

$(Act1)$	$a.T$	$=$	$a_1^+.a_1^-.T$	$(Act2)$	$\tau.T$	$=$	$\dot{\tau}.\dot{\tau}.T$
(Par)	$T \parallel_{S,M} U$	$=$	$T \lfloor\!\lfloor_{S,M} U + U \lfloor\!\lfloor_{S,\overline{M}} T + T \mid_{S,M} U$				
$(LM5)$	$(a_i^+.T) \lfloor\!\lfloor_{S,M} U$	$=$	$a^+_{n(M_a)}.(T \parallel_{S,M\cup\{a:(n(M_a),l_i)\}} U)$	$a \notin S$			
$(LM6)$	$(a_i^-.T) \lfloor\!\lfloor_{S,M} U$	$=$	$a_j^-.(T \parallel_{S,M-\{a:(j,l_i)\}} U)$	$a \notin S \wedge (j,l_i) \in M_a$			

Table 3.5: Axioms for the Dynamic Name Technique

$$\frac{T \xrightarrow{a_i^+} T'}{T \lfloor\!\lfloor_{S,M} U \xrightarrow{a^+_{n(M_a)}} T' \parallel_{S,M\cup\{a:(n(M_a),l_i)\}} U}\ a \notin S \qquad \frac{T \xrightarrow{\dot{\tau}} T'}{T \lfloor\!\lfloor_{S,M} U \xrightarrow{\dot{\tau}} T' \parallel_{S,M} U}$$

$$\frac{T \xrightarrow{a_i^-} T' \quad a:(j,l_i) \in M}{T \lfloor\!\lfloor_{S,M} U \xrightarrow{a_j^-} T' \parallel_{S,M-\{a:(j,l_i)\}} U}\ a \notin S$$

$$\frac{T \overset{\gamma}{\Longrightarrow} T' \quad U \overset{\gamma}{\Longrightarrow} U'}{T \mid_{S,M} U \xrightarrow{\gamma} T' \parallel_{S,M} U'}\ \mathit{type}(\gamma) \in S$$

Table 3.6: Rules for the Left and Synchronization Merge Operators

Lemma 3.1 *If a process $P \in BL_{CG}$ is finite state, then $\exists T \in BL_{dn,C}. \mathcal{A}_{BL} \vdash P = T$ with T strongly guarded sequential state.*

Proof: Let $T_1 \ldots T_n$ be the states of the ST semantics of P, $T_n \equiv P$. Since P is strongly guarded, each state T_i of the semantics of P is finitely branching. It can be easily seen that, for each $i \in \{1 \ldots n\}$, there exist $m_i \in \mathbb{N}$, $\{\theta^i_j\}_{j \leq m_i}$, $\{k^i_j\}_{j \leq m_i}$ s.t. $\mathcal{A}_{BL} \vdash T_i = \sum_{j \leq m_i} \theta^i_j.T_{k^i_j}$ where $\sum_\emptyset \equiv \underline{0}$. Hence we can characterize the behavior of P by means of a set of equations similarly to [123]. Moreover, similarly to the unique solution of equations theorem of [123], we have that there is a sequential term T such that $\mathcal{A}_{BL} \vdash T = T_n \equiv P$. This can be shown as follows. For each i, from 1 to n, we do the following. If i is such that $\exists j \leq m_i : k^i_j = i$ we have, by applying $(Rec2)$, that $T_i = recX.(\sum_{j \leq m_i : k^i_j \neq i} \theta^i_j.T_{k^i_j} + \sum_{j \leq m_i : k^i_j = i} \theta^i_j.X)$. Then we replace each subterm T_i occurring in the equations for $T_{i+1} \ldots T_n$ with its equivalent term. When, in the equation for $T_n \equiv P$, we have replaced T_{n-1}, we are done. $\square$

Since for sequential states the ST semantics coincide with the standard interleaving semantics and the axioms of $\mathcal{A}_{BL}$ involved are just the standard axioms for CCS, from [124] and Lemma 3.1 we derive the completeness of $\mathcal{A}_{BL}$.

Theorem 3.7 *$\mathcal{A}_{BL}$ is complete for $\simeq_{dn}$ over finite state processes of BL_{CG}.*

Proof: From [124] we have that given two sequential states T and T' such that $T \approx_{dn} T'$ then $\mathcal{A}_{BL} \vdash T = T'$, hence the result directly follows from Lemma 3.1. $\square$

3.5 Extending the Language with Refinement

The name technique is based on a very intuitive idea but produces a rather complicated representation of states and consequently unnecessarily large semantic models. The reason for this is the intricate structure of mappings M in terms $T \,\|_{S,M}\, U$. The problem is that, for any type a, the association function M_a, may present *holes* and *index permutations*. The exact nature of these two phenomena is explained by the following two examples, the first showing how holes can be generated and the second describing a computation that leads to an index permutation.

Example 3.4 A hole in the ordered sequence of indexes of actions a started by a term $T \,\|_{S,M}\, U$ can be generated as follows. Consider $a.\underline{0} \,\|_{\emptyset}\, a.\underline{0}$. After the righthand a starts, the state is $a.\underline{0} \,\|_{\emptyset,a:\{(1,r_1)\}}\, a_1^-.\underline{0}$. After the lefthand a starts, the state is $a_1^-.\underline{0} \,\|_{\emptyset,a:\{(1,r_1),(2,l_1)\}}\, a_1^-.\underline{0}$. Finally, after the righthand a terminates, the state is $a_1^-.\underline{0} \,\|_{\emptyset,a:\{(2,l_1)\}}\, \underline{0}$ where only index 2 is being used. Therefore we have a hole in position 1. $\blacksquare$

Note that the association function M_a for a given type a can be denoted by a string on the alphabet $Loc \cup \{*\}$, where $*$ denotes a hole in the ordered sequence of indexes of actions a started in $T \,\|_{S,M}\, U$. For example $\{(1,l_1),(3,r_1)\}$ is represented by the string "$l_1 {*} r_1$".

Example 3.5 An index permutation occurs when an association function does not preserve the order of indexes. For instance consider the process $(a.\underline{0} \,\|_{\emptyset}\, a.\underline{0}) \,\|_{\emptyset}\, a.\underline{0}$. After the rightmost a starts, the state is $(a.\underline{0} \,\|_{\emptyset,\emptyset}\, a.\underline{0}) \,\|_{\emptyset,a:r_1}\, a_1^-.\underline{0}$. After the leftmost a starts, the state is $(a_1^-.\underline{0} \,\|_{\emptyset,a:l_1}\, a.\underline{0}) \,\|_{\emptyset,a:r_1 l_1}\, a_1^-.\underline{0}$. After the rightmost a terminates, the state is $(a_1^-.\underline{0} \,\|_{\emptyset,a:l_1}\, a.\underline{0}) \,\|_{\emptyset,a:*l_1}\, \underline{0}$. Finally after the central a starts, the state is $(a_1^-.\underline{0} \,\|_{\emptyset,a:l_1 r_1}\, a_1^-.\underline{0}) \,\|_{\emptyset,a:l_2 l_1}\, \underline{0}$. Therefore we have an index permutation: the action with index 1 is mapped to l_2 and the action with index 2 is mapped to l_1. $\blacksquare$

A different technique which could avoid creating holes in the ordered sequences of indexes and which could guarantee that the order of indexes is preserved by mappings (so that indexes in the strings of the last example would become redundant) would greatly simplify the representation of system states and consequently reduce the size of semantic models.

Moreover, the two phenomena above cause problems when we try to extend our basic language with an action refinement operator $P[a \rightsquigarrow Q]$ which performs the *semantic* refinement [86] of all occurrences of a

in P by Q. This means that every time an action a is executed by P, $P[a \rightsquigarrow Q]$ executes instead a new copy of Q in parallel with P. As suggested also in [52], $P[a \rightsquigarrow Q]$ can be defined in terms of the parallel operator and other basic operators. As we will see, if we want to obtain a refinement operator with the desirable property that if both P and Q are finite state processes then $P[a \rightsquigarrow Q]$ is finite state, then we must have the possibility to define an elimination rule for the parallel operator such that $P \,\|_\emptyset\, Q$ is turned into P if Q is terminated. With the name technique such a rule cannot be implemented in general. For instance $a.\underline{0} \,\|_\emptyset\, a.\underline{0}$ reaches the state $a_1^-.\underline{0} \,\|_{\emptyset, a:\{(2,l_1)\}}\, \underline{0}$ which is not equivalent to $a_1^-.\underline{0}$ as well as $(a.P \,\|_\emptyset\, a.Q) \,\|_\emptyset\, a.\underline{0}$ reaches the state $(a_1^-.P \,\|_{\emptyset, a:l_1 r_1}\, a_1^-.Q) \,\|_{\emptyset, a:l_2 l_1}\, \underline{0}$ which is not equivalent to $a_1^-.P \,\|_{\emptyset, a:l_1 r_1}\, a_1^-.Q$. The point is that a parallel operator may be safely eliminated only if the related mapping is a simple identity, which is rarely the case due to the possible presence of holes or index permutations.

Note that the problem above arises because we consider semantic action refinement instead of, e.g., *syntactic* action refinement (see [86]), where a restricted form of refinement is simply obtained by syntactically replacing terms for actions. But it is not due to the particular implementation of semantic action refinement that we consider in this chapter with respect to, e.g., other implementations of semantic action refinement. This because, according to the notion of semantic action refinement, if actions a of a process P are refined by a term Q, every time a new a is executed by P, a new instance of Q must be put in parallel with P via a static operator. So no matter if this operator is a special operator different from the parallel operator used in the algebra (as, e.g., for the algebras of [52, 73]), since it is static we still have the problem of removing it when the execution of an instance of Q is terminated.

In the rest of the chapter we will consider a language which extends our basic language with an action refinement operator $P[a \rightsquigarrow Q]$, where we distinguish deadlock, denoted by δ, from successful termination, denoted by ε (otherwise ST bisimulation could not be a congruence for the refinement operator) and we employ the ACP [14] sequential composition operator ";" instead of the CCS prefix operator ".".

Let A, *Act* and *Var* be defined as for the basic language *BL* of Sect. 3.2. The terms of *RL* (Refinement Language) are generated by the following syntax:

$$P ::= \varepsilon \mid \delta \mid X \mid \mu \mid P;P \mid P+P \mid P \,\|_S\, P \mid P/L \mid recX.P \mid P[a \rightsquigarrow P]$$

We denote by RL_C the set of *RL* processes, i.e. closed terms of *RL*, and by RL_{CG} the set of strongly guarded processes of *RL*. In the following we will consider a *BL* process as being a process of *RL* by the obvious substitution of term $\mu;P$ for each operator $\mu.P$ and ε for $\underline{0}$.

3.6 ST Semantics via the Stack Technique

In order to solve all the problems we reported in the previous section and to give a satisfactory operational semantics to a language including a semantic action refinement operator, we introduce a completely new technique for representing the ST semantics. This technique is based on the idea of eliminating the holes in the sequences of started action indexes. In particular, started actions of a given type are organized as a stack of coins over a table where the coin on the top of the stack is the action with index 1 and the other actions are indexed in increasing order from top to bottom. When a new action starts the corresponding coin is put on the top of the stack (and the old actions are renumbered accordingly). When an action terminates the corresponding coin is removed and the hole is "eliminated by gravity" (causing a renumbering of all the actions below it).

Since the index of a started action changes dynamically while other actions start and terminate, this technique is not based on names (seen as identifiers for actions) but is more similar to the approach [83, 85, 52] based on pointers. In particular, the event of starting of an action a is represented in semantic models by a transition labeled with a^+ (so no index is observable) whilst the event of termination of an action a is represented by a transition labeled with a_i^- where i is the current position of the action on the stack. The event of action start referred to by a transition a_i^- can be uniquely determined by going back in the history of process computations (reconstructing the history of the stack state at each backward step) until the transition a^+ that pushed on the stack the action that now is at position i is reached. More precisely, the procedure for determining which transition a^+ is pointed by a_i^- is the following.

Let k represent the current position on the stack of the action referred to by a_i^-. Initially we have $k = i$.

- When, going back, we meet an a_j^-, we do the following.
  - If $j \leq k$, then we have to consider an additional action on the stack closer to the top than the one we are referring to (it was removed by a_j^-). Therefore we pose $k = k + 1$ so that the new value of k is the position of our action before the event a_j^-.
  - Otherwise, the additional action does not influence the position of the one we are referring to, so k is unchanged.
- When, going back, we meet an a^+, we do the following.
  - If $k = 1$, then we have reached the transition a^+ that pushed on the stack the action a_i^- and we are done.
  - Otherwise, we have to consider one less action on the stack (it was added by a^+). Therefore we pose $k = k - 1$ so that the new value of k is the position of our action before the event a^+.

This stack-like behavior is expressed compositionally by parameterizing each parallel operator with a mapping M, but now we can rely on association strings which are no longer affected by the two problems of holes and index ordering discussed above. Once again, since the method for updating indexes in the case of an action start or termination is fixed, actions of processes that perform equivalent computations get the same indexes when they terminate and ST bisimilarity can simply be checked by applying standard bisimilarity.

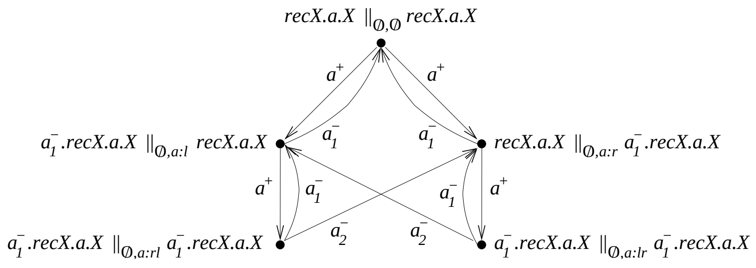


Figure 3.3: Example of Recursion with the Stack Technique

Let w range over the set $IAS = \{w : \mathbb{N}^+ \rightharpoonup Loc \mid \exists k \geq 1.\, dom(w) = \{1 \ldots k\}\}$ of index association strings, i.e. nonempty strings over the alphabet of locations $Loc = \{l, r\}$ [9], ranged over by loc. M includes an association string $w \in IAS$ for each action type a. A string w associated to a type a represents a stack of started a actions, where the action a_1 on the top of the stack corresponds to the leftmost position in w and the action a_i corresponds to the i-th position in w. The location in the i-th position of w determines if the action a_i is executed by the lefthand term (if the location is "l") or righthand term (if the location is "r") of the parallel composition operator. The index j of the action a of the lefthand (or righthand) term associated to a_i is determined as follows. Index j is given by the position of the l (or r) in the string obtained from w by removing all locations r (or l).

Example 3.6 In Fig. 3.3 we depict the ST semantic model of $recX.a.X \,||_\emptyset\, recX.a.X$ obtained by applying the stack technique. [10] By comparing the semantic model of Fig. 3.3 with that of Fig. 3.2 we can see that the phenomenon of holes of the dynamic name technique generates only two additional states. If we consider $(recX.a.X \,||_\emptyset\, recX.a.X) \,||_\emptyset\, recX.a.X$ we have that, due to the combined effects of holes and index permutations, the ST semantic model obtained with the dynamic name technique has 42 states, whilst that obtained with the stack technique has only 16 states. ■

3.6.1 Definition of ST Operational Semantics via the Stack Technique

Now we show how the stack technique can be exploited to give operational ST semantics to our refinement language RL. We need a richer syntax to represent the states of semantic models. We denote by $A_e = A \cup \{e\}$ the set of state observable action types, where e is a distinguished type that will be used in the definition of the refinement operator. Let α range over A_e and $\mathcal{S}, \mathcal{L}$ range over the subsets of A_e. The set of all state action types is denoted by $Act_e = A_e \cup \{\tau\}$, where τ is a distinguished type representing an internal computation. Let $STA_s = \{\alpha^+ \mid \alpha \in A_e\} \cup \{\alpha_i^- \mid \alpha \in A_e \wedge i \in \mathbb{N}^+\}$ (where the s stands for the stack technique), ranged

[9]Even if the set of nonempty strings over Loc is usually denoted by Loc^+ we prefer to stick to this notation to be consistent with the notation of the dynamic name technique.

[10]We apply the stack technique to a simple language with prefixing instead of general sequential composition in the running example only, for ease of comparison.

over by γ; [11] and $STAct_s = STA_s \cup \{\dot{\tau}\}$. θ ranges over $STAct_s$ and η ranges over $Act_e \cup STAct_s$. The meta-variable χ ranges over $STAct_s \cup \{\surd\}$, where $\surd$ is a distinguished action representing successful termination. A mapping M is a relation from A_e to $\mathbb{N}^+ \times Loc$ such that $\forall \alpha \in A_e. M_\alpha \in IAS$, i.e. M is a set including an independent index association string for each different state action type. Similarly to the dynamic name technique we will use $a : (i, loc)$ to stand for $(a, i, loc) \in M$ and we will use the shorthand notation $a : w$, where $w \in IAS$ is an index association string, to stand for the set of triples $\{a : (i, loc) \mid (i, loc) \in w\}$ referring to the same action a. For instance we will use $a : lr$ to stand for $\{a : (1, l), a : (2, r)\}$. Finally, let φ range over the bijections over A_e. The set RL_s of state terms of RL is generated by the following syntax:

$$T ::= \varepsilon \mid \delta \mid X \mid \eta \mid T;T \mid T + T \mid T \parallel_{\mathcal{S},M} T \mid T/\mathcal{L} \mid recX.T \mid T[a \rightsquigarrow T] \mid !T \mid T[\varphi]$$

We denote by $RL_{s,C}$ the set of processes, i.e. closed terms, of RL_s. The bang operator "!" and the (bijective) relabeling operator "$[\varphi]$" are auxiliary operators that are used for the definition of the refinement operator. [12] In the following, in order to avoid ambiguities, we assume the following operator precedence relation: hiding = bang = relabeling > sequential composition > recursion > parallel composition > choice > refinement.

Again we consider the operators "$\parallel_{\mathcal{S}}$" occurring in a RL process P as being "$\parallel_{\mathcal{S},\emptyset}$" when P is regarded as a state.

The semantics of state terms in $RL_{s,C}$ produces a transition system labeled over $STAct_s$. The operational rules for the operators "$+$", "$/\mathcal{L}$" and "$recX$" are those in Table 3.2 where we replace χ for θ and $\mathcal{L}$ for L. The operational rules for "ε", "η" and the operators "$;$" and "$\parallel_{\mathcal{S},M}$" are presented in Table 3.7.

The following auxiliary definitions are used in the definition of the operational semantics of RL. The function $type : STAct_s \cup \{\surd\} \longrightarrow Act_e \cup \{\surd\}$ is defined in the obvious way. We define $M[\alpha \mapsto w]$ with $\alpha \in A_e, w \in IAS$, which modifies a mapping M so that w becomes the new index association string for actions of type α, as follows: $M[\alpha \mapsto w] = \{\alpha' : (i, loc) \in M \mid \alpha' \neq \alpha\} \cup \{\alpha : (i, loc) \mid (i, loc) \in w\}$. The termination predicate $\surd$ is defined as $T\surd \Leftrightarrow (\exists U \in RL_{s,C}.\ T \xrightarrow{\surd} U) \wedge (\nexists \theta \in STAct_s, U' \in RL_{s,C}. T \xrightarrow{\theta} U')$. The expression $\#_{i,l}(w)$ computes the position of the i-th l in the string w. We have that $\#_{i,l}(w)$ is the only $j \in \mathbb{N}^+$ such that $w(j) = l$ and $|\{k \leq j \mid w(k) = l\}| = i$. Similarly for $\#_{i,r}(w)$. Finally we define $w \ominus i$ as the string obtained by removing the i-th element from the string w, i.e. $w \ominus i = \{(j, loc) \in w \mid j < i\} \cup \{(j-1, loc) \mid (j, loc) \in w \wedge j > i\}$. [13]

[11] For simplicity and reuse, we usually denote the meta-variable for a set, which is the version with the stack technique of a set already used in Sect. 3.3 and 3.4 (e.g. STA_s corresponds to STA_{dn} and STA_{sn}), by the same name used in the case of static and dynamic names.

[12] The restriction to bijective relabelings allows us to give a simple operational semantics to the operator "$T[\varphi]$". This because actions with different types cannot be relabeled into actions with the same type, hence it is not necessary to re-index the relabeled actions in order to keep them distinguished. On the other hand the capability of performing bijective relabelings is sufficient for defining the refinement operator.

[13] To be precise both $\#_{i,loc}(w)$ and $w' \ominus j$ should be defined to assume a default value when w does not have an i-th symbol loc or j is greater than the length of w'. Since these situations cannot arise for terms reachable from a term $P \in RL$ or, as we will see, derivable via the axiomatization of weak ST bisimulation from a term $P \in RL$, the actual default value is not important.

$$\alpha \xrightarrow{\alpha^+} \alpha_1^- \qquad \tau \xrightarrow{\dot\tau} \dot\tau \qquad \theta \xrightarrow{\theta} \varepsilon \qquad \varepsilon \xrightarrow{\surd} \delta$$

$$\frac{T \xrightarrow{\alpha^+} T'}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\alpha^+} T' \,\|_{\mathcal{S},M[\alpha\mapsto lM_\alpha]}\, U}\ \alpha\notin\mathcal{S} \qquad \frac{T \xrightarrow{\theta} T'}{T;U \xrightarrow{\theta} T';U}$$

$$\frac{U \xrightarrow{\alpha^+} U'}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\alpha^+} T \,\|_{\mathcal{S},M[\alpha\mapsto rM_\alpha]}\, U}\ \alpha\notin\mathcal{S} \qquad \frac{T \xrightarrow{\surd} T' \quad U \xrightarrow{\chi} U'}{T;U \xrightarrow{\chi} U'}$$

$$\frac{T \xrightarrow{\alpha_i^-} T' \quad \neg(T'\surd \ \wedge\ \mathcal{S}=\emptyset)}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\alpha^-_{\#_{i,l}(M_\alpha)}} T' \,\|_{\mathcal{S},M[\alpha\mapsto M_\alpha\ominus\#_{i,l}(M_\alpha)]}\, U}\ \alpha\notin\mathcal{S} \qquad \frac{T \xrightarrow{\alpha_1^-} T' \quad T'\surd}{T \,\|_{\emptyset,M}\, U \xrightarrow{\alpha^-_{\#_{1,l}(M_\alpha)}} U}$$

$$\frac{U \xrightarrow{\alpha_i^-} U' \quad \neg(U'\surd \ \wedge\ \mathcal{S}=\emptyset)}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\alpha^-_{\#_{i,r}(M_\alpha)}} T \,\|_{\mathcal{S},M[\alpha\mapsto M_\alpha\ominus\#_{i,r}(M_\alpha)]}\, U'}\ \alpha\notin\mathcal{S} \qquad \frac{U \xrightarrow{\alpha_1^-} U' \quad U'\surd}{T \,\|_{\emptyset,M}\, U \xrightarrow{\alpha^-_{\#_{1,r}(M_\alpha)}} T}$$

$$\frac{T \xrightarrow{\dot\tau} T' \quad \neg(T'\surd \ \wedge\ \mathcal{S}=\emptyset)}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\dot\tau} T' \,\|_{\mathcal{S},M}\, U} \qquad \frac{T \xrightarrow{\dot\tau} T' \quad T'\surd}{T \,\|_{\emptyset,M}\, U \xrightarrow{\dot\tau} U}$$

$$\frac{U \xrightarrow{\dot\tau} U' \quad \neg(U'\surd \ \wedge\ \mathcal{S}=\emptyset)}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\dot\tau} T \,\|_{\mathcal{S},M}\, U'} \qquad \frac{U \xrightarrow{\dot\tau} U' \quad U'\surd}{T \,\|_{\emptyset,M}\, U \xrightarrow{\dot\tau} T}$$

$$\frac{T \xrightarrow{\chi} T' \quad U \xrightarrow{\chi} U'}{T \,\|_{\mathcal{S},M}\, U \xrightarrow{\chi} T' \,\|_{\mathcal{S},M}\, U'} \quad \mathit{type}(\chi) \in \mathcal{S} \cup \{\surd\}$$

Table 3.7: Rules for the Stack Technique

Let us comment on the operational rules for $T \,\|_{\mathcal{S},M}\, U$.

When T performs α^+ ($\alpha \notin \mathcal{S}$), then the new action is pushed on the top of the stack of α actions. This is represented by putting an l in the first position of the association string for α. Symmetrically for a move α^+ of U.

When T performs α_i^- ($\alpha \notin \mathcal{S}$), the corresponding α action (whose position on the stack is that of the i-th l in the association string for α) terminates and is eliminated from the stack. This behavior is expressed by two rules in Table 3.7 because we eliminate the parallel operator in the case T becomes a successfully terminated process. Note that this can happen only if, after the termination of α_i^-, there are no more action to terminate, hence it is just sufficient to consider $i = 1$ in the elimination rule. Symmetrically for a move α_i^- of U.

$$\frac{(\, T[a \leftrightarrow e] \;\|_{\{e\},\emptyset}\; !(e^+; U; e_1^-)\,)/\{e\} \xrightarrow{\;\chi\;} T'}{T[a \rightsquigarrow U] \xrightarrow{\;\chi\;} T'}$$

$$\frac{T \xrightarrow{\;\alpha^+\;} T'}{T[\varphi] \xrightarrow{\;\varphi(\alpha)^+\;} T'[\varphi]} \qquad \frac{T \xrightarrow{\;\alpha_i^-\;} T'}{T[\varphi] \xrightarrow{\;\varphi(\alpha)_i^-\;} T'[\varphi]}$$

$$\frac{T \xrightarrow{\;\chi\;} T'}{T[\varphi] \xrightarrow{\;\chi\;} T'[\varphi]} \quad \mathit{type}(\chi) \in \{\tau, \surd\}$$

$$!T \xrightarrow{\;\surd\;} \delta$$

$$\frac{T \xrightarrow{\;\alpha^+\;} T'}{!T \xrightarrow{\;\alpha^+\;} T' \,\|_{\emptyset,\alpha:l} !T} \qquad \frac{T \xrightarrow{\;\dot\tau\;} T'}{!T \xrightarrow{\;\dot\tau\;} T' \,\|_{\emptyset,\emptyset} !T}$$

Table 3.8: Rules for Refinement

The semantic rules for the refinement operator are based on its definition in terms of the parallel operator and other basic operators. Our approach to ST semantics enables the following definition of $T[a \rightsquigarrow U]$ that closely adheres to the intuition of the way it works:

$$(T[a \leftrightarrow e] \,\|_{\{e\},\emptyset} !(e^+; U; e_1^-))/\{e\}$$

where the bijective relabeling $\alpha \leftrightarrow \alpha'$ is defined by $\alpha \leftrightarrow \alpha' = \{(\alpha, \alpha'), (\alpha', \alpha)\} \cup \{(\alpha'', \alpha'') \mid \alpha'' \in A_e \wedge \alpha'' \notin \{\alpha, \alpha'\}\}$. For each a executed by the process T a corresponding process U is activated by the bang operator in the righthand term. In this way if T executes several auto-concurrent actions a then a corresponding number of processes U are executed in parallel by the righthand term. The correct association between actions a and processes U is guaranteed by the fact that the events of starting and termination of each auto-concurrent action e are uniquely related by the ST semantics. [14]

The operational rules for the refinement operator, "!" and "$[\varphi]$" are presented in Table 3.8.

The operational rules for $!T$ just say that we can execute zero or more parallel copies of T. The number of copies depends on the initial choice performed by $!T$ between termination (action $\surd$) and execution of T in parallel with a new instance of $!T$. Such new instance is generated only after the execution of the first action of T. This avoids the immediate generation of an unbounded number of instances of $!T$ which would lead to infinite state models. Note that, in the definition of the refinement operator, this interpretation of

[14]This definition of semantic action refinement is slightly different from the usual definition [78, 73, 86] in that in $T[a \rightsquigarrow U]$ each execution of U is preceded and followed by the occurrence of a silent transition $\dot\tau$. In order to obtain a definition which adheres completely to the usual one it is simply sufficient to "skip" e transitions (instead of just hiding them with "$/\{e\}$"), similarly as done in [52].

the bang operator leads to the execution of the correct number of instances of its argument, because we exploit the fact that a process in parallel with another process may actually perform the terminating action $\surd$ only when both processes are ready to terminate (actions $\surd$ of parallel processes must synchronize).

Theorem 3.8 *The interleaving semantics of a RL process P not including the refinement operator is finite state if and only if the ST semantics of P obtained with the stack technique is finite state.*

Proof:

Proof of "$\Longrightarrow$" Similarly to the proof of Theorems 3.1 and 3.4 we consider the finite state term P' obtained by syntactically substituting $\mu;\mu$ for each action μ occurring in the term P. If, inside the state terms of the ST semantics of P, we syntactically replace each ST action with its type and we discard the sets M in the operators $\|_{S,M}$, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because, since holes in the ordered sequence of started action indices generated by action termination are immediately eliminated by the stack policy, the maximum index an action may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.

Proof of "$\Longleftarrow$" The same as in the proof of Theorem 3.1. $\square$

Our approach ensures that the finiteness of semantic models is preserved by the action refinement operator.

Theorem 3.9 *If P and Q are finite state RL processes, then $P[a \rightsquigarrow Q]$ is a finite state process.*

Proof: The fact that $P[a \rightsquigarrow Q]$ is finite state easily derives from the presence of the elimination rules for the parallel operator and from the fact that the maximum number of self-concurrent actions a in the P process is finite (hence the maximum number of parallel processes opened by the bang operator in a certain state is finite) because P is finite state. $\square$

Example 3.7 In Fig. 3.4 we present the finite ST semantic model of $recX.a;X[a \rightsquigarrow b;c]$ obtained with the stack technique. In Fig. 3.5 we show an initial fragment of the infinite ST semantic model of the same term obtained with the dynamic name technique. Note that ST semantics via the dynamic name technique over the whole language *RL* is simply obtained as follows. The operational rules for the refinement, relabeling and bang operators are the same as for the stack technique, except that e^+ is replaced by e_1^+ in the premise for refinement, α^+ is replaced by α_i^+ in the rules for relabeling and bang operators and $\varphi(\alpha)^+$ is replaced by $\varphi(\alpha)_i^+$ in the rule for relabeling. Fig. 3.5 makes clear that, in the absence of an elimination rule for the parallel operator, the number of parallel operators generated by the bang operator grows as new actions to

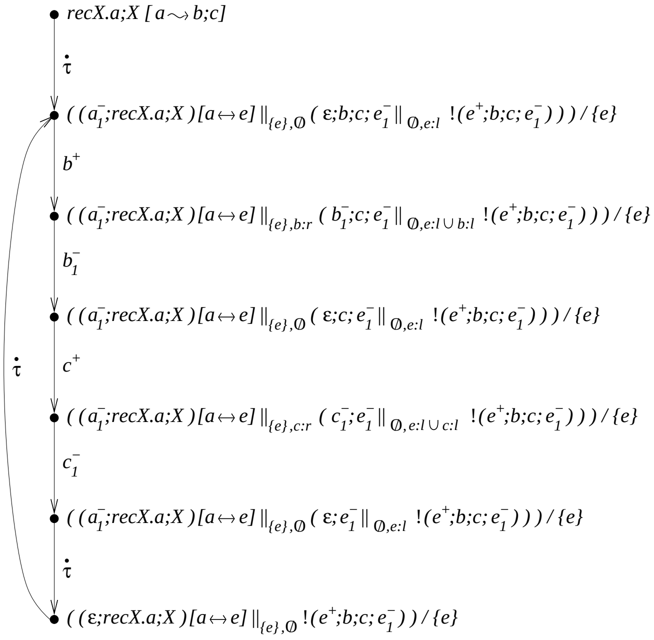


Figure 3.4: Example of Refinement with the Stack Technique

be refined start and terminate. Therefore even refining a simple recursive term such as $recX.a;X$ leads to an infinite semantic model. [15] ■

Now we present a simple syntactic criterion which constitutes a sufficient condition for a process of RL_C to be finite state. In the following corollary we call "static" the operators of parallel composition, hiding and action refinement only.

Corollary 3.3 *Let P be a process of RL_C s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator or in the lefthand side of a ";". Then P is a finite state process.*

3.6.2 Weak ST Bisimulation via the Stack Technique

The equivalence notion we consider over terms of RL_s, denoted by $\simeq_s$, is again observational congruence extended to open terms [122], where the alphabet of visible actions is STA_s and hidden actions are $\dot{\tau}$ actions.

We have the following theorem of consistency with the static name ST semantics.

Theorem 3.10 *Given two processes P and Q of BL_C we have that $P \simeq_s Q$ if and only if $P \simeq_{sn} Q$.*

[15] For the simple example $recX.a;X[a \rightsquigarrow b;c]$, the execution of a refinement by means of $!(e_1^+;b;c;e_1^-)$ always leads to $\varepsilon \,||_{\emptyset,\emptyset}!(e_1^+;b;c;e_1^-)$, where the parallel operator could in fact be eliminated. The fact that, with the name technique, we cannot apply the elimination rule to the parallel operators generated by refinement, can be seen by considering a refinement (e.g. $a \rightsquigarrow b;c$) of the term $recX.a;X \,||\, recX.a;X$, whose semantic model (see Fig. 3.2) includes states which exhibit "holes" in the index sequences of started actions.

$$
\begin{array}{l}
\bullet\ recX.a;X\,[a \rightsquigarrow b;c] \\
\big\downarrow \dot{\tau} \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},\emptyset} (\varepsilon;b;c;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow b_1^+ \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},b:\{(1,r_1)\}} (b_1^-;c;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\} \cup b:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow b_1^- \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},\emptyset} (\varepsilon;c;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow c_1^+ \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},c:\{(1,r_1)\}} (c_1^-;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\} \cup c:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow c_1^- \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},\emptyset} (\varepsilon;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow \dot{\tau} \\
\bullet\ ((\varepsilon;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},\emptyset} (\varepsilon \,\|_{\emptyset,\emptyset}\ !(e_1^+;b;c;e_1^-)))/\{e\} \\
\big\downarrow \dot{\tau} \\
\bullet\ ((a_1^-;recX.a;X)[a \leftrightarrow e] \,\|_{\{e\},\emptyset} (\varepsilon \,\|_{\emptyset,\, e:\{(1,l_1)\}} (\varepsilon;b;c;e_1^- \,\|_{\emptyset,\, e:\{(1,l_1)\}}\ !(e_1^+;b;c;e_1^-))))/\{e\} \\
\vdots\ b_1^+ \\
\downarrow
\end{array}
$$

Figure 3.5: Example of Refinement with the Name Technique

Proof: See Appendix C. □

From Theorems 3.10 and 3.5 we immediately derive also the equivalence between the stack technique and the dynamic name technique.

Corollary 3.4 *Given two processes P and Q of BL_C we have that $P \simeq_s Q$ if and only if $P \simeq_{dn} Q$.*

Moreover from Theorems 3.10 and 3.2 it easily derives that $\simeq_s$ is preserved by all the operators of RL, including recursion.

Corollary 3.5 *$\simeq_s$ is a congruence for RL.*

Proof: The congruence w.r.t. the refinement operator derives from the congruence w.r.t. relabeling and bang operators. The congruence w.r.t. the bang operator derives from congruence w.r.t basic operators since it is easy to see that the semantic model of $!P$ is isomorphic to that of $recX.(\varepsilon + P \,\|_{\emptyset,\emptyset} X)$ where X does not occur free in P. □

Theorem 3.11 *Observational ST congruence is decidable over processes of RL_C which possess a finite state stack technique ST semantics.*

$(Act1)$	$\alpha = \alpha^+; \alpha_1^-$	$(Act2)$	$\tau = \dot{\tau}; \dot{\tau}$
$(Seq1)$	$(T;U);R = T;(U;R)$	$(Seq2)$	$(T+U);R = T;R + U;R$
$(Seq3)$	$\varepsilon;T = T$	$(Seq4)$	$T;\varepsilon = T$
$(Seq5)$	$\delta;T = \delta$		
(Par)	$T \parallel_{\mathcal{S},M} U = T \lfloor\!\lfloor_{\mathcal{S},M} U + U \lfloor\!\lfloor_{\mathcal{S},\overline{M}} T + T \mid_{\mathcal{S},M} U$		
$(LM5)$	$(\theta;T) \lfloor\!\lfloor_{\emptyset,M} \varepsilon = \theta;T$		
$(LM6)$	$\varepsilon \lfloor\!\lfloor_{\mathcal{S},M} T = \delta$		
$(LM7)$	$(\alpha^+;T) \lfloor\!\lfloor_{\mathcal{S},M} U = \alpha^+; (T \parallel_{\mathcal{S},M[a \mapsto l M_\alpha]} U)$	$\alpha \notin \mathcal{S}$	
$(LM8)$	$(\alpha_i^-;T) \lfloor\!\lfloor_{\mathcal{S},M} U = \alpha^-_{l(w,i)}; (T \parallel_{\mathcal{S},M[a \mapsto M_\alpha \ominus \#_{i,l}(M_\alpha)]} U)$	$\alpha \notin \mathcal{S}$	
$(SM7)$	$\varepsilon \mid_{\mathcal{S},M} \varepsilon = \varepsilon$	$(SM8)$	$\varepsilon \mid_{\mathcal{S},M} (\theta;T) = \delta$
$(Hi5)$	$\varepsilon/\mathcal{L} = \varepsilon$		

Table 3.9: Axioms for the Stack Technique

(Ref)	$T[a \rightsquigarrow U] = (\, T[a \leftrightarrow e] \parallel_{\{e\},\emptyset} !(e^+;U;e_1^-)\,)/\{e\}$		
$(Rel1)$	$(T+U)[\varphi] = T[\varphi] + U[\varphi]$	$(Rel2)$	$(\alpha^+;T)[\varphi] = \varphi(\alpha)^+;(T[\varphi])$
$(Rel3)$	$(\alpha_i^-;T)[\varphi] = \varphi(\alpha)_i^-;(T[\varphi])$	$(Rel4)$	$(\dot{\tau};T)[\varphi] = \dot{\tau};(T[\varphi])$
$(Rel5)$	$\varepsilon[\varphi] = \varepsilon$	$(Rel6)$	$\delta[\varphi] = \delta$
$(Bang)$	$!T = recX.(\varepsilon + T \lfloor\!\lfloor_{\emptyset,\emptyset} X)$ provided that X is not free in T		

Table 3.10: Axioms for Refinement

Proof: An immediate consequence of the fact that we rely on standard observational congruence. □

3.6.3 Axiomatization of Weak ST Bisimulation via the Stack Technique

Now we will show that with the stack technique it is easy to define an axiomatization for $\simeq_s$ which is complete over finite state processes of RL_{CG} by following an approach similar to the axiomatization of standard bisimulation for process algebras with a structural operational interleaving semantics.

The axiom system $\mathcal{A}_{RL}$ for $\simeq_s$ on RL_s terms is formed by: the standard axioms presented in Table 3.4 where we replace the operator ";" for the operator ".", [16] "δ" for "$\underline{0}$", $\mathcal{L}$ for L and $\mathcal{S}$ for S; the axioms of Table 3.9 which are specific for the stack technique; and the axioms of Table 3.10 which deal with the refinement operator.

[16] With the operator ";" we can write the axiom $(Tau1)$ simply as $\theta;\dot{\tau} = \theta$. Moreover we could adopt a more elegant formulation of the axioms for the hiding operator where in $(T;U)/L$ we distribute hiding over the arguments of ";", i.e. $(T;U)/L = (T/L);(U/L)$.

$$\frac{T \xrightarrow{\alpha^+} T'}{T \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} U \xrightarrow{\alpha^+} T' \parallel_{\mathcal{S},M[\alpha \mapsto l M_\alpha]} U} \quad \alpha \notin \mathcal{S}$$

$$\frac{T \xrightarrow{\alpha_i^-} T' \quad \neg(T'\surd \,\wedge\, \mathcal{S} = \emptyset)}{T \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} U \xrightarrow{\alpha^-_{\#_{i,l}(M_\alpha)}} T' \parallel_{\mathcal{S},M[\alpha \mapsto M_\alpha \ominus \#_{i,l}(M_\alpha)]} U} \quad \alpha \notin \mathcal{S} \qquad \frac{T \xrightarrow{\alpha_1^-} T' \quad T'\surd}{T \mathbin{\lfloor\!\lfloor}_{\emptyset,M} U \xrightarrow{\alpha^-_{\#_{1,l}(M_\alpha)}} U}$$

$$\frac{T \xrightarrow{\dot{\tau}} T' \quad \neg(T'\surd \,\wedge\, \mathcal{S} = \emptyset)}{T \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} U \xrightarrow{\dot{\tau}} T' \parallel_{\mathcal{S},M} U} \qquad \frac{T \xrightarrow{\dot{\tau}} T' \quad T'\surd}{T \mathbin{\lfloor\!\lfloor}_{\emptyset,M} U \xrightarrow{\dot{\tau}} U}$$

$$\frac{T \xRightarrow{\chi} T' \quad U \xRightarrow{\chi} U'}{T \,|_{\mathcal{S},M}\, U \xrightarrow{\chi} T' \parallel_{\mathcal{S},M} U'} \quad \mathit{type}(\chi) \in \mathcal{S} \cup \{\surd\}$$

Table 3.11: Rules for the Left and Synchronization Merge Operators

The axiom (Par) is the standard one except that when the position of processes T and U is exchanged we must invert left and right inside M. More precisely, the inverse $\overline{M}$ of a mapping M is obtained, similarly to the dynamic name technique, by inverting locations in index association strings as follows: $\overline{M} = \{a : (i,r) \mid a : (i,l) \in M\} \cup \{a : (i,l) \mid a : (i,r) \in M\}$. The axiom $(LM5)$ reflects the elimination rules for the parallel operator. Finally, the axioms $(LM7)$ and $(LM8)$ just reflect the operational rules of the parallel operator for an independent move of the lefthand process.

In Table 3.11 we present the operational rules for "$\lfloor\!\lfloor_{\mathcal{S},M}$" and "$|_{\mathcal{S},M}$", which are obtained from the rules of the parallel operator by following an approach similar to that of [1].

Theorem 3.12 *The axioms of $\mathcal{A}_{RL}$ are sound for $\simeq_s$ over RL_s terms.*

Proof: A simple consequence of the soundness of the axiomatization of [1] and of the fact that the semantic model of $!P$ is isomorphic to that of $\mathit{rec}X.(\varepsilon + P \mathbin{\lfloor\!\lfloor}_{\emptyset,\emptyset} X)$ where X does not occur free in P. □

Once we have defined a sequential state to be a state that can be built using only ε, δ, X and operators $\theta; T$, $T' + T''$, $\mathit{rec}X.T$, and assumed the standard definition of [122] for strong guardedness of sequential states, we have the following theorem.

Lemma 3.2 *If a process $P \in RL_{CG}$ is finite state then $\exists T \in RL_{s,C}. \mathcal{A}_{RL} \vdash P = T$ with T strongly guarded sequential state.*

Proof: The proof of this theorem is completely analogous to that of Lemma 3.1, in particular ";" has the same role as "." and "δ" has the same role as "$\underline{0}$". The only difference is that a term "ε" must be added to the summation in the equation for a state T_i if T_i has an outgoing transition labeled with "$\surd$". □

Since for sequential states the ST semantics coincide with the standard interleaving semantics and the axioms of $\mathcal{A}_{RL}$ involved are just the standard axioms for CCS (it suffices to consider "$\eta; T$" as being "$\eta.T$", "δ" as being "$\underline{0}$" and "ε" as being "$\surd.\underline{0}$"), from [124] and Lemma 3.2 we derive the completeness of $\mathcal{A}_{RL}$.

Theorem 3.13 *$\mathcal{A}_{RL}$ is complete for $\simeq_s$ over finite state processes of RL_{CG}.*

Proof: From [124] we have that given two sequential states T and T' such that $T \approx_s T'$ then $\mathcal{A}_{RL} \vdash T = T'$, hence the result directly follows from Lemma 3.2. □

3.7 Comparison of the Three Techniques

In this section we will compare the size of the ST semantic models produced with our three techniques for processes of the Basic Language.

The static name technique produces much smaller semantic models with respect to the other two techniques intuitively because it does not produce different representations for states reached by starting the same set of actions (of the same type) in different orders: the same state is reached no matter which order of action execution is chosen. This because, differently from the dynamic name and stack techniques, when we establish weak ST bisimulation of processes through the static name technique, "all the work" is done by the equivalence which associates static names.

Theorem 3.14 *Given a term $P \in BL_C$, the number of states of the semantic model of P produced with the static name technique is less or equal to the number of states of the semantic model of P produced with the dynamic name and stack techniques. Moreover if the behavior of P is such that it may execute at least two concurrent actions of the same type, then the number of states of the semantic model of P produced with the static name technique is strictly smaller than the number of states of the semantic model of P produced with the dynamic name and stack techniques.* ■

Proof: It is easy to see that, in both the dynamic name and the stack technique ST semantic models of P, the number of states corresponding to the same state $\langle T, ex\rangle$ of the ST static name semantic model of P (i.e. states U such that $sn(U) = \langle T, ex\rangle$ according to Definitions A.8 and A.14) is at least $\sum_{a \in A : ex_a \neq \emptyset}(|ex_a|!)$, i.e. for each action type we consider the factorial of the number of the action of that type in execution in the state. This because, since the actions in ex are in parallel execution in $\langle T, ex\rangle$, they are independent (not causally related), hence they can be started in any order. Therefore both the dynamic name and the stack technique ST semantic models of P include at least a different state U such that $sn(U) = \langle T, ex\rangle$ for each possible permutation of the actions in ex of the same type. Formally, if for each action type a such that $ex_a \neq \emptyset$ we consider a bijection $Perm_a$ from $\{1 \ldots |ex_a|\}$ to ex_a, then both in the dynamic name and the stack technique ST semantic models of P there exists a state U such that, for each action type a considered above, we have (according to Definitions A.6 and A.12) $dsmap(U)_a = Perm_a$ for the dynamic name technique and $ssmap(U)_a = Perm_a$ for the stack technique. Finally note that states of the dynamic name and the stack technique semantic models of P corresponding to different states of the static name semantic model of P are obviously different, otherwise sn would not be a function. □

The stack technique produces smaller semantic models with respect to the dynamic name technique intuitively because it avoids creating holes in the ordered sequences of indexes of active actions of the same type (and consequently also index permutations, see Sect. 5).

Theorem 3.15 *Given a term $P \in BL_C$, the number of states of the semantic model of P produced with the stack technique is less or equal to the number of states of the semantic model of P produced with the dynamic name technique. Moreover if the behavior of P is such that it may execute at least two concurrent actions of the same type, then the number of states of the semantic model of P produced with the stack technique is strictly smaller than the number of states of the semantic model of P produced with the dynamic name technique.* ■

Proof: As we showed in the proof of Theorem 3.14 the number of states of the dynamic name semantic model of P in correspondence with a given state $\langle T, ex \rangle$ of the static name semantic model of P is at least $\sum_{a \in A : ex_a \neq \emptyset} (|ex_a|!)$. It is easy to see that, since in the stack technique the actions in execution of the same type are always recorded in a stack whose dimension is equal to the number of such actions, the number of states of the stack technique semantic model of P in correspondence with $\langle T, ex \rangle$ is at most $\sum_{a \in A : ex_a \neq \emptyset} (|ex_a|!)$, i.e. the number of possible configurations (permutations of actions of the same type) for all the stacks. Formally, this derives from the fact that in each state U of the stack technique semantic model of a process $P \in BL_C$, the indexes (representing stack positions) associated with the actions in execution of a given type in the state always range from 1 to k for some k, i.e. for each action type a we have $dom(ssmap(U)_a) = \{1 \ldots k_a\}$ according to Definition A.12. Moreover it is easy to see that for each ordering of the actions of ex in the stacks, there is at most one state U of the stack technique semantic model of P in correspondence with $\langle T, ex \rangle$ implementing that ordering, i.e. such that $ssmap(U)_a$ yields the desired ordering for the actions of ex of type a. Finally, if the behavior of P is such that it may execute at least two concurrent actions of the same type, then in the dynamic name semantics of P there is a state U such that we have two actions in execution, both with the same type a, with associated indexes 1 and 2. Moreover U must have an outgoing transition representing the termination of the action a_1 leading to a state U' where we have a "hole" in position 1. Therefore, considered the state $sn(U')$ of the static name semantics of P corresponding to U' (see Definition A.8) which has a single action in execution, we have that in the stack technique semantics of P there is a single state corresponding to $sn(U')$, while in the dynamic name semantics of P there are at least two states corresponding to $sn(U')$ (U' and the state U'' such that $sn(U'') = sn(U')$ and the action a in execution has index 1). □

3.8 Conclusion

Even if the stack technique is more adequate in the context of ST semantics because it allows to decide and axiomatize ST bisimulation also in the presence of an action refinement operator, we believe that the three techniques for expressing ST semantics that we have presented have different features that may make one of them more suitable than another one depending on the context of application. We have introduced the static name technique first of all because it allows us to use the equivalence of [4] which has been proven in [84] to coincide to the original definition of ST equivalence [78]. In this way we have that all three equivalences considered in this chapter actually coincide with ST bisimulation. Secondly because it is a technique producing much smaller semantic models with respect to the other two techniques. Moreover we have introduced the dynamic name technique for the following reasons. First of all, it shows how dynamic names, which have been previously used in the literature for producing finite models in the presence of recursion, can be managed compositionally, thus giving rise to structural operational semantics for terms of process algebras and axiomatizations. Secondly, the dynamic name technique is more intuitive than the stack technique because it identifies an action with the same name throughout all its execution and makes it possible to understand the (quite complex) notion of levelwise renaming, without getting at the same time into the intricate behavior of the stack technique. Thirdly, for certain problems it may constitute the unique solution, i.e. the stack technique may not be applicable. For example, as we will see in the rest of the thesis, when ST semantics is used to express execution of activities in timed systems, the models used in the literature (e.g., timed automata or generalized semi-Markov processes for probabilistic time) require an activity to maintain its name throughout all its execution (name of the clock representing the activity) so the stack technique cannot be applied to derive models of this kind from an algebraic specification. Moreover, even if the dynamic name technique is not practically applicable to a language including an action refinement operator (semantic action refinement [86]), we could still rely on syntactic action refinement [86], i.e. we could perform a limited form of action refinement by syntactically replacing terms for actions.

The finiteness result for ST semantics and refinement makes the top-down design of concurrent systems actually performable. This result also finds application in other areas more related to the topic of this thesis. In the context of probabilistically timed process algebras where time is expressed by associating a general probability distribution of duration to actions, we have developed two languages, one based on ST semantics via static names, which will be presented in Chapter 13, and another one based on ST semantics via dynamic names, which will be presented in Chapter 7. The use of the ST semantics via static and dynamic names (as opposed to the use of the stack technique) gives rise to semantic models which are very close to Generalized Semi-Markov Processes (GSMPs) which can be analyzed to evaluate performance measures of systems. In particular names assigned to actions by ST semantics become the names of the elements of the GSMP deriving from an algebraic specification. Since in such process algebras the mathematical evaluation of performance measures of systems rely on the analysis of the stationary behavior of cyclic systems, it is crucial that we can obtain finite state models also in the presence of recursion. Moreover, in this context the finiteness result for action refinement gives the new possibility of “refining”, phase-type distributions (a special class of duration probability distributions which are very useful in many applications and can be used to approximate every kind of duration distribution) into combinations of exponential distributions that are easily analyzable through well-known mathematical techniques (i.e. analysis of Continuous Time Markov Chains) for deriving performance measures of systems. In Chapters 7 and 8 we will use syntactic action refinement in conjuction with the dynamic name technique to turn systems using activities with a phase-type distributed duration into purely Markovian systems.

Finally, we would like to note that, if we just consider the process algebras commonly used in the literature which can be endowed with an interleaving semantics (and we do not consider ad-hoc operators which exploit the fact of splitting actions through ST semantics), our results are not language dependent. For example, considering ACP [14] instead of CSP parallel composition would only lead to more complex mappings (parameterizing the parallel operators) for representing the associations between the computations of $P \,\|\, Q$ and the computations of P and Q that originate them. This because in ACP an action of type a of $P \,\|\, Q$ can be originated from the synchronization of two actions of P and Q with a different type, e.g. b and c. Therefore we have to parameterize parallel operators with mappings associating (indexed) actions of different types. Nevertheless, apart from the implementation details of levelwise renaming, the static name, dynamic name and stack technique can still be used to finitely represent the behavior of systems. In particular concerning the dynamic name and stack technique, the fact that we obtain a finite ST semantic model for terms having a finite interleaving semantic model is not at all related to the operators used in the term, but is due to the way we generate names for action started at the top level by the term and reuse names of actions terminated at the top level by the term.

Part II

Interactive Timed Processes

Chapter 4

Interactive Weighted Markov Chains

In this chapter we introduce Interactive Weighted Markov Chains (IWMCs), which are basically an extension of continuous time Markov chains (see Sect. 2.2.2) with *action transitions*, representing the ability of the process to interact with other processes, and *probabilistic transitions*, representing probabilistic choices internally performed by the process. In particular interactive weighted Markov chains extend interactive Markov chains of [99] with the capability of representing probabilistic choices through transitions labeled with *weights* [150]. Besides introducing the model of interactive weighted Markov chains, we will introduce a calculus generating IWMCs as semantic models of terms (called calculus of interactive weighted Markov chains) and a notion of observational congruence for IWMCs which is a congruence for all the operators of the calculus. Finally we will present an axiomatization for such an equivalence which is complete for finite-state strongly guarded terms of the calculus. We will address the problem of axiomatizing weakly guarded (and fully unguarded) terms in Chapter 9.

4.1 Introduction

The advantages of using process algebras for the performance modeling and evaluation of concurrent systems due to their feature of compositionality have been widely recognized [107, 87, 7, 19, 49, 90, 139, 69, 99, 113]. Particularly simple and successful has been the extension of standard process algebras with time delays whose duration follows an exponential probability distribution, called Markovian process algebras [107, 87, 19, 138, 99]. As we will see, the advantage of restricting to this kind of delays stems from the *memoryless property* of exponential distributions. This property basically says that in each time point in which an exponential delay has started but not terminated yet, the residual duration of the delay is still distributed as the entire duration of the delay. The "timed" behavior of systems specified with a Markovian process algebra can represented by a continuous time Markov chain (CTMC), i.e. a simple continuous time stochastic process where in each time point the future behavior of the process is completely independent on its past behavior (see Chapter 2). Due to their simplicity CTMCs can be analyzed with standard mathematical techniques and software tools (see e.g. [147]) for deriving performance measures of systems.

Let us recall some basic concepts about CTMCs. Transitions leaving a state of a CTMC represent exponential delays which are thought of as being in a race: the fastest one that terminates determines the transition to be executed. More precisely, given a state s with n outgoing transitions labeled with rates $\lambda_1, \ldots, \lambda_n$, respectively, we have that:

- The average sojourn time in s is exponentially distributed with rate $\sum_{i=1}^{n} \lambda_i$.
- The probability of executing the k-th outgoing transition of s is $\lambda_k / \sum_{i=1}^{n} \lambda_i$.

CTMCs can be easily extended with states which have a zero sojourn time (the so-called *vanishing* states) and outgoing probabilistic transitions (labeled, e.g., with a *weight* [150]) representing probabilistic choices, similarly as done, e.g., in [141]. More precisely, given a state s with n outgoing transitions labeled with weights $w_1, \ldots, w_n$, respectively, we have that:

- The sojourn time in s is zero.
- The probability of executing the k-th outgoing transition of s is $w_k / \sum_{i=1}^{n} w_i$.

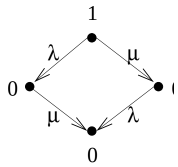
In the following we will call *Weighted Markov Chains (WMCs)* this extension of CTMCs with weighted transitions. Extending CTMCs in this way is very convenient in that it significantly simplifies the task of modeling real systems (in that alternative system behaviors can be expressed via probabilistic choices) without increasing the "complexity" of the underlying class of stochastic processes. This because models with both *tangible* states (the ordinary states of CTMCs) and vanishing states can be easily turned into CTMCs via a simple procedure which eliminates vanishing states. In particular the elimination of a vanishing state simply causes each of its ingoing transitions to be split into a set of transitions corresponding to the probabilistic transitions leaving the vanishing state (see [19] Chapter 4 for a precise definition of this procedure).

In [99] specifying concurrent systems as the parallel composition of interacting subsystems described by CTMCs is made possible simply by extending CTMCs with standard action transitions, thus giving rise to Interactive Markov Chains (IMCs). Following this idea we introduce interaction in WMCs by developing Interactive Weighted Markov Chains (IWMCs), which are quite similar to the model considered in [141]. An IWMC represents the behavior of a component by employing both *standard action transitions*, representing the interactive behavior of the component, and *probabilistic transitions* and *exponentially timed transitions*, representing the timed probabilistic behavior of the component. Action transitions are just standard CCS/CSP transitions which are executed in zero time: when several action transitions are enabled in an IWMC state, the choice among them is just performed non-deterministically and when IWMCs are composed in parallel they synchronize following the CSP [109] approach, where the actions belonging to a given set S are required to synchronize. Probabilistic transitions are labeled with a weight and represent probabilistic choices performed proportionally to the weights. Finally, exponentially timed transitions are labeled with a rate and represent timed choices performed according to a race between exponential delays, as already explained. Similarly as in [99], the interrelation between standard action transitions and performance related transitions (probabilistic and exponentially timed transitions) is governed by the so-called *maximal progress assumption* [133]: the possibility of executing τ transitions prevents the execution performance related transitions, thus expressing that the system cannot wait if it has something internal to do. But differently from [99], where such a priority is captured in the definition of equivalence among IMCs, we prefer to express priority by cutting transitions which cannot be performed when defining and composing IWMCs (a solution also hinted in [100]). This allows us to obtain smaller system models and to define a notion of bisimulation among IWMCs more simply, without having to discard any transitions when establishing equivalence.

The notion of weak bisimulation for IWMCs that we present matches exponentially timed transitions according to Markovian bisimulation [107], deals with probabilistic choices according to probabilistic bisimulation [117], and abstracts from standard τ actions as in [122]. In particular, with respect to [99], we do not complicate the notion of equivalence by including conditions about the "stability" of bisimilar states, in that, for the time being, we will focus in obtaining a congruence result only for IWMCs which do not include τ loops. An IWMC that is forced in a τ loop can be seen as a "Zeno process", i.e. a system which performs infinite computations without going beyond a certain point in time. Discarding IWMCs with τ loops allows us to avoid the technical complications deriving from the treatment of Zenoness (see e.g. [105, 99]) and on the other hand seems not to be so restrictive. We will discuss the treatment of timed systems with τ loops in Chapter 9.

Similarly as in [99] we also define a Markovian process algebra (called calculus of IWMCs) which produces IWMCs as semantic models of its terms. The calculus of IWMC is an extension of the basic process algebra introduced in Chapter 2 with prefixes "$\lambda.\_$", representing exponentially timed delays of rate λ, and prefixes "$w.\_$", representing probabilistic choices performed with weight w.

As usual in Markovian process algebras, choices between exponentially timed delays expressed with the operator "$\_ + \_$" simply become alternative exponentially timed transitions in semantic models, hence are resolved through a *race policy*. Due to the adoption of such a race policy, the parallel execution of exponentially timed delays can be simply represented with an *interleaving* semantics. For instance, if we consider the parallel execution of two delays λ and μ which is represented by:



we have that an expansion law like $\lambda.\underline{0} \,\|\, \mu.\underline{0} = \lambda.\mu.\underline{0} + \mu.\lambda.\underline{0}$ holds true. We point out that the CTMC above correctly depicts the aforementioned scenario thanks to the memoryless property of the exponential distribution, because an action can be regarded as being initiated in the same state in which it terminates its execution. For instance, if in the initial state of the CTMC above (i.e., the state labeled with initial probability 1) the action with rate λ is terminated before the action with rate μ, the leftmost state is reached and its outgoing transition is labeled with μ because, when entering that state, the time to the completion of the action with rate μ is still exponentially distributed with rate μ. We also observe that no transition is possible from the initial state to the absorbing one as the probability that the two actions terminate simultaneously is zero.

Similarly to exponentially timed delay prefixes, choices between weight prefixes expressed with the operator "$\_+\_$" simply become alternative weighted transitions in semantic models, thus representing probabilistic choices. Moreover, since probabilistic choices are performed in zero time, we can simply adopt an interleaving semantics also for the parallel execution of weight prefixes.

Therefore, the semantics of terms of the calculus of IWMCs is simply defined via SOS by extending the standard interleaving semantics for action prefixes presented in Chapter 2 also to exponentially timed delay prefixes and weight prefixes. Moreover, the interrelation among the three kind of prefixes is captured in the semantics by: (i) applying a notion of priority of weight prefixes over exponentially timed delay prefixes due to the assumption that probabilistic choices are resolved urgently, and (ii) applying a notion of priority of τ prefixes over performance related prefixes due to the maximal progress assumption previously explained. We also show that the notion of observational congruence over *IWMCs*, obtained by modifying weak bisimulation over *IWMCs* in the standard way [122], is a congruence for all the operators of the calculus over strongly guarded processes and we produce an axiomatization for this equivalence which is complete over finite-state strongly guarded processes.

The chapter is structured as follows. In Sect. 4.2 we present the model of *IWMCs*. In Sect. 4.3 we present the calculus of *IWMCs* and its operational semantics. Finally, in Sect. 4.4 we present the notion of observational congruence over *IWMCs* and its complete axiomatization.

4.2 The IWMC Model

The model of interactive weighted Markov chains extends CTMCs by expressing in addition to the execution of exponentially timed delays, also (i) the execution of standard actions which can synchronize and have a zero duration, and (ii) the execution (in zero time) of probabilistic choices expressed through transitions labeled with weights. As far as the first extension is concerned, it leads to models which are also quite usual in real-time process algebras [131] where transitions representing temporal delays are distinguished from standard action transitions which are chosen non-deterministically and are performed in zero time. Such an extension leads to the possibility of modeling a concurrent system as the (parallel) composition of several subsystems. As far as the extension of CTMCs to probabilistic choices is concerned, it leads to an enhancement of the expressive power of IMCs [99] similar to that obtained in [141], while preserving the simplicity of the approach. IWMCs can express three kind of choices: exponentially timed choices, probabilistic choices and non-deterministic choices. As far as exponentially timed choices are concerned, whenever a state of the IWMC enables several ordinary CTMC transitions labeled with rates $\lambda \in \mathbb{R}^+$, the choice is resolved according to *r*ace policy by executing the transition which terminates first. As far as probabilistic choices are concerned, whenever a state of the IWMC enables several probabilistic transitions, the choice of the transition is performed probabilistically according to the weights $w \in \mathbb{R}^+$ labeling such transitions. As far as non-deterministic choices are concerned, they arise whenever an IWMC state enables several action transitions, just like in standard labeled transition systems deriving from CCS/CSP terms. In particular alternative transitions labeled with invisible τ actions represent internal non-deterministic choices which are performed in zero time and can never be "resolved" through synchronization with other system components. On the contrary, visible actions a in an IWMC are seen as incomplete actions which wait for a synchronization with other system components (they represent potential interaction with the environment). Therefore the choice of such actions in any IWMC state is governed by an external form of non-determinism, as their execution completely depends on the environment. Note that since we adopt a CSP synchronization policy for IWMCs which produces visible actions from the synchronization of visible actions (thus allowing for multiway synchronization) the only way to turn an incomplete action into a complete one is by means of a hiding operator, which turns visible actions into τ actions. Similarly as in [99] an IWMC represents a *complete system* only when it does not include any transition labeled by a visible action. Note that IWMCs representing complete systems may still include non-determinism due to multiple internal τ transitions enabled in the same state (internal non-determinism). Therefore, adversaries (or schedulers) play an important role in the performance analysis of IWMCs in that they allow internal non-determinism to be removed from an IWMC thus turning it into a CTMC.

More precisely, in an IWMC we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. In such states the IWMC just performs a non-deterministic choice among the τ transitions in zero time and may potentially interact with the environment through one of the visible actions.

- *probabilistic states*, enabling probabilistic transitions and (possibly) visible action transitions a only. In such states (also called vanishing states) the IWMC just performs a probabilistic choice among the probabilistic transitions in zero time and may potentially interact with the environment through one of the visible actions.

- *timed states*, enabling exponentially timed transitions and (possibly) visible action transitions a only. The IWMC sojourns in these states (also called tangible states) until one of the exponential delays terminates and the corresponding transition is performed. While the IWMC sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.

- *waiting states*, enabling standard visible actions only or no transition at all. In such states the IWMC sojourns indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions.

In the following we present the formal definition of Interactive Weighted Markovian Transition System (IWMTS), then we will define interactive weighted Markov chains as IWMTSs possessing an initial state. Formally, rates, belonging to $\mathbb{R}^+$, are ranged over by $\lambda, \lambda', \ldots$ while weights, belonging to $\mathbb{R}^+$, are ranged over by $w, w', \ldots$. We use $\theta, \theta', \ldots$ to range over both rates and weights. Moreover, we denote the set of standard action types used in a IWMTS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of states of an IWMTS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of IWMTSs easier. Let us suppose that $T \subseteq (\Sigma \times \mathit{Labels} \times \Sigma)$ is a transition relation, where Labels is a set of transition labels, ranged over by l. We use $s \xrightarrow{l} s'$ to stand for $(s, l, s') \in T$, $s \xrightarrow{l}$ to stand for $\exists s' : s \xrightarrow{l} s'$, and $s \not\xrightarrow{l}$ to stand for $\nexists s' : s \xrightarrow{l} s'$.

Definition 4.1 *An Interactive Weighted Markovian Transition System (IWMTS) is a tuple* $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a)$ *with*

- Σ *a set of states,*
- Act *a set of standard actions,*
- $T_w \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$, $T_e \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$, *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *three transition relations, containing probabilistic, exponentially timed and action transitions, respectively, such that:* [1]

  1. $\forall s \in \Sigma.\ s \xrightarrow{\tau} \implies \nexists \theta.\, s \xrightarrow{\theta}$
  2. $\forall s \in \Sigma.\ \exists w.\, s \xrightarrow{w} \implies \nexists \lambda.\, s \xrightarrow{\lambda}$

■

Definition 4.2 *An Interactive Weighted Markov Chain (IWMC) is a tuple* $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a, s_0)$, *where* $s_0 \in \Sigma$ *is the initial state of the IWMC and* $(\Sigma, Act, T_w, T_e, T_a)$ *is an IWMTS.* ■

The constraints over transition relations T_w, T_e and T_a guarantee that each state of the IWMC belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ actions then it cannot perform exponentially timed or probabilistic transitions. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of delays. The second requirement says that if a state can perform probabilistic transitions then it cannot perform exponentially timed transitions. Such a property derives from the assumption of *urgency of choices*: probabilistic choices cannot be delayed but must be performed immediately, hence they prevent the execution of exponentially timed delays.

[1]For the sake of readability here and in the rest of the chapter we assume the following operator precedence when writing constraints for transition relations: existential quantifier $>$ implication.

4.3 The Calculus of Interactive Weighted Markov Chains

In this section we introduce the syntax and operational semantics of the calculus of interactive weighted Markov chains.

4.3.1 Syntax of Terms and Informal Semantics of Operators

The calculus of interactive weighted Markov chains is an extension of the standard process algebra introduced in Chapter 2, which allows us to express priority, probabilistic choices and exponentially timed delays. [2] This is done by including into the calculus, in addition to standard actions, actions w representing probabilistic choices with weight w and actions λ representing exponential delays with rate λ. The possibility of expressing priority derives from the interrelation of standard τ actions, w actions and λ actions. In particular we make the maximal progress assumption: the system cannot wait if it has something internal to do. Therefore we assume that, in a choice, τ actions have priority over w actions and λ actions, i.e. both $\tau.P + w.Q$ and $\tau.P + \lambda.Q$ behave as $\tau.P$. Moreover we assume that probabilistic choices are urgent, hence, in a choice, w actions have priority over λ actions, i.e. $w.P + \lambda.Q$ behave as $w.P$.

Let Var be a set of process variables ranged over by X, Y, Z. Let $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi(\tau) = \tau \wedge \varphi(Act - \{\tau\}) \subseteq Act - \{\tau\}\}$ be a set of *action relabeling functions*, ranged over by φ.

Definition 4.3 *We define the language IWMC as the set of terms generated by the following syntax*

$$P ::= \underline{0} \mid X \mid w.P \mid \lambda.P \mid \alpha.P \mid P + P \mid P/L \mid P[\varphi] \mid P \,\|_S\, P \mid recX.P$$

where $L, S \subseteq Act - \{\tau\}$. An IWMC process is a closed term of IWMC. We denote by $IWMC_g$ the set of strongly guarded terms of IWMC. [3] ■

"$\underline{0}$" denotes a process that cannot move. The operators "." and "+" are the CCS prefix and choice. The choice among weights is carried out through the preselection policy by giving each of them a probability proportional to its weight. The choice among rates is carried out through the race policy by executing them concurrently until one of them terminates. Moreover τ actions have priority over w and λ actions in a choice, while w actions have priority over λ actions. "$/L$" is the hiding operator which turns into τ the actions in L, "$[\varphi]$" is the relabeling operator which relabels visible actions according to φ. "$\|_S$" is the CSP parallel operator, where synchronization over actions in S is required. Finally "$recX$" denotes recursion in the usual way.

In the following we will consider only terms of $IWMC_g$, i.e. strongly guarded terms of our calculus. A justification for the fact that we do not consider processes with weakly guarded recursion is that they give rise to τ loops in the semantic models, hence to Zeno processes, i.e. processes which perform infinite

[2] To be precise, with respect to the algebra considered in Chapter 2, here we use a recursion operator "$recX.\_$", instead of process constants, to denote recursion.

[3] We consider w and λ prefixes as being guards in the definition of strong guardedness.

$$\alpha.P \xrightarrow{\alpha} P$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \,\|_S\, Q \xrightarrow{\alpha} P' \,\|_S\, Q}\ \alpha \notin S \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \,\|_S\, Q \xrightarrow{\alpha} P \,\|_S\, Q'}\ \alpha \notin S$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \,\|_S\, Q \xrightarrow{a} P' \,\|_S\, Q'}\ a \in S$$

$$\frac{P \xrightarrow{a} P'}{P/L \xrightarrow{\tau} P'/L}\ a \in L \qquad \frac{P \xrightarrow{\alpha} P'}{P/L \xrightarrow{\alpha} P'/L}\ a \notin L$$

$$\frac{P \xrightarrow{\alpha} P'}{P[\varphi] \xrightarrow{\varphi(\alpha)} P'[\varphi]} \qquad \frac{P\{recX.P/X\} \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Table 4.1: Standard Rules

computations without going beyond a certain point in time. We will discuss the treatment of weakly guarded processes in Chapter 9.

4.3.2 Operational Semantics

The semantics of IWMC terms produces a transition system labeled by actions in Act, weights in $\mathbb{R}^+$ and rates in $\mathbb{R}^+$. We use $\gamma, \gamma', \ldots$ to range over transition labels. Such a transition system is defined as being the *IWMTS* $\mathcal{M} = (IWMC_g, Act, T_w, T_e, T_a)$, where: T_a is the least subset of $IWMC_g \times Act \times IWMC_g$ satisfying the standard operational rules of Table 4.1, T_w is obtained from the least multiset over $IWMC_g \times \mathbb{R}^+ \times IWMC_g$ satisfying the operational rules of Table 4.2 (similarly to [107], we consider a transition to have arity m if and only if it can be derived in m possible ways from the operational rules) by summing the weights of the multiple occurrences of the same transition, and T_e is obtained from the least multiset over $IWMC_g \times \mathbb{R}^+ \times IWMC_g$ satisfying the operational rules of Table 4.3 by summing the rates of the multiple occurrences of the same transition. In Tables 4.2 and 4.3 we use $P \xrightarrow{a}$ to stand for $\exists P' : P \xrightarrow{a} P'$, $P \stackrel{\tau}{\not\longrightarrow}$ to stand for $\nexists Q : P \xrightarrow{\tau} Q$ and $P \stackrel{w}{\not\longrightarrow}$ to stand for $\nexists w, Q : P \xrightarrow{w} Q$.

The rules of Table 4.2 define probabilistic transitions, by taking into account the priority of "τ" actions over weights. Note that we consider a "global" kind of weights which are applied also across the parallel operator. Moreover we can just interleave parallel weight transitions because they are executed in zero time. The rules of Table 4.3 define exponentially timed transitions by taking into account the priority of "τ"

$$w.P \xrightarrow{w} P$$

$$\frac{P \xrightarrow{w} P' \wedge Q \not\xrightarrow{\tau}}{P + Q \xrightarrow{w} P'} \qquad \frac{Q \xrightarrow{w} Q' \wedge P \not\xrightarrow{\tau}}{P + Q \xrightarrow{w} Q'}$$

$$\frac{P \xrightarrow{w} P' \wedge Q \not\xrightarrow{\tau}}{P \parallel_S Q \xrightarrow{w} P' \parallel_S Q} \qquad \frac{Q \xrightarrow{w} Q' \wedge P \not\xrightarrow{\tau}}{P \parallel_S Q \xrightarrow{w} P \parallel_S Q'}$$

$$\frac{P \xrightarrow{w} P' \wedge \nexists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{w} P'/L} \qquad \frac{P \xrightarrow{w} P'}{P[\varphi] \xrightarrow{w} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{w} P'}{recX.P \xrightarrow{w} P'}$$

Table 4.2: Rules for Probabilistic Moves

$$\lambda.P \xrightarrow{\lambda} P$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \not\xrightarrow{\tau} \wedge Q \not\xrightarrow{w}}{P + Q \xrightarrow{\lambda} P'} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \not\xrightarrow{\tau} \wedge P \not\xrightarrow{w}}{P + Q \xrightarrow{\lambda} Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge Q \not\xrightarrow{\tau} \wedge Q \not\xrightarrow{w}}{P \parallel_S Q \xrightarrow{\lambda} P' \parallel_S Q} \qquad \frac{Q \xrightarrow{\lambda} Q' \wedge P \not\xrightarrow{\tau} \wedge P \not\xrightarrow{w}}{P \parallel_S Q \xrightarrow{\lambda} P \parallel_S Q'}$$

$$\frac{P \xrightarrow{\lambda} P' \wedge \nexists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{\lambda} P'/L} \qquad \frac{P \xrightarrow{\lambda} P'}{P[\varphi] \xrightarrow{\lambda} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{\lambda} P'}{recX.P \xrightarrow{\lambda} P'}$$

Table 4.3: Rules for Exponentially Timed Moves

actions and weights over rates. Such rules are similar to those considered in [99] apart from the way in which priority of "τ" actions is expressed: while in [99] such a priority is captured in the definition of equivalence among systems, we cut transitions that cannot be performed directly in semantic models.

Note that even if the operational rules in Tables 4.2 and 4.3 include negative premises, this does not cause inconsistencies because when applying such rules for deriving the moves of a term P, the negative premises always refer to the moves of a subterm of P (and not of P itself), hence the operational semantics is stratifiable [89].

We are now in a position to define the *IWMC* obtained as the semantic model of a process of the calculus.

Definition 4.4 *The semantic model $\mathcal{M}[\![P]\!]$ of $P \in IWMC_g$ is the IWMC defined by:*

$$\mathcal{M}[\![P]\!] = (S_P, Act, T_{w,P}, T_{e,P}, T_{a,P}, P)$$

where:

- *S_P is the least subset of $IWMC_g$ such that:*
  - $P \in S_P$
  - *if $P' \in S_P$ and $P' \xrightarrow{\gamma} P''$, then $P'' \in S_P$*
- *$T_{w,P}, T_{e,P}$ and $T_{a,P}$ are the restriction of T_w, T_e and T_a to $S_P \times Act \times S_P$, $S_P \times \mathbb{R}^+ \times S_P$ and $S_P \times \mathbb{R}^+ \times S_P$.* ■

4.4 Observational Congruence for IWMCs

Now we will introduce a notion of strong and weak bisimulation over IWMCs and a notion of observational congruence over terms of our calculus. In particular observational congruence deals with exponentially timed choices according to Markovian bisimulation [107] deals with probabilistic choices according to probabilistic bisimulation [117], and abstracts from standard τ actions as in [122]. Then we will show that the observational congruence is indeed a congruence with respect to all the operator of our calculus.

In our context we express cumulative probabilities and cumulative exponential times by aggregating weights and rates, respectively.

Definition 4.5 *Let $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a)$ be a IWMTS. The function $TW : \Sigma \times PDF^+ \times \mathcal{P}(\Sigma) \longrightarrow \mathbb{R}^+ \cup \{0\}$, which computes the aggregated weight that a state $s \in \Sigma$ reaches a set of states $I \in \mathcal{P}(\Sigma)$ is defined as:* [4]

$$TW(s, I) = \sum \{\!|\, w \mid \exists s' \in I.\ (s, w, s') \in T_w \,|\!\}$$

[4] We use $\{\!|$ and $|\!\}$ to denote multiset parentheses. The summation of an empty multiset is assumed to yield 0.

Similarly the function $TR : \Sigma \times PDF^{+} \times \mathcal{P}(\Sigma) \longrightarrow \mathbb{R}^{+} \cup \{0\}$, *which computes the aggregated rate that a state* $s \in \Sigma$ *reaches a set of states* $I \in \mathcal{P}(\Sigma)$ *is defined as:*

$$TR(s, I) = \sum \{\!| \lambda \mid \exists s' \in I.\ (s, \lambda, s') \in T_e |\!\}$$

■

We are now in a position to define the notion of strong bisimilarity for IWMCs.

Definition 4.6 *Let* $\mathcal{M} = (\Sigma, Act, T_w, T_e, T_a)$ *be a IWMTS. An equivalence relation* β *on* Σ *is a strong bisimulation iff* $s_1\ \beta\ s_2$ *implies*

- *for every* $\alpha \in Act$ *and* $s_1' \in \Sigma$,

  $$s_1 \xrightarrow{\alpha} s_1' \textit{ implies } s_2 \xrightarrow{\alpha} s_2' \textit{ for some } s_2' \textit{ with } s_1'\ \beta\ s_2',$$

- *for every equivalence class* I *of* β,

  $$TW(s_1, I) = TW(s_2, I) \textit{ and } TR(s_1, I) = TR(s_2, I)$$ [5]

Two states s_1 *and* s_2 *are strongly bisimilar, denoted by* $s_1 \sim s_2$, *iff* (s_1, s_2) *is included in some strong bisimulation. Two IWMCs* $(\mathcal{M}_1, s_{0,1})$ *and* $(\mathcal{M}_2, s_{0,2})$ *are strongly bisimilar, if their initial states* $s_{0,1}$ *and* $s_{0,2}$ *are strongly bisimilar in the IWMTS obtained with the disjoint union of* $\mathcal{M}_1$ *and* $\mathcal{M}_2$. ■

The definition of weak bisimilarity is an adaptation of that presented in [105, 99] to our context.

Let $\stackrel{\alpha}{\Longrightarrow}$ denote $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$, i.e. a sequence of transitions including a single α transition and any number of τ transitions. Moreover we define $\stackrel{\hat{\alpha}}{\Longrightarrow}\ =\ \stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow}\ = (\stackrel{\tau}{\longrightarrow})^*$, i.e. a possibly empty sequence of τ transitions. Moreover we let I^{τ} denote the set of processes that may silently evolve into an element of I, i.e. $I^{\tau} = \{s_1 \mid \exists s_2 \in I : s_1 \stackrel{\hat{\tau}}{\Longrightarrow} s_2\}$.

Definition 4.7 *Let* $\mathcal{M} = (\alpha, Act, T_w, T_e, T_a)$ *be a IWMTS. An equivalence relation* β *on* α *is a weak bisimulation iff* $s_1\ \beta\ s_2$ *implies*

- *for every* $\alpha \in Act$ *and* $s_1' \in \Sigma$,

  $$s_1 \stackrel{\hat{\alpha}}{\Longrightarrow} s_1' \textit{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \textit{ for some } s_2' \textit{ with } s_1'\ \beta\ s_2',$$

- *for every* $s_1' \in \Sigma$,

  $$s_1 \stackrel{\hat{\tau}}{\Longrightarrow} s_1' \textit{ implies } s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2' \textit{ for some } s_2' \textit{ such that, for every equivalence class } I \textit{ of } \beta,$$

  $$TW(s_1', I^{\tau}) = TW(s_2', I^{\tau}) \textit{ and } TR(s_1', I^{\tau}) = TR(s_2', I^{\tau})$$

Two states s_1 *and* s_2 *are weakly bisimilar, denoted by* $s_1 \approx s_2$, *iff* (s_1, s_2) *is included in some weak bisimulation. Two IWMCs* $(\mathcal{M}_1, s_{0,1})$ *and* $(\mathcal{M}_2, s_{0,2})$ *are weakly bisimilar, if their initial states* $s_{0,1}$ *and* $s_{0,2}$ *are weakly bisimilar in the IWMTS obtained with the disjoint union of* $\mathcal{M}_1$ *and* $\mathcal{M}_2$. ■

[5] According to the definition of IWMTS for each state s of an IWMTS at least one of $TW(s, I)$ and $TR(s, I)$ must be zero.

Differently from [105, 99] we do not express conditions about the stability of bisimilar states because we are interested in obtaining a congruence result only for strongly guarded processes of our calculus. Such processes cannot produce an IWMC which is forced in a τ loop, hence we do not have to recognize this situation.

Similarly as in [105, 99] it is possible to reformulate weak bisimilarity in the following way, which is simpler but less intuitive.

Theorem 4.1 *Let $\mathcal{M} = (\alpha, Act, T_w, T_e, T_a)$ be a IWMTS. An equivalence relation β on α is a weak bisimulation iff $s_1 \; \beta \; s_2$ implies*

- *for every $\alpha \in Act$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \overset{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta \; s_2',$$

- *$s_2 \overset{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that, for every equivalence class I of β,*

  $$TW(s_1, I) = TW(s_2', I) \text{ and } TR(s_1, I) = TR(s_2', I)$$ ■

The proof that this reformulation is correct derives from that given in [99], simply by treating weights as rates of exponential distributions.

Now we introduce the notion of observational congruence over $IWMC_g$ processes, and we will show it to be a congruence with respect to all the operators of our calculus. Such a notion is defined, starting from the notion of weak bisimulation over IWMCs, according to the classical notion of observational congruence [122] and probabilistic bisimulation [117].

Definition 4.8 *Two closed terms P, Q of $IWMC_g$ are observational congruent, written $P \simeq Q$, iff:*

- *for every $\alpha \in Act$ and $P' \in IWMC_g$,*

  $$P \xrightarrow{\alpha} P' \text{ implies } Q \overset{\alpha}{\Longrightarrow} Q' \text{ for some } Q' \text{ with } P' \approx Q',$$

- *for every $\alpha \in Act$ and $Q' \in IWMC_g$,*

  $$Q \xrightarrow{\alpha} Q' \text{ implies } P \overset{\alpha}{\Longrightarrow} P' \text{ for some } P' \text{ with } P' \approx Q',$$

- *for every equivalence class I of β,*

  $$TW(P, I) = TW(Q, I) \text{ and } TR(P, I) = TR(Q, I)$$ ■

We consider $\simeq$ as being defined also on the open terms of $IWMC_g$ by extending observational congruence with the standard approach of [122].

Theorem 4.2 *$\simeq$ is a congruence over terms of $IWMC_g$ w.r.t. all the operators of $IWMC$, including recursion.*

Proof The proof of this theorem follows the lines of the similar proof in [122] that is adapted to our setting.

■

4.4.1 Axiomatization

In this section we present an axiom system which is complete for $\simeq$ on finite state $IWMC_g$ terms.

The axiom system $\mathcal{A}_{IWMC}$ for $\simeq$ on $IWMC_g$ terms is formed by the axioms presented in Fig. 4.1. In this figure "$\|\!\lfloor$" and "$|$" denote, respectively, the left merge and synchronization merge operators (see e.g. [1]). We recall from Sect. 4.3.2 that θ ranges over weights in $\mathbb{R}^+$ and rates in $\mathbb{R}^+$, while $\gamma, \gamma', \ldots$ range over actions in Act, weights and rates.

$(A1)$	$P + Q = Q + P$		$(A2)$	$(P+Q)+R = P+(Q+R)$	
$(A3)$	$\alpha.P + \alpha.P = \alpha.P$		$(A4)$	$P + \underline{0} = P$	
$(Tau1)$	$\gamma.\tau.P = \gamma.P$		$(Tau2)$	$P + \tau.P = \tau.P$	
$(Tau3)$	$\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$				
$(Prob)$	$w.P + w'.P = (w + w').P$				
$(ExpT)$	$\lambda.P + \lambda'.P = (\lambda + \lambda').P$				
$(Pri1)$	$\tau.P + \theta.Q = \tau.P$		$(Pri2)$	$w.P + \lambda.Q = w.P$	
$(Hi1)$	$\underline{0}/L = \underline{0}$		$(Hi2)$	$(\gamma.P)/L = \gamma.(P/L)$	$\gamma \notin L$
$(Hi3)$	$(a.P)/L = \tau.(P/L)$	$a \in L$	$(Hi4)$	$(P+Q)/L = P/L + Q/L$	
$(Rel1)$	$\underline{0}[\varphi] = \underline{0}$		$(Rel2)$	$(\alpha.P)[\varphi] = \varphi(\alpha).(P[\varphi])$	
$(Rel3)$	$(\theta.P)[\varphi] = \theta.(P[\varphi])$		$(Rel4)$	$(P+Q)[\varphi] = P[\varphi] + Q[\varphi]$	
(Par)	$P \parallel_S Q = P \mathbin{\|\!\lfloor}_S Q + Q \mathbin{\|\!\lfloor}_S P + P \mid_S Q$				
$(LM1)$	$\underline{0} \mathbin{\|\!\lfloor}_S P = \underline{0}$				
$(LM2)$	$(a.P) \mathbin{\|\!\lfloor}_S Q = \underline{0}$	$a \in S$			
$(LM3)$	$(\gamma.P) \mathbin{\|\!\lfloor}_S Q = \gamma.(P \parallel_S Q)$	$\gamma \notin S$			
$(LM4)$	$(P+Q) \mathbin{\|\!\lfloor}_S R = P \mathbin{\|\!\lfloor}_S R + Q \mathbin{\|\!\lfloor}_S R$				
$(SM1)$	$P \mid_S Q = Q \mid_S P$				
$(SM2)$	$\underline{0} \mid_S P = \underline{0}$				
$(SM3)$	$(\gamma.P) \mid_S (\gamma'.Q) = \underline{0}$	$(\gamma \notin S \ \vee\ \gamma \neq \gamma') \wedge \tau \notin \{\gamma, \gamma'\}$			
$(SM4)$	$(\tau.P) \mid_S Q = P \mid_S Q$				
$(SM5)$	$(a.P) \mid_S (a.Q) = a.(P \parallel_S Q)$	$a \in S$			
$(SM6)$	$(P+Q) \mid_S R = P \mid_S R + Q \mid_S R$				
$(Rec1)$	$recX.P = recY.(P\{Y/X\})$	provided that Y is not free in recX.P			
$(Rec2)$	$recX.P = P\{recX.P/X\}$				
$(Rec3)$	$Q = P\{Q/X\} \Rightarrow Q = recX.P$	provided that X is strongly guarded in P			

Figure 4.1: Axiomatization for IWMC

The axioms $(Pri1)$ and $(Pri2)$ express the two kinds of priorities of $IWMC$, respectively, priority of τ actions over weights and rates and priority of weights over rates. The axiom (Par) is the standard one which expresses parallel composition in terms of left and synchronization merge. The axioms ($Rec1-3$) handle strongly guarded recursion in the standard way [124].

If we consider the obvious operational rules for "$\lfloor\lfloor_S$" and "$|_S$" that derive from those we presented for the parallel operator [6] then the axioms of $\mathcal{A}_{IWMC}$ are sound.

A sequential state is defined to be one which includes "$\underline{0}$", "X" and operators ".", "+", "$recX$" only; leading to the following theorem.

Theorem 4.3 *If an IWMC$_g$ process P is finite state, then $\exists P' : \mathcal{A}_{IWMC} \vdash P = P'$ with P' sequential state.*

Proof Let $s_1 \dots s_n$ be the states of the operational semantics of P, $s_n \equiv P$. It can be easily seen that for each $i \in \{1 \dots n\}$, there exist J_i and $\theta_{i,j}$, $k_{i,j}$ with $j \in J_i$ such that $\mathcal{A}_{IWMC} \vdash s_i = \sum_{j \in J_i} \theta_{i,j}.s_{k_{i,j}}$ where $\sum_\emptyset \equiv \underline{0}$. Then for each i, from 1 to n, we do the following. If i is such that $\exists j \in J_i : k_{i,j} = i$ we have, by applying ($Rec3$), that $s_i = recX.(\sum_{j \in J_i : k_{i,j} \neq i} \theta_{i,j}.s_{k_{i,j}} + \sum_{j \in J_i : k_{i,j} = i} \theta_{i,j}.X)$. Then we replace each subterm s_i occurring in the equations for $s_{i+1} \dots s_n$ with its equivalent term. When, in the equation for $s_n \equiv P$, we have replaced s_{n-1}, we are done. ■

For sequential states the axioms of $\mathcal{A}_{IWMC}$ involved are just the standard axioms of [124], and the axioms for priority and probabilistic and exponentially timed choice. From Theorem 4.3 and by resorting to arguments similar to those presented in [124] and [99] we derive the completeness of $\mathcal{A}_{IWMC}$.

Theorem 4.4 *$\mathcal{A}_{IWMC}$ is complete for $\simeq$ over finite state IWMC$_g$ processes.*

Proof The proof of this theorem follows the lines of the proof of [124]. In particular weights are treated as rates of exponential distributions in the proof of [99]. ■

4.5 Conclusion

Several algebraic languages which express exponentially distributed durations like the calculus of *IWMCs* have been previously developed. A complete overview of such languages will be given in the conclusion of Chapter 11. Here we would like just to compare the calculus of *IWMCs* with the algebra PM-TIPP introduced in [141], which basically constitutes an alternative way of extending the calculus of *IMCs* [99] with probabilistic choices. The main difference between our calculus and PM-TIPP is that, while in PM-TIPP "sequences" of subsequent probabilistic choices are aggregated by the operational semantics into single probabilistic transitions, in the calculus of *IWMCs* such aggregation is not performed. The main drawback is that, since in our approach the aggregation of probabilistic transitions is performed not even when establishing equivalence, we may consider two systems to be different just because they carry out the same probabilistic choice in a different number of steps. On the other hand, by adopting a simpler

[6] The definition of the operational rule for "$|_S$" must allow for actions "τ" to be skipped [1], as reflected by axiom ($SM4$).

approach with respect to that of [141], we have obtained a simpler theory that has led to the development of an axiomatization which is complete over finite-state processes. Moreover, the model state-space reduction that would have been obtained by aggregating probabilistic transitions can nevertheless be carried out when deriving the underlying CTMC as described in Sect. 4.1.

Chapter 5

Interactive Timed Automata

In this chapter we introduce Interactive Timed Automata (ITA), a variant of classical Timed Automata [11, 132], where action executions, clock reset events and events enabled on the basis of clock constraints are expressed by means of separate transitions. The advantage of ITA with respect to existing timed automata is that action transitions can be dealt with separately from time-related transitions, hence making it easy to define, e.g., a notion of weak bisimulation. Moreover in ITA, similarly as for IWMCs (see Chapter 4), the interrelation of the different kind of transitions gives the new possibility of expressing a form of priority in timed automata when composing them to obtain a concurrent system. We start by formalizing ITA and Interactive Prioritized Timed Transition Systems (IPTTSs) which are both closed under CSP parallel composition and hiding. Then, we introduce a semantics for ITA which maps ITA onto IPTTSs by explicitely representing time passage through time transitions. Finally, we show that two weakly bisimilar ITA give rise to two weakly bisimilar semantic models and that our semantic mapping is compositional with respect to both CSP parallel composition and hiding.

5.1 Introduction

Timed automata (see e.g. [11, 133, 132, 131] and the references therein) are formal description paradigms for the specification and analysis of real-time concurrent systems, i.e. systems which are critical from the viewpoint of time bounds. There are several different variants of timed automata which differ for expressiveness and technical details. Basically they are extensions of standard labeled transition systems which allow for the representation of time by means of *clocks*. The time value assumed by a clock in a timed automata increases as time passes. The transitions of a timed automata may cause the setting of a clock with a certain time value. Moreover we can associate time constraints to transitions, expressed as bounds on clocks, which must be satisfied in order to execute the transition. The definition of a particular class of timed automata usually involves the definition of their semantics, which maps a timed automaton into a transition system where time is no longer represented symbolically by means of clocks, but is represented explicitly via time transitions. Often a notion of parallel composition between timed automata is defined, so that a concurrent system can be represented as the parallel of several timed automata representing system components. Once the system is specified with a "net" of timed automata, the real-time properties that it must satisfy can be validated automatically through techniques such as e.g. *model checking* [11]. Finally the development of an adequate notion of equivalence between timed automata which is a congruence with respect to parallel composition is important to minimize the system state space, hence the system analysis.

In this chapter we develop a variant of classical timed automata, called Interactive Timed Automata (ITA), which aims at simplifying and improving equivalence reasoning for timed automata in the presence of composition operators. The idea is to express action executions, clock reset events and events enabled on the basis of clock constraints by means of separate transitions, instead of having one single kind of transition expressing all these features in a combined fashion (as usual in timed automata). In particular in an ITA we have three kind of transitions: *standard action transitions* labeled with a visible action or an invisible action τ, *clock reset transitions* labeled with a clock, and *clock bound transitions* labeled with an expression made up from constraints on clock values. Clock reset transitions are executed in zero time and cause the clock in their label to be set to 0, while clock bound transitions can be executed only if the expression in their label is satisfied. Similarly as for weighted transitions of IWMCs, we assume clock reset transitions to be urgent and to have priority over clock bound transitions which may let time pass. Moreover, as for IWMCs, we assume internal τ transitions to have priority over time related transitions due to the maximal progress assumption [133].

Employing three separate kind of transitions, instead of a single one, allows us to express bisimulation between timed automata as a simple extension to standard bisimulation over transition systems labeled with actions, similarly as for IWMCs (see Chapter 4). In particular, with respect to the existing equivalence notions for timed automata, it is easy to define a notion of weak bisimulation which abstracts from τ transitions, hence improving the capability of minimizing the state space of specified systems. With respect to existing timed automata, abstracting from τ transitions in ITA is particularly significant in that, similarly to IWMCs, we see visible action transitions as incomplete actions which wait for a synchronization with other system components. Therefore complete interactions in ITA specifications are not visible, so they can be abstracted away. In particular, as for IWMCs, we compose ITA in parallel by employing a CSP synchronization policy, which allows for multi-way synchronizations, and we also consider an hiding operator which turns visible actions into complete τ actions.

We also show that ITA can be straightforwardly mapped into existing timed automata (we will consider those defined in [132, 154]), hence previous decidability results and software tools can be exploited for analysing real-time properties in ITA specifications.

We then produce a semantics for ITA which maps an ITA onto a transition system where: (i) the passage of time is explicitely represented by transitions labeled with numeric time delays and (ii) clock reset transitions and clock bound transitions are turned into prioritized transitions reflecting the precedence of clock reset transitions over clock bound transitions. Differently from existing approaches, we express semantic models of ITA by means of "interactive" timed transition systems which can be themselves composed and for which we define a notion of weak bisimulation. This allows us to develop a semantic mapping which is compositional with respect to parallel composition and hiding.

We start (in Sect. 5.2) by formalising the model of ITA and by showing that they can be easily mapped into the time automata model of [132, 154]. Besides we show that the class of ITA is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over ITA. Then (in Sect. 5.3) we introduce the model of Interactive Prioritized Timed Transition Systems (IPTTSs) which includes: *standard action transitions*, representing the interactive behavior of a system component, *numeric time transitions* representing a fixed temporal delay and *prioritized transitions* representing events to be executed at different priority levels. Moreover we show that the class of IPTTSs is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over IPTTSs. Finally, (in Sect. 5.4) we present the semantics for ITA which maps ITA onto IPTTSs and we show that weakly bisimilar ITA give rise to weakly bisimilar semantic models and that the semantic mapping is compositional with respect to both CSP parallel composition and hiding.

5.2 Definition of Interactive Timed Automata

In this section we present the model of Interactive Timed Automata (ITA), we show that ITA can be mapped into the timed automata model of [132, 154] and that ITA are closed with respect to CSP parallel composition and hiding.

5.2.1 The ITA Model

Time delays are modeled in ITA by means of clocks C which are set to zero by means of reset transitions and count upwards while time passes. In the following we will distinguish different clocks used in an ITA through "names", where C_n denotes the clock with name n. An ITA represents the behavior of a system component by employing both clock reset transitions and clock bound transitions, representing the timed behavior of the component and standard action transitions, representing the interactive behavior of the component. Clock reset transitions are labeled with a clock name C_n and represent the event of reset of the clock (which is set to zero). After such event C_n counts upwards while time passes and states are traversed by the automaton. When several clock reset transitions are enabled in an ITA state, the choice among them is just non-deterministic. Clock bound transitions are labeled with a clock constraint ϕ (an expression built from bounds on the clock values) and they can be executed only when the status of clocks satisfies such a constraint. A system is allowed to stay in a state enabling several clock bound transitions as long as *all* clock constraints labeling the transitions can be satisfied. Action transitions are just standard CCS/CSP transitions labeled with actions a: when several action transitions are enabled in an ITA state, the choice among them is just performed non-deterministically and when ITA are composed in parallel they synchronize following the CSP [109] approach, where the actions belonging to a given set S are required to synchronize. In particular, similarly as for IWMCs (see Chapter 4), alternative transitions labeled with invisible τ actions represent internal non-deterministic choices which are performed in zero time and can never be "resolved" through synchronization with other system components. On the contrary, visible actions a in an ITA are seen as incomplete actions which wait for a synchronization with other system components (they represent potential interaction with the environment). Therefore the choice of such actions in any ITA state is governed by an external form of non-determinism, as their execution completely depends on the environment. Note that since we adopt a CSP synchronization policy for ITA which produces visible actions from the synchronization of visible actions (thus allowing for multiway synchronization) the only way to turn an incomplete action in a complete one is by means of a hiding operator, which turns visible actions into τ actions. Similarly as for IWMCs, an ITA represents a *complete system* only when it does not include any transition labeled by a visible action. Note that ITA representing complete systems may still include non-determinism due to multiple internal τ transitions enabled in the same state (internal non-determinism).

More precisely, in an ITA we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. In such states the ITA just performs a non-deterministic choice among the τ transitions in zero time and may potentially interact with the environment through one of the visible actions.

- *reset states*, enabling reset transitions C_n and (possibly) visible action transitions a only. In such states the ITA just performs a choice among the clock reset transitions in zero time and may potentially interact with the environment through one of the visible actions.

- *timed states*, enabling clock bound transitions ϕ and (possibly) visible action transitions a only. In such states all the clocks of the ITA count upwards as time passes. The system is allowed to sojourn in the state as long as *all* clock constraints labeling its outgoing transitions can be satisfied. Moreover, it can non-deterministically leave the state at any time through a bound transition ϕ whose constraint ϕ is (at that time) satisfied. Moreover, while the ITA sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.

- *waiting states*, enabling standard visible actions only or no transition at all. In such states the ITA sojourns indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions.

In the following we present the formal definition of Interactive Timed Automaton Transition System (ITATS), then we will define Interactive Timed Automata as ITATSs possessing an initial state. Formally, we use $T, T', \ldots$, representing sets of time values, to range over subsets of $\mathbb{R}^+ \cup \{0\}$. Moreover, we denote the set of standard action types used in an ITATS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of clocks of an ITATS is denoted by $\mathcal{C} = \{C_n \mid n \in \mathcal{C}Names\}$, where $\mathcal{C}Names$ is a set of clock names. Given a set $\mathcal{C}$, we denote with $\mathcal{C}^\Phi$, ranged over by $\phi, \phi', \ldots$, the set of constraints over clocks of $\mathcal{C}$ (the labels of clock bound transitions), which is defined as the set of terms generated by the following syntax:

$$\phi ::= C_n \in T \mid \phi \wedge \phi$$

Moreover, let $\mathcal{C} \cup \mathcal{C}^\Phi$ be ranged over by $\theta, \theta', \ldots$. The set of states of an ITATS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of ITATSs easier. Let us suppose that $T \subseteq (\Sigma \times Labels \times \Sigma)$ is a transition relation, where *Labels* is a set of transition labels, ranged over by l. We use $s \xrightarrow{l} s'$ to stand for $(s, l, s') \in T$, $s \xrightarrow{l}$ to stand for $\exists s' : s \xrightarrow{l} s'$, and $s \not\xrightarrow{l}$ to stand for $\nexists s' : s \xrightarrow{l} s'$.

Definition 5.1 *An Interactive Timed Automata Transition System (ITATS) is a tuple* $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ *with*

- Σ *a set of states,*
- $\mathcal{C}$ *a set of clocks,*
- *Act a set of standard actions,*
- $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^\Phi \times \Sigma)$, *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *three transition relations representing clock reset and clock bound events and action execution, respectively, such that:* [1]

  1. $\forall s \in \Sigma.$
     $s \xrightarrow{\tau} \implies \nexists \theta.\, s \xrightarrow{\theta}$
  2. $\forall s \in \Sigma.$
     $\exists C_n.\, s \xrightarrow{C_n} \implies \nexists \phi.\, s \xrightarrow{\phi}$

Definition 5.2 *An Interactive Timed Automata (ITA) is a tuple* $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, *where* $s_0 \in \Sigma$ *is the initial state of the ITA and* $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ *is an ITATS.* ■

The constraints over transition relations T_r, T_b and T_a guarantee that each state of the ITA belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ actions then it cannot perform clock reset transitions or clock bound transitions. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of time-related activity. The second requirement says that if a state can perform clock reset transitions then it cannot perform clock bound transitions. Such a property derives from the assumption of *urgency of clock resets*: clock reset transitions cannot be delayed but must be performed immediately, hence they prevent the execution of clock bound transitions.

[1] For the sake of readability here and in the rest of the chapter we assume the following operator precedence when writing constraints for transition relations: existential quantifier > "and" operator > implication.

5.2.2 Mapping ITA onto Ordinary Timed Automata

An ITA can be easily turned into an ordinary timed automata that can be analyzed with existing software tools, provided that it only makes use of clock constraints where sets of acceptable time values T are finite collections of intervals. In particular we show how ITA satisfying the above condition can be transformed into the timed automata of [132, 154] which can then be analyzed with the software tool Kronos [154]. Note that in this way we inherit the decidability results for model checking of formulae of the timed temporal logic TCTL [11] over the timed automata of [132, 154].

In [154] timed automata are defined to be transition systems where: (i) each transition is labeled with a set of actions, an enabling condition (built from constraints over clock values of the form $C_n \leq t$, $C_n < t$, $C_n \geq t$ or $C_n > t$ and conjunctions) and a clock assignment function; and (ii) each state is labeled with a condition representing a state invariant. The transformation is easily carried out as follows. We turn:

- each clock reset transition of the ITA into a transition which is labeled with an empty set of actions, is always enabled and sets the clock in the label of the ITA transition to the value 0;
- each clock bound transition of the ITA into a transition which is labeled with an empty set of actions, whose enabling condition is the condition labeling the ITA transition (it is possible to do this because we assumed that sets of acceptable time values T in the ITA clock constraints are finite collections of intervals), and performs no clock assignment;
- each action transition of the ITA into a transition which is labeled with the singleton containing the action labeling the ITA transition, is always enabled and performs no clock assignment.

As far as state invariants are concerned, we associate:

- with silent states an invariant which is always false,
- with reset states an invariant which is always false,
- with timed states an invariant which is the conjunction of all the conditions labeling the clock bound transitions leaving the timed state (i.e. we can sojourn in the state only as long as all clock bound transitions are "enabled"),
- with waiting states an invariant which is always true.

5.2.3 Parallel of ITA

Now, we address the problem of defining parallel composition à la CSP [109] of ITA, where the standard actions of a given set S are required to synchronize and the synchronization of two actions of type a is again an action of type a. Given a clock renaming function $ren : \mathcal{C} \longrightarrow \mathcal{C}$, we assume $ren(\phi)$ to be the constraint ϕ' obtained from ϕ by renaming clocks in ϕ according to function ren. In particular we define the renaming function $l : \mathcal{C} \longrightarrow \mathcal{C}$ by $\{(C_n, C_{n,l}) \mid C_n \in \mathcal{C}\}$ and, similarly, function $r : \mathcal{C} \longrightarrow \mathcal{C}$ by $\{(C_n, C_{n,r}) \mid C_n \in \mathcal{C}\}$.

Definition 5.3 *The parallel composition* $\mathcal{T}_1 \|_S \mathcal{T}_2$ *of two ITA* $\mathcal{T}_1 = (\Sigma_1, \mathcal{C}_1, Act, T_{r,1}, T_{b,1}, T_{a,1}, s_{0,1})$ *and* $\mathcal{T}_2 = (\Sigma_2, \mathcal{C}_2, Act, T_{r,2}, T_{b,2}, T_{a,2}, s_{0,2})$*, with* $S \subset Act - \{\tau\}$ *being the synchronization set, is the tuple* $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, (s_{0,1}, s_{0,2}))$ *with*

- $\Sigma = \Sigma_1 \times \Sigma_2 \times \mathcal{M}$ *the set of states,*
- $\mathcal{C} = \{C_{n,l} \mid C_n \in \mathcal{C}_1\} \cup \{C_{n,r} \mid C_n \in \mathcal{C}_2\}$
- $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^{\Phi} \times \Sigma)$, *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *are the least transition relations, such that* $\forall (s_1, s_2) \in \Sigma$.

$$\mathbf{1_l}\ s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$$

$$\mathbf{2}\ s_1 \xrightarrow{a} s_1' \ \wedge\ s_2 \xrightarrow{a} s_2',\ a \in S \implies (s_1, s_2) \xrightarrow{a} (s_1', s_2')$$

$$\mathbf{3_l}\ s_1 \xrightarrow{C_n} s_1' \ \wedge\ s_2 \stackrel{\tau}{\not\longrightarrow} \implies (s_1, s_2) \xrightarrow{C_{n,l}} (s_1', s_2)$$

$$\mathbf{4_l}\ s_1 \xrightarrow{\phi} s_1' \ \wedge\ s_2 \stackrel{\tau}{\not\longrightarrow} \ \wedge\ \nexists C_n.\, s_2 \xrightarrow{C_n} \implies (s_1, s_2) \xrightarrow{l(\phi)} (s_1', s_2)$$

and also the symmetric rules $\mathbf{1_r}, \mathbf{3_r}, \mathbf{4_r}$ *referring to the local transitions of* $\mathcal{T}_2$*, which are obtained from the rules* $\mathbf{1_l}, \mathbf{3_l}, \mathbf{4_l}$ *by exchanging the roles of states* s_1 (s_1') *and* s_2 (s_2')*, by turning* l *into* r *in the subscripts of clocks, and by turning the renaming function* l *into* r*, hold true.*

- $(s_{0,1}, s_{0,2}) \in \Sigma$ *the initial state* ■

Each state $s \in \Sigma$ of the composed model is represented by a pair of states ($s_1 \in \Sigma_1$ and $s_2 \in \Sigma_2$). Moreover we rename clocks of both ITA $\mathcal{T}_1$ and $\mathcal{T}_2$ so to avoid a name conflict whenever two clocks with the same name C_n are simultaneously in execution in both ITA. Rules **1** (**2**) describe the behavior of the composed model in the case of a standard action α performed by one (or both, via a synchronization) ITA, when $\alpha \notin S$ ($\alpha \in S$). Rules **3** and **4** define the behavior of the composed model in the case of clock reset and clock bound transitions, respectively, locally performed by components. Note that the negative clauses in the premises enforce the maximal progress and the urgency of clock resets assumptions.

Theorem 5.1 *Let* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *be two ITA. Then for each* $S \subseteq Act - \{\tau\}$, $\mathcal{T}_1 \|_S \mathcal{T}_2$ *is an ITA.*
Proof *Just a trivial consequence of the fact that the maximal progress and urgency of clock resets assumptions are enforced when composing ITA.* ■

5.2.4 Hiding of ITA

Now, we address the problem of defining hiding of ITA, where the standard actions of a given set L are turned into invisible τ actions.

As we already explained, the capability of hiding actions make it possible to turn visible "incomplete" actions into invisible "complete" ones, thus giving the possibility of building a complete system from several system components. In particular while a visible action transition (as long as it is enabled) can delay indefinitely before being performed, when such an action is turned into an invisible action it must be executed in zero time.

Now we formally define the hiding $\mathcal{T}/L$ of a ITA $\mathcal{T}$, where the set L of the visible actions to be hidden is a subset of $Act - \{\tau\}$.

Definition 5.4 *The hiding $\mathcal{T}/L$ of a ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_{r,1}, T_{b,1}, T_{a,1}, s_0)$ with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden is the tuple $(\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$ where $T_r \subseteq (\Sigma \times \mathcal{C} \times \Sigma)$, $T_b \subseteq (\Sigma \times \mathcal{C}^{\Phi} \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least set of transitions, such that $\forall s \in \Sigma$.* [2]

1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

3 $s \xrightarrow{\theta}_1 s' \wedge \ \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{\theta} s'$ ■

Rules **1** and **2** are standard. Rule **3** says that the effect of the hiding operator over states of $\mathcal{T}$ which enable standard actions in L is to preempt all clock related transitions according to the maximal progress assumption.

Theorem 5.2 *Let $\mathcal{T}$ be a ITA. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{T}/L$ is an ITA.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption is enforced when hiding ITA.* ■

5.2.5 Equivalence of ITA

Now we will introduce a notion of strong and weak bisimulation over ITA. In particular weak bisimulation matches the clock related transitions similarly as in the static name approach of Chapter 3 and abstracts from standard τ actions similarly as in [122].

Given an ITATS $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$, strong bisimulation over states is defined by associating clock names similarly as in the static name technique of Chapter 3, so that equivalence does not depend on the particular names used for clocks. We use H to range over association histories of clock names, i.e. bijections from $\mathcal{C}$ to $\mathcal{C}$. We denote by $\mathcal{H}$ the set of all association histories.

We now present strong bisimulation for ITA which is defined by means of a family of bisimulations β_H, each indexed by an association history. First of all, let us say that a $\mathcal{H}$-indexed family of binary relations $\{\beta_H \mid H \in \mathcal{H}\}$ over Σ is *symmetric* if and only if $(s_1, s_2) \in \beta_H$ implies $(s_2, s_1) \in \beta_{\overline{H}}$, where $\overline{H} = \{(C_{n'}, C_n) \mid (C_n, C_{n'}) \in H\}$. Moreover, we use $H \leftarrow (C_n, C_{n'})$ to denote the association history H' obtained from H by adding the pair $(C_n, C_{n'})$ and removing old associations $(C_n, C_{n''})$ and $(C_{n'''}, C_{n'})$, for some $C_{n''}$ and $C_{n'''}$, already contained in H, thus preserving the structure of bijection from $\mathcal{C}$ to $\mathcal{C}$.

[2]In order to distinguish transition of $T_{r,1}$, $T_{b,1}$ and $T_{a,1}$ from transitions of T_r, T_b and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

Definition 5.5 *Let $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ be a ITATS. A symmetric $\mathcal{H}$-indexed family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ of binary relations over Σ is a strong bisimulation family iff $s_1\ \beta_H\ s_2$ implies*

- *for every $\alpha \in Act$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \xrightarrow{\alpha} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta_H\ s_2',$$

- *for every $C_n \in \mathcal{C}$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{C_n} s_1' \text{ implies } s_2 \xrightarrow{C_{n'}} s_2' \text{ for some } s_2', C_{n'} \text{ with } s_1'\ \beta_{H\leftarrow(C_n, C_{n'})}\ s_2',$$

- *for every $\phi \in \mathcal{C}^\Phi$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{\phi} s_1' \text{ implies } s_2 \xrightarrow{H(\phi)} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta_H\ s_2',$$

Two states s_1 and s_2 are strongly bisimilar with respect to association history $H \in \mathcal{H}$, denoted by $s_1 \sim_H s_2$, iff there exist some strong bisimulation family $\mathbf{B} = \{\beta_H \mid H \in \mathcal{H}\}$ such that $(s_1, s_2) \in \beta_H$. Two ITA $(\mathcal{T}_1, s_{0,1})$ and $(\mathcal{T}_2, s_{0,2})$ are strongly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are such that there exists $H \in \mathcal{H}$ such that $s_{0,1} \sim_H s_{0,2}$ in the ITATS obtained with the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$. ∎

Note that when matching clock bound transitions the association history H is used as a clock renaming function for constraints ϕ.

We now present weak bisimulation for ITA which is obtained from strong bisimulation with the standard technique of [122]. We use γ to range over transition labels, i.e. $Act \cup \mathcal{C} \cup \mathcal{C}^\Phi$. Let $\stackrel{\gamma}{\Longrightarrow}$ denote $(\xrightarrow{\tau})^* \xrightarrow{\gamma} (\xrightarrow{\tau})^*$, i.e. a sequence of transitions including a single γ transition and any number of τ transitions. Moreover we define $\stackrel{\hat{\alpha}}{\Longrightarrow}\ =\ \stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow}\ = (\xrightarrow{\tau})^*$, i.e. a possibly empty sequence of τ transitions.

Definition 5.6 *Let $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a)$ be a ITATS. A symmetric $\mathcal{H}$-indexed family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ of binary relations over Σ is a weak bisimulation family iff $s_1\ \beta_H\ s_2$ implies*

- *for every $\alpha \in Act$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta_H\ s_2',$$

- *for every $C_n \in \mathcal{C}$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{C_n} s_1' \text{ implies } s_2 \stackrel{C_{n'}}{\Longrightarrow} s_2' \text{ for some } s_2', C_{n'} \text{ with } s_1'\ \beta_{H\leftarrow(C_n, C_{n'})}\ s_2',$$

- *for every $\phi \in \mathcal{C}^\Phi$ and $s_1' \in \Sigma$,*

  $$s_1 \xrightarrow{\phi} s_1' \text{ implies } s_2 \stackrel{H(\phi)}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1'\ \beta_H\ s_2',$$

Two states s_1 and s_2 are weakly bisimilar with respect to association history $H \in \mathcal{H}$, denoted by $s_1 \approx_H s_2$, iff there exist some weak bisimulation family $\mathbf{B} = \{\beta_H \mid H \in \mathcal{H}\}$ such that $(s_1, s_2) \in \beta_H$. Two ITA $(\mathcal{T}_1, s_{0,1})$ and $(\mathcal{T}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are such that $s_{0,1} \approx_\emptyset s_{0,2}$ in the ITATS obtained with the disjoint union of $\mathcal{T}_1$ and $\mathcal{T}_2$. ∎

5.3 Interactive Prioritized Timed Transition Systems

In this section we introduce Interactive Prioritized Timed Transition Systems (IPTTSs) that will be used in the next section to define a semantics for ITA.

5.3.1 The IPTTS Model

In this section we formally introduce Interactive Prioritized Timed Transition Systems (IPTTS) which essentially include three type of transitions: *standard action transitions*, representing the interactive behavior of a system component, *prioritized transitions*, representing behaviors of the system component executed according to a certain priority level, and *numeric time transitions* representing a fixed temporal delay.

As far as standard action transitions are concerned they have exactly the same behavior as in ITA. Prioritized transitions are labeled with a certain priority level $p \in \mathbb{N}^+$ and, where transitions with a higher priority level take priority (e.g. when composing two IPTTSs in parallel) over prioritized transitions with a lower priority level. Moreover, we assume standard τ transitions to take priority over prioritized transitions, no matter which is the priority level of such transitions (due to the maximal progress assumption). Given a time domain $TD \subseteq \mathbb{R}^+$, numeric time transitions are labeled with a certain delay $t \in TD$ representing the passage of t time units. As usual in the real time literature (see e.g. [132]), several timed transition leaving a state offer the possibility to the observer to choose the amount of time after which he wants to observe the status of the system.

In IPTTS we have two different kinds of state:

- *silent states* which are exactly like in ITA.
- *non-silent states* enabling numeric timed transitions and/or prioritized transitions all with the same priority level and (possibly) visible action transitions a, only. In such states numeric timed transitions (which cause the amount of time labeling the transition to pass) and prioritized transitions are chosen by means of a non-deterministic choice. Moreover the IPTTS may potentially interact with the environment through one of its visible actions.

In the following we present the formal definition of Interactive Prioritized Timed Transition System (IPTTS), then we will define Rooted Interactive Prioritized Timed Transition Systems as IPTTSs possessing an initial state. Formally, given a time domain $TD \subseteq \mathbb{R}^+$, we use $t, t', \ldots$, representing time values, to range over TD. Moreover we use $p, p', \ldots$, representing priority levels, to range over $\mathbb{N}^+$. Finally we use θ to range over time values in TD and priorities in $\mathbb{N}^+$.

Definition 5.7 *An Interactive Prioritized Timed Transition System (IPTTS) is a tuple* $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a)$ *with*

- Σ *a set of possibly infinite states,*

- *TD a time domain, i.e. the set of possible values over which the labels of the numeric timed transitions range.*

- *Act a set of standard actions,*

- $T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$ *and* $T_t \subseteq (\Sigma \times \mathbb{R}^+ \times \Sigma)$ *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *three transition relations representing prioritized behaviors, time passage and action execution, respectively.* T_p*,* T_t *and* T_a *must be such that* $\forall s \in \Sigma$.

  - $s \xrightarrow{\tau} \implies \nexists t. s \xrightarrow{t} \wedge \nexists p. s \xrightarrow{p}$
  - $s \xrightarrow{p} \implies \nexists p' < p. s \xrightarrow{p'}$
  - $s \xrightarrow{\tau} \vee \exists t. s \xrightarrow{t} \vee \exists p. s \xrightarrow{p}$ ■

Definition 5.8 *A Rooted Interactive Prioritized Timed Transition System (RIPTTS) is a tuple* $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a, s_0)$*, where* $s_0 \in \Sigma$ *is the initial state and* $(\Sigma, TD, Act, T_p, T_t, T_a)$ *is an IPTTS.* ■

The meaning of the constraints over transition relations is the following. The first requirement says that (similarly as in ITA) if a state that can perform internal τ actions then it cannot perform time-related transitions (*maximal progress* assumption). The second requirement says that if a state can perform prioritized transitions with a certain priority level then it cannot perform prioritized transitions with a lower priority level. The third requirement says that (similarly as in ITA) we cannot have states where time is not allowed to pass (time deadlocks).

5.3.2 Parallel of Rooted IPTTSs

Now we define, similarly as for ITA, the parallel composition à la CSP of RIPTTSs.

In such a parallel composition the discrete timed transitions of the composed RIPTTSs are constrained to synchronize, so that the same amount of time passes for both systems, i.e. when time advances for one RIPTTS it must also advance for the other RIPTTS.

Definition 5.9 *The parallel composition* $\mathcal{D}_1 \|_S \mathcal{D}_2$ *of two RIPTTSs* $\mathcal{D}_1 = (\Sigma_1, TD, Act, T_{p,1}, T_{t,1}, T_{a,1}, s_{0,1})$ *and* $\mathcal{D}_2 = (\Sigma_2, TD, Act, T_{p,2}, T_{t,2}, T_{a,2}, s_{0,2})$*, with* $S \subset Act - \{\tau\}$ *being the synchronization set, is the tuple* $(\Sigma, TD, Act, T_p, T_t, T_a, (s_{0,1}, s_{0,2}))$ *with:*

- $\Sigma = \Sigma_1 \times \Sigma_2$ *the set of states*

- $T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$*,* $T_t \subseteq (\Sigma \times TD \times \Sigma)$ *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *the least transition relations, such that*

  1_l $s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$

$$\mathbf{1_r}\;\; s_2 \xrightarrow{\alpha} s_2',\; \alpha \notin S \implies (s_1,s_2) \xrightarrow{\alpha} (s_1,s_2')$$

$$\mathbf{2}\;\; s_1 \xrightarrow{a} s_1' \;\wedge\; s_2 \xrightarrow{a} s_2',\; a \in S \implies (s_1,s_2) \xrightarrow{a} (s_1',s_2')$$

$$\mathbf{3_l}\;\; s_1 \xrightarrow{p} s_1' \;\wedge\; s_2 \stackrel{\tau}{\not\longrightarrow} \;\wedge\; \nexists p' > p. s_2 \xrightarrow{p'} \implies (s_1,s_2) \xrightarrow{p} (s_1',s_2)$$

$$\mathbf{3_r}\;\; s_2 \xrightarrow{p} s_2' \;\wedge\; s_1 \stackrel{\tau}{\not\longrightarrow} \;\wedge\; \nexists p' > p. s_1 \xrightarrow{p'} \implies (s_1,s_2) \xrightarrow{p} (s_1,s_2')$$

$$\mathbf{4}\;\; s_1 \xrightarrow{t} s_1' \;\wedge\; s_2 \xrightarrow{t} s_2' \implies (s_1,s_2) \xrightarrow{t} (s_1',s_2')$$

- $(s_{0,1}, s_{0,2}) \in \Sigma$ *the initial state.* ∎

When evaluating action transitions we just make use of standard rules. Prioritized transitions are determined by taking into account priorities according to a "global" notion of priority (see the classification made in [62]) where priorities are applied across the parallel operator. Finally timed transitions are evaluated by just requiring them to synchronize.

Theorem 5.3 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two RIPTTSs. Then for each $S \subseteq Act - \{\tau\}$, $\mathcal{D}_1 \|_S \mathcal{D}_2$ is a RIPTTS.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption and the priorities among prioritized transitions are enforced when composing RIPTTSs and that timed transitions must synchronize in order to be executed.* ∎

5.3.3 Hiding of Rooted IPTTSs

Now we define, similarly as for ITA, the hiding of RIPTTSs.

Definition 5.10 *The hiding $\mathcal{D}/L$ of a RIPTTS $\mathcal{D}_1 = (\Sigma, TD, Act, P_1, T_{p,1}, T_{t,1}, T_{a,1}, s_0)$, with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden, is the tuple $(\Sigma, TD, Act, P, T_p, T_t, T_a, s_0)$, with:*

- *P the partial function obtained from P_1 by removing from its domain those states (and the associated probability spaces) which enable at least one transition labeled with an action in L*
- *$T_p \subseteq (\Sigma \times \mathbb{N}^+ \times \Sigma)$, $T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that $\forall s \in \Sigma$.* [3]

$$\mathbf{1}\;\; s \xrightarrow{\alpha}_1 s',\; \alpha \notin L \implies s \xrightarrow{\alpha} s'$$

$$\mathbf{2}\;\; s \xrightarrow{a}_1 s',\; a \in L \implies s \xrightarrow{\tau} s'$$

$$\mathbf{3}\;\; s \xrightarrow{\theta}_1 \wedge\; \nexists a \in L. s \xrightarrow{a}_1 \implies s \xrightarrow{\theta}$$

∎

[3]In order to distinguish transition of $T_{p,1}$, $T_{t,1}$ and $T_{a,1}$ from transitions of T_p, T_t and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

Similarly as for ITA, in the definition of the hiding operator in addition to standard rules we make use of rules which enforce the maximal progress assumption.

Theorem 5.4 *Let $\mathcal{D}$ be a RIPTTS. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{D}/L$ is a RIPTTS.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption is enforced when hiding RIPTTSs.* ■

5.3.4 Equivalence of Rooted IPTTSs

Now we introduce a notion of weak bisimulation for RIPTTSs which matches prioritized and timed transitions according to strong bisimulation and abstracts from standard τ actions similarly as in [122].

Definition 5.11 *Let $\mathcal{D} = (\Sigma, TD, Act, T_p, T_t, T_a)$ be an IPTTS. An equivalence relation β on Σ is a weak bisimulation iff $s_1 \ \beta \ s_2$ implies*

- *for every $\alpha \in Act$,*

  $$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \xRightarrow{\hat{\alpha}} s_2' \text{ for some } s_2' \text{ with } s_1' \ \beta \ s_2',$$

- *for every $\theta \in \mathbb{N}^+ \cup TD$,*

  $$s_1 \xrightarrow{\theta} s_1' \text{ implies } s_2 \xrightarrow{\theta} s_2' \text{ for some } s_2' \text{ with } s_1' \ \beta \ s_2',$$

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. Two RIPTTSs $(\mathcal{D}_1, s_{0,1})$ and $(\mathcal{D}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are weakly bisimilar in the IPTTS obtained with the disjoint union of $\mathcal{D}_1$ and $\mathcal{D}_2$. ■

5.4 A Semantics for Interactive Timed Automata

In this section we present a semantics for interactive timed automata which maps them onto interactive prioritized timed transition systems. Such a semantics explicitely represents the passage of time by means of transitions labeled with numeric time delays and turns clock reset transitions into prioritized transitions with priority level 2 and clock bound transitions into prioritized transitions with priority level 1.

We now formally define the semantics of an ITA.

Definition 5.12 *The semantics of an ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$ is the RIPTTS $[\![\mathcal{T}]\!] = (\Sigma', \mathbb{R}^+, Act, T_p, T_t, T_a', s_0')$ where:*

- *$\Sigma' = (\Sigma \times Spent)$ is the set of states of the RIPTTS, where Spent, ranged over by v, is the set of functions from $\mathcal{C}$ to $\mathbb{R}^+ \cup \{0\}$, expressing the time already spent in execution by the clocks of the ITA from the last reset event*
- *$\mathbb{R}^+$ is the time domain: we consider continuous time.*

(**P1**) $\dfrac{s \xrightarrow{\phi} s' \ \wedge \ v \vdash \phi}{\langle s, v\rangle \xrightarrow{1} \langle s', v\rangle}$	(**P2**) $\dfrac{s \xrightarrow{C_n} s'}{\langle s, v\rangle \xrightarrow{2} \langle s', v \leftarrow (C_n, 0)\,\rangle}$
(**T**) $\dfrac{\nexists C_n : s \xrightarrow{C_n} \ \wedge \ s \not\xrightarrow{\tau} \ \wedge \ \bigwedge\{\ \exists t' \geq t : v + t' \vdash \phi \ \mid \ s \xrightarrow{\phi} \}}{\langle s, v\rangle \xrightarrow{t} \langle s, v + t\rangle}$	
(**A**) $\dfrac{s \xrightarrow{\alpha} s'}{\langle s, v\rangle \xrightarrow{\alpha} \langle s', v\rangle}$	

Table 5.1: Semantic rules for ITA

- *Act is the set of standard actions considered in the ITA.*
- T_p *is the set of prioritized transitions which are defined as the least relation over* $\Sigma' \times \mathbb{N}^+ \times \Sigma'$ *satisfying the operational rules in the first part of Table 5.1.*
- T_t *is the set of timed transitions which are defined as the least relation over* $\Sigma' \times \mathbb{R}^+ \times \Sigma'$ *satisfying the operational rules in the second part of Table 5.1.*
- T_a' *is the set of action transitions which are defined as the least relation over* $\Sigma' \times Act \times \Sigma'$ *satisfying the operational rules in the third part of Table 5.1.*
- $s_0' = \langle s_0, \mathbf{0}\rangle$, *with* $\mathbf{0} = \{(C_n, 0) \mid C_n \in \mathcal{C}\}$ *is the initial state of the RIPTTS, where the ITA is in the initial state and all clocks start from zero.* ■

In Table 5.1 we make use of the following notation. $v \vdash \phi$ holds true if and only if the formula obtained from ϕ by replacing clocks with time values according to v is true. We assume, as expected, "$\bigwedge \emptyset$" to yeld true. Moreover we define $v \leftarrow (C_n, t)$ to be the function obtained from v by replacing the pair (C_n, t') already contained in v with the new pair (C_n, t). Finally, we define $v + t$, with $t \in \mathbb{R}^+$, to be the function obtained from v by adding t to the time value associated with each clock in v.

Theorem 5.5 *Let* $\mathcal{T}'$, $\mathcal{T}''$ *be two ITA. If* $\mathcal{T}' \approx \mathcal{T}''$ *then* $[\![\mathcal{T}']\!] \approx [\![\mathcal{T}'']\!]$.

Proof *It is just sufficient to show that from a weak bisimulation family* $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ *between the two ITA* $\mathcal{T}'$ *and* $\mathcal{T}''$ *we can derive a weak bisimulation* β *between the two RIPTTSs* $[\![\mathcal{T}']\!]$ *and*

$[\![\mathcal{T}'']\!]$. *In particular* β *is obtained by letting* $(s_1, v_1)\ \beta'\ (s_2, v_2)$ *whenever* $s_1 \beta_H s_2$ *and* $v_1, v_2 \in Spent$ *are such that for each* $(C_{n'}, C_{n''}) \in H$ *we have* $v_1(C_{n'}) = v_2(C_{n''})$. *The fact that* β *is a weak bisimulation simply derives from the fact that in clock bound transitions leaving equivalent ITA states we must have the same constraints for clocks that are matched by the equivalence.* ■

The following theorems show that the semantics of ITA is indeed compositional.

Theorem 5.6 *Let* $\mathcal{T}'$, $\mathcal{T}''$ *be two ITA. For each* $S \subseteq Act - \{\tau\}$ *we have* $[\![\mathcal{T}']\!] \,\|_S [\![\mathcal{T}'']\!] \approx [\![\mathcal{T}' \,\|_S\, \mathcal{T}'']\!]$.

Proof *It is just sufficient to show that we can build a weak bisimulation* β *between the two RIPTTSs* $[\![\mathcal{T}']\!] \,\|_S [\![\mathcal{T}'']\!]$ *and* $[\![\mathcal{T}' \,\|_S\, \mathcal{T}'']\!]$. *In particular* β *is obtained by letting* $((s', v'), (s'', v''))\ \beta\ ((s', s''), v)$ *whenever the spent lifetimes of clocks in* v' *and* v'' *are the same as those in* v *once clocks renaming performed in parallel composition is taken into account. The fact that* β' *is a weak bisimulation derives from the fact that the set of clock bound transitions executable in a state of an ITA obtained as the parallel composition of two ITA is the union of the clock bound transitions executable by the two ITA. Therefore the set of time transitions performable in such a state of the composed model turns out to be equivalent to the synchronization of the time transitions independently generated from the states of the two ITA. Moreover when composing in parallel ITA* τ *transitions pre-empt timed-related transitions and clock reset transitions pre-empt clock bound transitions just as in IPTTSs* τ *transitions pre-empt prioritized (and timed) transitions and transitions with priority level* 2 *pre-empt transitions with priority level* 1. ■

Theorem 5.7 *Let* $\mathcal{T}$ *be an ITA. For each* $L \subseteq Act - \{\tau\}$ *we have* $[\![\mathcal{T}]\!]/L \approx [\![\mathcal{T}/L]\!]$.

Proof *It is just sufficient to show that we can build a weak bisimulation* β *between the two RIPTTSs* $[\![\mathcal{T}]\!]/L$ *and* $[\![\mathcal{T}/L]\!]$. *In the case of the hiding operator the structure of the states of the two RIPTTSs turns out to be the same and* β *is just obtained by considering the identity relation, i.e. we relate each state* (s, v) *to itself. The fact that* β' *is a weak bisimulation simply derives from the fact that the only effect of hiding standard actions is to preempt the execution of time-related transitions both in ITA and in RIPTTSs.* ■

Chapter 6

Interactive Generalized Semi-Markov Processes

In this chapter we introduce Interactive Generalized Semi-Markov Processes (IGSMP), an extension of Generalised Semi-Markov Processes (GSMPs), i.e. probabilistic timed systems where durations of delays are expressed by means of random variables with a general probability distribution, with action transitions representing the ability of a process to interact with another process. Such an extension causes the problem of adequately handling non-deterministic choices in Generalised Semi-Markov Processes (GSMPs) to arise. In particular we define IGSMPs so that the probabilistic duration of a delay is not decided all at once when the delay starts as done in [66], but step by step in each system state (in the theory of GSMPs this corresponds to recording spent lifetimes instead of residual lifetimes of delays). In this way an adversary cannot take decisions a priori, based on the knowledge he may get about the future behavior of the system. Moreover we define well-named IGSMPs, a canonical form for names of clocks used in an IGSMP which is based on the dynamic approach presented in Chapter 3 and makes it possible to define equivalence over IGSMPs as a simple probabilistic extension of standard weak bisimulation [122]. More precisely, in this chapter we introduce IGSMPs and a formal procedure for deriving performance models (GSMPs) from complete IGSMP specifications. Moreover, we formalize the class of well-named IGSMP models and the class of Interactive Stochastic Timed Transition Systems (ISTTSs) which are both closed under CSP parallel composition and hiding. Based on that, we introduce a semantics for IGSMPs which maps well-named IGSMP models onto ISTTSs by recording spent lifetimes of delays. Finally, we show that two weakly bisimilar IGSMPs give rise to two weakly bisimilar semantic models and that our semantic mapping is compositional with respect to both CSP parallel composition and hiding.

6.1 Introduction

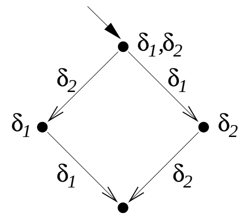
Introducing a modeling paradigm for timed concurrent systems that can express probability distributions of general nature is important for two main reasons. First, it allows us to overcome the strong limitations in expressiveness inherent in purely Markovian specification paradigms (see Chapter 4) where even fixed (non probabilistic) durations cannot be represented. Second, it leads to an integration of stochastic time analysis with real-time analysis as explained in Chapter 1. The price to pay is a more complex representation of system behavior and much more involved system analysis techniques.

6.1.1 The Basic Idea

Some previous efforts have been made in order to develop models for general distributions [87, 7, 49, 139]. With respect to such approaches, which somehow face this problem by starting from process algebra semantical viewpoint, in [41] we have introduced the idea that a specification paradigm expressing systems with generally distributed delays should originate from probabilistic models which are well-founded from the viewpoint of probability theory.

As we have explained in Chapter 2, the stochastic processes mainly studied in the literature for performance evaluation purposes are in increasing order of expressivity: continuous time Markov chains (CTMCs), semi-Markov processes (SMPs), and generalized semi-Markov processes (GSMPs). The difference among them lies in the set of instants of process life which satisfy the Markov property, i.e. those instants such that the future behavior of the stochastic process depends only on the current state of the process and not on its past behavior. For CTMCs the Markov property holds in every instant of process life, for SMPs it holds only in the instants of state change, and for GSMPs it never holds, but can be retrieved through a different representation of process states (each state is turned into a continuous infinity of states) by the standard technique of [65]. Since CTMCs can represent only activities with an exponentially distributed duration (only this distribution has the required memoryless property), the only candidates for representing systems with generally distributed durations are SMPs and GSMPs, and we now show that GSMPs are actually needed for our purposes. Consider the example of two delays δ_1 and δ_2 executed in parallel, the former with a deterministic duration 5 and the latter with a Gaussian duration with mean 0.5 and variance 1. This situation can be represented as in Fig. 6.1. In Fig. 6.1 each state is labeled with the set of delays which are in execution during the period of time the system sojourns in the state. In the beginning both delays are in contemporaneous execution and the system sojourns in the first state until one delay terminates. When this happens the system performs the transition labeled with the terminated action. Suppose that δ_1 terminates before δ_2 and the system reaches the state labeled with δ_2. In this state the delay δ_2 continues its execution until it terminates. As a consequence the sojourn time of the system in the state labeled with δ_2 (which is given by the residual distribution of delay δ_2) is not determined simply by the fact that the system is in this state, but depends on the time δ_2 has already spent in execution in the first state. In particular since we can no longer rely on the memoryless property (which is enjoyed only by exponential distributions) the residual distribution of δ_2 is different for every value of time spent. Therefore the process is not Markovian even in the instant when this state is entered.

This example shows that even the simple case of two parallel delays with generally distributed durations cannot be represented by an SMP. The process of the example is, instead, a GSMP (δ_1 and δ_2 are its elements



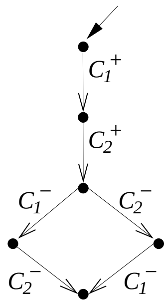
$$PDF(\delta_1) = Det(5), \qquad PDF(\delta_2) = Gauss(0.5, 1)$$

Figure 6.1: Parallel Execution of Generally Distributed Delays

and state labels in Fig. 6.1 denote active elements). This can be seen as follows. If we imagine to give a different representation of the process where we replace the state labeled with δ_2 with infinitely many states each denoting a different spent lifetime for the delay δ_2, we can retrieve the Markov property. The sojourn time in each of the newly derived states would then be determined by the state itself (it would be given by the distribution of delay δ_2 conditioned on a particular value for the spent lifetime) and not by the previous behavior of the process. Another way to retrieve the Markov property would be to consider residual lifetimes instead of spent lifetimes (see Chapter 2).

Despite of the fact that in GSMPs the Markov property in instants of state change can be retrieved via an infinite state representation, when we consider the initial finite state system description (as that of Fig. 6.1) we have that the system behavior in a certain state indeed depends on what happened previously. In particular, delays executed in a state cannot be simply be considered as starting in that state as for CTMCs (or SMPs), but the may continue their execution from previous states. In other words a delay can no longer be considered as being executed *atomically* in a single transition, but we have to represent delays that start in a certain state, evolve through several states, and terminate in another state (in the previous example both δ_1 and δ_2 start in the first state and may terminate in another state). As a consequence the execution of a generally distributed delay must be characterized in models by the *two events* of *delay start* and *delay termination*.

In [87, 139] the Markovian representation of system behavior via delays "atomically" executed in transitions (which are labeled by the duration distribution of such delays) is adapted to systems with general distributions by adding to such transitions some information uniquely determining (through e.g. a pointer) the state where the delay labeling the transition is actually started. On the contrary, we prefer to stick to the notation of GSMPs, where representing the execution of generally distributed delays is simply done by: uniquely *identifying* each delay with a different *element* (similar to a clock of a timed automata), associating to each *element* the duration distribution of the delay it represents as in Fig. 6.1, assuming that element execution continues when going from state to state, and making explicit events of element start (or clock reset in timed automata). In this way we have a system representation which can be easily transformed into a GSMP (see Sect. 6.3) or a timed automata (see Chapter 8) for analysis purposes.



$$PDF(C_1) = Det(5), \quad PDF(C_2) = Gauss(0.5, 1)$$

Figure 6.2: Event-Based Representation of Parallel Execution of Delays

In particular we represent temporal delays by clocks and we explicitely represent both basic events of start and termination of a clock C explicitely by means of a clock start transition C^+ and a clock termination transition C^-. On the other hand we do not explicitely indicate the set of clocks which are being executed in a state (the active elements in states of GSMPs represented in Fig. 6.1 as state labels) since such information can be easily inferred from its outgoing clock termination transitions. The resulting event-based representation of the system of Fig. 6.1 is depicted in Fig. 6.2, where delays δ_1 and δ_2 are represented by clocks C_1 and C_2, respectively, and we initially start both clocks by means of two explicit clock start transitions executed in zero time. To be precise, in our model we will consider clock start transitions labeled with pairs $\langle C^+, w \rangle$, where w is a *weight* [150] used for representing probabilistic choices. In this way we reach the same expressivity as GSMPs.

6.1.2 A Model for Timed Concurrent Systems with General Distributions

By following the "interactive" idea of [99], an IGSMP represents the behavior of a component by employing both *standard action transitions*, representing the interactive behavior of the component, and *clock start transitions* and *clock termination transitions*, representing the timed probabilistic behavior of the component. Action transitions are just standard CCS/CSP transitions: when several action transitions are enabled in an IGSMP state, the choice among them is just performed non-deterministically and when IGSMPs are composed in parallel they synchronize following the CSP [109] approach, where the actions belonging to a given set S are required to synchronize. Clock start transitions are labeled with a clock name and a weight and represent the event of starting of a temporal delay whose probabilistic duration is given by the distribution associated with the clock. When several clock start transitions are enabled in an IGSMP state, the choice between them is performed probabilistically according to the weights of the transitions. Clock termination transitions are labeled with a clock name and represent the event of termination of the corresponding temporal delay. A system stays in a state enabling several termination transitions until one of the temporal delays currently in execution terminates and the corresponding transition is performed.

Besides IGSMPs, we also define *well-named IGSMPs*, a canonical form for names of clocks used in an IGSMP which is based on the dynamic approach presented in Chapter 3. In particular, similarly as in Chapter 3 where the name associated with the two events of action start and action termination generated by the execution of an action is a pair consisting of the type a of the action and an index $i \in \mathbb{N}$, in a well-named IGSMP names n of clocks C_n consist of pairs "f, i", where f is the probability distribution associated with the clock and i an index. For each different distribution f, the index to be used when a new clock with distribution f starts is determined by following the same rule as for the indexes used in start events of actions of the same type in the dynamic approach of Chapter 3. The use of well-named IGSMPs makes it possible to define equivalence as a simple probabilistic extension of standard weak bisimulation [122] without having to associate clock names in the definition of equivalence as done for ITA in Chapter 5. When considering IGSMPs the introduction of such a canonical form for names of clocks is important because, with respect to ITA, the presence of probabilistic choices in IGSMPs would cause a notion of equivalence which associates clock names like that used for ITA to become rather complex (see Chapter 13 where we will define an equivalence of this kind).

As for IWMCs and ITA, we define CSP parallel composition and hiding of well-named IGSMPs. In order to obtain well-named IGSMPs as the result of such operations we have to suitably rename clocks by following the technique introduced in the dynamic approach of Chapter 3.

IGSMPs, similarly as IWMCs and ITA, may include both internal and external non-deterministic choices. While external non-deterministic choices may be resolved through synchronization with other system components (they are not present in complete system specifications), internal non-determinism represents an inherent underspecification of the system performance. Therefore adversaries (or schedulers) play an important role in the performance analysis of IGSMPs in that they allow internal non-determinism to be removed from an IGSMP thus turning it into a GSMP. In this chapter we will present a formal procedure for transforming the semantic model obtained from a complete IGSMP system specification which does not include internal non-determinism into a *GSMP*. Such a procedure just turns each delay of the system into a different element of the *GSMP* and *IGSMP* weighted choices into probabilistic choices of a *GSMP*.

6.1.3 Dealing with Non-Determinism in Systems with General Distributions

Introducing non-determinism in probabilistic systems with general distributions causes new problems to arise with respect to the classical theory of GSMPs. Such problems arise when we consider the interplay of non-deterministic choices and the probabilistic behavior of clocks when IGSMPs are actually executed. Following the classical approach of discrete event simulation (see e.g. [54]), in the instant a clock starts, the clock is set to a temporal value sampled from its duration distribution. As time passes the clock counts down and it terminates when it reaches value zero. From a technical viewpoint this means that, while the GSMP proceeds from state to state, we keep track of the quantity of time that clocks must still spend in execution (the *residual lifetimes* of the clocks). This approach to the execution of an IGSMP, which has been previously applied in [66] to systems including non-determinism and generally distributed time, has the drawback that an adversary can base its decisions (concerning non-deterministic choices) on the knowledge obtained a priori about the future behavior of the system, e.g. the information about the quantity of time that a delay will spend in execution.

In this chapter we consider a new alternative approach to the execution of systems including non-determinism and generally distributed time which adequately handles non-deterministic choices. The idea is that we want the probabilistic duration of a generally distributed delay not to be decided all at once when the delay starts, but step by step in each system state. More precisely, this is realized by keeping track of the quantity of time spent by clocks in execution (*spent lifetimes* of clocks), and by evaluating, when a new IGSMP state is entered, the distribution of the residual duration of the clock from (i) the duration distribution associated with the clock, and (ii) the time it has already spent in execution. Such an approach, which is based on recording spent lifetimes instead of residual lifetimes, is adherent to the classical behavior of Timed Automata [132] where clocks are increased (and not decreased) while time passes. Besides it indeed solves the problem of executing a system with non-deterministic choices because, since the residual duration of clocks is sampled in every state traversed by the IGSMP, the adversary cannot gain a priori knowledge on the system behavior. Finally, considering spent lifetimes instead of residual lifetimes is correct also from a probabilistic viewpoint, because in probability theory the two approaches are both valid alternative ways to interpret a GSMP [65]. It is worth noting that the choice of adopting this alternative approach for representing the execution of an IGSMP is conceptual and not at all related with the technical differences between the formalism considered in [66] and IGSMPs. We could apply the technique used in [66] to IGSMPs as well.

Similarly as in [66], based on our approach to the execution of an IGSMP, we produce a semantics for IGSMPs which maps an IGSMP onto a transition system where: (i) the passage of time is explicitely represented by transitions labeled with numeric time delays and (ii) duration probability distributions are turned into infinitely branching probabilistic choices which lead to states performing numeric time delays with different durations. Differently from [66], we express semantic models of IGSMPs by means of "interactive" probabilistic timed transition systems which can be themselves composed and for which we define a notion of weak bisimulation. This allows us to develop a semantic mapping which is compositional with respect to parallel composition and hiding.

6.1.4 Outline of the Chapter

We start (in Sect. 6.2) by formalising the model of IGSMPs and the model of well-named IGSMPs. Moreover we show that the class of well-named IGSMPs is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over well-named IGSMPs. Then, (in Sect. 6.3) we present the formal procedure for deriving a performance model in the form of a *GSMP* from a complete IGSMP system specification. Moreover, (in Sect. 6.4) we introduce the model of Interactive Stochastic Timed Transition Systems (ISTTSs) which include: *standard action transitions*, representing the interactive behavior of a system component, *numeric time transitions* representing a fixed temporal delay, and *probabilistic transitions* (expressed by means of probability spaces) representing (infinitely branching) probabilistic choices. Moreover we show that the class of ISTTSs is closed with respect to CSP parallel composition and hiding and we introduce a notion of weak bisimulation over ISTTSs. Furthermore, (in Sect. 6.5) we present the semantics for IGSMPs which maps IGSMPs onto ISTTSs and we show that weakly bisimilar IGSMPs give rise to weakly bisimilar semantic models and that the semantic mapping is compositional with respect to both CSP parallel composition and hiding. Finally, (in Sect. 6.6) we report some concluding remarks.

6.2 Definition of Interactive Generalized Semi-Markov Process

In this section we will present the model of Interactive Generalized Semi-Markov Processes (IGSMPs) and of well-named interactive generalized semi-Markov processes: a canonical form for IGSMPs which introduces some constraints on clock names and makes it simple to establish equivalence over IGSMPs.

6.2.1 The IGSMP Model

The model of interactive generalized semi-Markov processes extends that of generalized semi-Markov processes (see Sect. 2.2.6) by expressing in addition to GSMP clocks (or elements) execution, also the execution of standard actions which can synchronize and have a zero duration. As far as probabilistic delays are concerned, they are modeled as in GSMPs by means of clocks C whose duration is expressed through general probability distributions. In the following we will distinguish different clocks used in an IGSMP through "names", where C_n denotes the clock with name n. In an IGSMP the execution of a clock C_n is represented by means of two events: the event of clock start C_n^+ followed by the relative event of clock termination C_n^-. Therefore in an IGSMP we have three types of transitions: standard action transitions representing action execution, clock start transitions representing events C_n^+ and clock termination transitions representing events C_n^-. When a transition C_n^+ is performed by the IGSMP the clock C_n starts and continues its execution in every state traversed by the IGSMP. Whenever the clock C_n terminates, then the IGSMP executes the corresponding termination transition C_n^-. In particular, since, as in GSMPs, each started clock C_n which has not terminated yet must continue its execution in each state traversed by the IGSMP, all such states must have an outgoing transition C_n^-. Obviously clocks which can be simultaneously under execution in an IGSMP state must have different names (even if they have the same duration distribution), so that the event of termination of a clock C_n^- is always uniquely related to the corresponding event of start of the same clock C_n^+. Similarly as GSMPs, IGSMPs can also express probabilistic choices. This is obtained by associating with each start transition C_n^+ a weight $w \in \mathbb{R}^+$. In this way when a state of the IGSMP enables several clock start transitions $\langle C_n^+, w\rangle$, the choice of the clock to be started is performed probabilistically according to the weights w of the transitions. For instance, a state enabling two transitions labeled with $\langle C_n^+, w\rangle$ and $\langle C_{n'}^+, w'\rangle$ respectively starts clock C_n with probability $w/(w+w')$ and starts clock $C_{n'}$ with probability

$w'/(w + w')$. On the other hand, IGSMPs also have, in addition to GSMPs, the capability of expressing non-deterministic choices. This because, as in standard labeled transition systems deriving from CCS/CSP terms, in the states of an IGSMP action transitions are just non-deterministically chosen. As in IWMCs and ITA, alternative transitions labeled with invisible τ actions represent internal non-deterministic choices, while alternative transitions labeled with visible actions a (which are seen as incomplete potential transitions which wait for a synchronization with other system components) represent external non-deterministic choices which depend on the environment. An IGSMP represents a *complete system* only when it does not include any transition labeled by a visible action. Visible actions are synchronized via a CSP synchronization policy and a hiding operator is used to turn (synchronized) visible actions into complete invisible actions. This approach differs from that of the stochastic automaton model of [66], where two different kinds of semantics have to be defined in order to describe the *actual behavior* of closed systems and the *potential behavior* of open systems. In our approach both the potential and the actual behavior of the system are represented within the same model and complete systems are obtained by hiding all the actions of the model.

More precisely, in an IGSMP we have four different kinds of state:

- *silent states*, enabling invisible action transitions τ and (possibly) visible action transitions a only. In such states the IGSMP just performs a non-deterministic choice among the τ transitions in zero time and may potentially interact with the environment through one of the visible actions (see e.g. Fig. 6.3.a).

- *probabilistic states*, enabling $\langle C_n^+, w\rangle$ transitions and (possibly) visible action transitions a only. In such states the IGSMP just performs a probabilistic choice among the clock start transitions in zero time and may potentially interact with the environment through one of the visible actions (see e.g. Fig. 6.3.b).

- *timed states*, enabling C_n^- transitions and (possibly) visible action transitions a only. In such states the IGSMP executes all the clocks labeling the outgoing termination transitions according to their residual duration distribution. The clock that terminates first determines the transition to be performed. Note that since, as in GSMPs, we assume that clocks cannot terminate at the same instant, we always have a unique clock terminating before the other ones (see e.g. Fig. 6.3.c). While the IGSMP sojourns in the state, it may (at any time) potentially interact with the environment through one of the outgoing visible action transitions.

- *waiting states*, enabling standard visible actions only or no transition at all. In such states the IGSMP sojourns indefinitely. It may, at any time, potentially interact with the environment through one of the outgoing visible action transitions (see e.g. Fig. 6.3.d).

In the following we present the formal definition of Interactive Generalized Semi-Markovian Transition System (IGSMTS), then we will define interactive generalized semi-Markov processes as IGSMTSs possessing

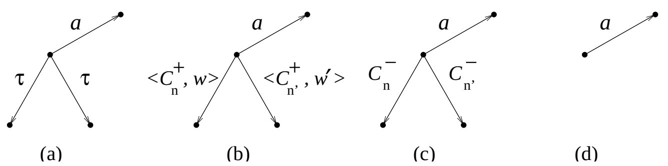


Figure 6.3: Some examples of possible states of an IGSMP

an initial state. Formally, we denote with PDF the set of probability distribution functions over $\mathbb{R}$ ranged over by $f, g, \ldots$ and with PDF^+ the set of probability distribution functions over $\mathbb{R}$ such that $f(x) = 0$ for $x < 0$ (representing duration distributions). Weights, belonging to $\mathbb{R}^+$, are ranged over by $w, w', \ldots$. Moreover, we denote the set of standard action types used in a IGSMTS by Act, ranged over by $\alpha, \alpha', \ldots$. As usual Act includes the special type τ denoting internal actions. The set $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. The set of clocks of an IGSMTS is denoted by $\mathcal{C} = \{C_n \mid n \in \mathcal{C}Names\}$, where $\mathcal{C}Names$ is a set of clock names. Given a set $\mathcal{C}$, we denote with $\mathcal{C}^+ = \{\langle C_n^+, w\rangle \mid C_n \in \mathcal{C}, w \in \mathbb{R}^+\}$ the set of events denoting the starting of a clock and $\mathcal{C}^- = \{C_n^- \mid C_n \in \mathcal{C}\}$ the set of events denoting the termination of a clock. Let $\mathcal{C}^+ \cup \mathcal{C}^-$ be ranged over by $\theta, \theta', \ldots$. The set of states of an IGSMTS is denoted by Σ, ranged over by $s, s', \ldots$. We assume the following abbreviations that will make the definition of IGSMTSs easier. Let us suppose that $T \subseteq (\Sigma \times \mathit{Labels} \times \Sigma)$ is a transition relation, where Labels is a set of transition labels, ranged over by l. We use $s \xrightarrow{l} s'$ to stand for $(s, l, s') \in T$, $s \xrightarrow{l}$ to stand for $\exists s' : s \xrightarrow{l} s'$, and $s \not\xrightarrow{l}$ to stand for $\not\exists s' : s \xrightarrow{l} s'$.

Definition 6.1 *An Interactive Generalized Semi-Markovian Transition System (IGSMTS) is a tuple* $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a)$ *with*

- Σ *a set of states,*
- $\mathcal{C}$ *a set of clocks,*
- $D : \mathcal{C} \longrightarrow PDF^+$ *a function that assigns a duration probability distribution function to each clock,*
- Act *a set of standard actions,*
- $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *three transition relations representing clock start and termination events and action execution, respectively, such that:* [1]

  1 $\forall s \in \Sigma.$
  $s \xrightarrow{\tau} \implies \not\exists \theta.\, s \xrightarrow{\theta}$

  2 $\forall s \in \Sigma.$
  $\exists C_n, w.\, s \xrightarrow{\langle C_n^+, w\rangle} \implies \not\exists C_{n'}.\, s \xrightarrow{C_{n'}^-}$

[1]For the sake of readability here and in the rest of the chapter we assume the following operator precedence when writing constraints for transition relations: existential quantifier > "and" operator > implication.

3 $\exists \mathcal{S} : \Sigma \longrightarrow \mathcal{P}(\mathcal{C})$ *the* active clock function*, such that* $\forall s \in \Sigma$.

a) - $s \xrightarrow{\alpha} s' \implies \mathcal{S}(s') = \mathcal{S}(s)$

- $s \xrightarrow{\langle C_n^+, w\rangle} s' \implies \mathcal{S}(s') = \mathcal{S}(s) \cup \{C_n\}$

- $s \xrightarrow{C_n^-} s' \implies C_n \in \mathcal{S}(s) \;\wedge\; \mathcal{S}(s') = \mathcal{S}(s) - \{C_n\}$

b) $\exists C_n, w.\, s \xrightarrow{\langle C_n^+, w\rangle} \implies C_n \notin \mathcal{S}(s)$

c) $C_n \in \mathcal{S}(s) \wedge s \stackrel{\tau}{\not\longrightarrow} \wedge \;\nexists C_{n'}, w.\, s \xrightarrow{\langle C_{n'}^+, w\rangle} \implies s \xrightarrow{C_n^-}$

4 $\forall s \in \Sigma$.

$s \xrightarrow{\langle C_n^+, w\rangle} s' \implies act(s') \subseteq act(s)$

where the enabled action function $act : \Sigma \longrightarrow \mathcal{P}(Act)$ *is defined by* $act(s) = \{\alpha \mid s \xrightarrow{\alpha}\}$. ∎

Definition 6.2 *An Interactive Generalized Semi-Markov Process (IGSMP) is a tuple* $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$*, where* $s_0 \in \Sigma$ *is the initial state of the IGSMP and* $(\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a)$ *is an IGSMTS such that function* $\mathcal{S}$ *in item* 3 *of Definition 6.1 also satisfies* $\mathcal{S}(s_0) = \emptyset$. ∎

The constraints over transition relations T_+, T_- and T_a guarantee that each state of the IGSMP belongs to one of the four kind of states above. In particular, the first requirement says that if a state can perform internal τ actions then it cannot perform events of clock starts or clock terminations. Such a property derives from the assumption of *maximal progress*: the possibility of performing internal actions prevents the execution of delays. The second requirement says that if a state can perform clock start events then it cannot perform clock termination events. Such a property derives from the assumption of *urgency of delays*: clock start events cannot be delayed but must be performed immediately, hence they prevent the execution of clock termination transitions. The third requirement checks that clock starting and termination transitions are consistent with the set of clocks that should be in execution in each state of the IGSMP. This is done by defining a function $\mathcal{S}$ which maps each state onto the expected set of clocks in execution, i.e. the set of clocks which have started but not terminated yet. In particular, in the initial state s_0 such a set is empty. The constraint *a)* defines the construction rule of the active clock set for each state reachable from s_0. In the case of a transition from a state s to a state s' labeled with an standard action, the active clocks of s' stem from the active clocks of s, as no clock can be terminated given that a standard action has been performed. If a transition from s to s' is labeled with a clock start event $\langle C_n^+, w\rangle$, then s' inherits the active clock set of s and adds to this set the started clock C_n. Finally, in the case of a transition from s to s' labeled with a clock termination event C_n^-, s' inherits the active clock set of s without such a terminated clock C_n. Constraints *b)* and *c)* concern the legality of the outgoing transitions of a state. In particular, the former says that the name of a clock labeling a starting transition must be fresh (i.e. no clock with such a name must be currently in execution). The latter says that each state without τ and $\langle C_n^+, w\rangle$ outgoing transitions must have a $C_{n'}^-$ outgoing transition for each active clock $C_{n'}$. This definition preserves both the maximal progress and the urgency of delays assumptions and, in each state where it is possible, guarantees the possibility of terminating each delay that is still active. The fourth requirement of Definition 6.1 implements the following constraint over the structure of IGSMPs which makes their theory simpler. The unique role of clock start transitions in an IGSMP must be to lead to a timed state where the started clocks are actually executed, hence the execution of such transitions cannot cause new behaviors to be performable by the IGSMP. As we will see in Chapter 7, such a constraint is satisfied by the semantic models of terms of the calculus of IGSMPs, hence we consider this constraint not to be really restrictive for our purposes. Formally, we require that the set of action transitions enabled after a clock start transition is a subset of (or equal to) the set of action transitions enabled before such a transition. This guarantees that no new behaviors can be introduced by clock start transitions because: (i) no new behavior beginning with a τ transition can be executable after a clock start transition (states enabling clock start transitions cannot enable τ transitions), and (ii) every potential behavior beginning with a transition a executable after a clock start transition can never be actually executed by hiding a, because before the clock start transition there is a potential behavior beginning with the same action a, which, when hidden, preempts the clock start (see the following Sect. 6.2.4 about the hiding of IGSMPs).

6.2.2 The Well-Named IGSMP Model

The model of well-named IGSMPs represents a canonical form for names of clocks used in an IGSMP which is based on the dynamic approach presented in Chapter 3 and makes it simple to develop an equivalence notion over IGSMPs which matches clocks with the same duration distribution.

The constraint on the use of clock names in an IGSMP that we consider concerns the names n which are used for clocks when they start. As we already explained the name used for a starting clock must be fresh, i.e. no clock with such a name must be currently in execution. The requirement that we now add is that the new clock name which is used must depend from the duration distribution f associated with the starting clock and from the names of the clocks (with the same distribution f) already in execution, according to a fixed rule. In particular, we take the set of clock names to be defined by $\mathcal{C}Names = (PDF^+ \times \mathbb{N}^+)$, where "$f, i$" is a name for a clock with associated distribution f. The name "f, i" which is used for a starting clock must be such that i is the least $i \in \mathbb{N}^+$ which is not used in the name of any clock with the same distribution f already in execution. Note that, using just duration distributions as clock names is not sufficient because indexes $i \in \mathbb{N}^+$ are needed in order to have different clock names when clocks with the same duration distribution are simultaneously executed.

By applying the dynamic name technique introduced in Chapter 3 to clock names, we have that, since the method to compute the index for a starting clock is fixed, clocks of systems that perform equivalent execution traces get the same names. As a consequence, when establishing equivalence of well-named IGSMPs we do not have to associate clock names (as, e.g., we did for interactive timed automata in Chapter 5) but we can rely on a simple extension of standard bisimilarity.

Since in a well-named IGSMP names for clocks cannot be chosen arbitrarily and the clock names which are considered make it clear by themselves which is the duration distribution associated with a clock, with respect to IGSMTSs (Definition 6.1), in the definition of well-named IGSMTSs we omit set $\mathcal{C}$ and function D.

Definition 6.3 *A well-named Interactive Generalized Semi-Markovian Transition System is a tuple* $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ *where* Σ *and* Act *are defined as in Definition 6.1, while the definition of the transition relations* T_+, T_- *and* T_a *is obtained from that given in Definition 6.1 by substituting the constraint* $b)$ *of item* 3 *with:*

$$b) \quad \exists C_{f,i}, w.\, s \xrightarrow{\langle C^+_{f,i}, w\rangle} \implies i = \min\{j \mid j \in \mathbb{N}^+,\, C_{f,j} \notin \mathcal{S}(s)\} \qquad \blacksquare$$

Note that the new version of constraint $b)$ guarantees that the name used for a starting clock is always fresh as required by the old version of constraint $b)$ (see Definition 6.1).

Definition 6.4 *A well-named Interactive Generalized Semi-Markov Process is a tuple* $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a, s_0)$*, where* $s_0 \in \Sigma$ *is the initial state of the well-named IGSMP and* $(\Sigma, Act, T_+, T_-, T_a)$ *is a well-named IGSMTS such that function* $\mathcal{S}$ *in item* 3 *of Definition 6.1 also satisfies* $\mathcal{S}(s_0) = \emptyset$. $\blacksquare$

As an important remark, we would like to point out that, since the rule expressed by constraint $b)$ of Definition 6.4 reuses the indexes i of terminated clocks, each IGSMP with a finite set of states can be transformed into a well-named IGSMP with a finite set of states, by renaming clocks.

6.2.3 Parallel of Well-Named IGSMPs

Now, we address the problem of defining parallel composition à la CSP [109] of well-named IGSMPs, where the standard actions of a given set S are required to synchronize and the synchronization of two actions of type a is again an action of type a.

Intuitively, it should be clear that when composing in parallel two IGSMPs, a suitable renaming of the clocks is necessary in order to obtain a IGSMP, i.e. preserve the requirements on transition relations of Definition 6.2. Indeed composing in parallel two IGSMPs could lead to some conflict concerning the identification of the clocks of the composed model through names. More precisely, we have to cope with a name conflict whenever two clocks with the same name "f, i" are simultaneously in execution in both IGSMPs. In such a case the same name identifies two different clocks by compromising the relationship between the start and termination events of the two clocks. When considering well-named IGSMPs instead of just IGSMPs we have in addition the problem of preserving the rule for the name of starting clocks expressed by constraint $b)$ of Definition 6.4.

The solution that we adopt, which is taken from the dynamic name technique of Chapter 3, consists in using l and r (left and right) as references to the two well-named IGSMPs $\mathcal{G}', \mathcal{G}''$ which are composed in parallel as $\mathcal{G}' \,\|_S\, \mathcal{G}''$ and relating each clock name locally used in $\mathcal{G}'$ (or $\mathcal{G}''$) to the well-named IGSMP $\mathcal{G}'$ (or $\mathcal{G}''$)

through the reference l (or r). In this way C_{f,l_i} (C_{f,r_i}) denotes the clock $C_{f,i}$ executed by $\mathcal{G}'$ ($\mathcal{G}''$). In order to obtain a well-named IGSMP, when building the composed model, such "extended" names are renamed so that the rule for the name of starting clocks expressed by constraint b) of Definition 6.4 is satisfied. For instance, let us suppose that both $\mathcal{G}'$ and $\mathcal{G}''$ execute a clock with the same duration distribution f. For both well-named IGSMPs in isolation we represent such an event by activating the clock $C_{f,1}$. Somehow in the composed model we have to distinguish such clocks through names because they can be simultaneously in execution. Let us suppose that in $\mathcal{G}' \|_S \mathcal{G}''$ the first delay with distribution f that starts is the one executed by $\mathcal{G}'$. According to the well-naming rule in the composed model such a clock must get name "$f,1$". Hence we map $C_{f,1}$ to the "extended" name of the clock $C_{f,1}$ executed by $\mathcal{G}'$, thus creating the following mapping:

$$C_{f,1} \longrightarrow C_{f,l_1}$$

denoting that the first clock with distribution f of the composed model $C_{f,1}$ corresponds to the first clock with distribution f of the lefthand well-named IGSMP. Then, if the second clock to be executed is the clock $C_{f,1}$ belonging to the righthand well-named IGSMP, in the composed model we create the fresh name "$f,2$" (according to the well-naming rule) and have in addition the following mapping:

$$C_{f,2} \longrightarrow C_{f,r_1}$$

In Table 6.1 we present an example of execution of a composed model $\mathcal{G}' \|_S \mathcal{G}''$ by showing how the mapping function (between the clock names of the composed model $\mathcal{G}' \|_S \mathcal{G}''$ and the corresponding clock names locally used in $\mathcal{G}'$ and $\mathcal{G}''$) for clocks with distribution f evolves.

Well-named IGSMPs	*Composed Model*	*Mapping Function*
$\mathcal{G}'$ *starts* $C_{f,1}$	$\mathcal{G}' \|_S \mathcal{G}''$ *starts* $C_{f,1}$	$C_{f,1} \longrightarrow C_{f,l_1}$
$\mathcal{G}''$ *starts* $C_{f,1}$	$\mathcal{G}' \|_S \mathcal{G}''$ *starts* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,r_1}$
$\mathcal{G}''$ *starts* $C_{f,2}$	$\mathcal{G}' \|_S \mathcal{G}''$ *starts* $C_{f,3}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,r_1}$ $C_{f,3} \longrightarrow C_{f,r_2}$
$\mathcal{G}''$ *ends* $C_{f,1}$	$\mathcal{G}' \|_S \mathcal{G}''$ *ends* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,3} \longrightarrow C_{f,r_2}$
$\mathcal{G}'$ *starts* $C_{f,2}$	$\mathcal{G}' \|_S \mathcal{G}''$ *starts* $C_{f,2}$	$C_{f,1} \longrightarrow C_{f,l_1}$ $C_{f,2} \longrightarrow C_{f,l_2}$ $C_{f,3} \longrightarrow C_{f,r_2}$

Table 6.1: Renaming of the clocks in $\mathcal{G}' \|_S \mathcal{G}''$

By following such a procedure, we build the composed model by dynamically storing all current mappings between the clock names of the composed model and the local clock names of the two well-named IGSMPs by employing a table (mapping function) for each distribution f. In general, when a clock $C_{f,i}$ with distribution f is started by one of the two composed well-named IGSMPs, we do the following: (i) we choose the first index j for the distribution f which is unused in the composed model (by checking the table related to the duration probability distribution f), and we use the name "f, j" for the clock in the composed model; (ii) we add to the table related to distribution f the mapping $C_{f,j} \longrightarrow C_{f,l_i}$ if the clock is executed by the lefthand well-named IGSMP or $C_{f,j} \longrightarrow C_{f,r_i}$ if the clock is executed by the righthand well-named IGSMP. When a clock $C_{f,i}$ with distribution f is terminated by one of the two composed well-named IGSMPs, we do the following: (i) we establish the name "f, j" associated with the terminating clock in the composed model by checking the table related to distribution f (it must include $C_{f,j} \longrightarrow C_{f,l_i}$ if the clock is executed by the lefthand well-named IGSMP or $C_{f,j} \longrightarrow C_{f,r_i}$ if the clock is executed by the righthand well-named IGSMP); (ii) we remove from the table related to the duration probability distribution f the mapping for the name "f, j" of the composed model.

Now we formally define the parallel composition $\mathcal{G}_1 \,\|_S\, \mathcal{G}_2$ of two well-named IGSMPs $\mathcal{G}_1$ and $\mathcal{G}_2$, where the synchronization set S is a subset of $Act - \{\tau\}$.

We denote with $Loc = \{l, r\}$, ranged over by loc the set of locations, where l stands for left and r for right. We denote a mapping function, whose elements are pairs (j, loc_i), with $mapf$ which ranges over the set $MapF$ of partial bijections from $\mathbb{N}^+$ to $Loc \times \mathbb{N}^+$. Moreover, a global mapping M is a relation from PDF^+ to $\mathbb{N}^+ \times (Loc \times \mathbb{N}^+)$ such that $\forall f \in PDF^+ . M_f \in MapF$ [2], i.e. M is a global mapping including a mapping function for each different duration distribution. We denote the set of global mappings M by $\mathcal{M}$. In the following we use the shorthand $f : (j, loc_i)$ for $(f, (j, loc_i)) \in M$. Finally we make use of the auxiliary function $n : MapF \longrightarrow \mathbb{N}^+$ that computes the new index to be used for a clock name according to the well-naming rule, by choosing the minimum index not used by the other clocks with the same distribution already in execution, i.e. $n(mapf) = min\{k \mid k \notin dom(mapf)\}$.

Definition 6.5 *The parallel composition $\mathcal{G}_1 \,\|_S\, \mathcal{G}_2$ of two well-named IGSMPs $\mathcal{G}_1 = (\Sigma_1, Act, T_{+,1}, T_{-,1}, T_{a,1}, s_{0,1})$ and $\mathcal{G}_2 = (\Sigma_2, Act, T_{+,2}, T_{-,2}, T_{a,2}, s_{0,2})$, with S being the synchronization set, is the tuple $(\Sigma, Act, T_+, T_-, T_a, (s_{0,1}, s_{0,2}, \emptyset))$ with*

- $\Sigma = \Sigma_1 \times \Sigma_2 \times \mathcal{M}$ *the set of states,*
- $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *are the least transition relations, such that* $\forall (s_1, s_2, M) \in \Sigma$.

  1$_l$ $s_1 \xrightarrow{\alpha} s_1'$, $\alpha \notin S \implies (s_1, s_2, M) \xrightarrow{\alpha} (s_1', s_2, M)$

  2 $s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{a} s_2'$, $a \in S \implies$
  $(s_1, s_2, M) \xrightarrow{a} (s_1', s_2', M)$

[2]Given a relation M from A to B, we denote with M_a the set $\{b \in B \mid (a, b) \in M\}$.

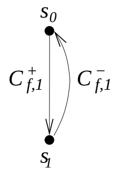


Figure 6.4: Example of Well-Named IGSMP

$$\mathbf{3_l} \;\; s_1 \xrightarrow{\langle C^+_{f,i}, w\rangle} s_1' \;\wedge\; s_2 \not\xrightarrow{\tau} \;\;\Longrightarrow$$
$$(s_1, s_2, M) \xrightarrow{\langle C^+_{f,n(M_f)}, w\rangle} (s_1', s_2, M \cup \{f : (n(M_f), l_i)\})$$

$$\mathbf{4_l} \;\; s_1 \xrightarrow{C^-_{f,i}} s_1' \;\wedge\; s_2 \not\xrightarrow{\tau} \;\wedge\; \not\exists C_{g,h}, w.\, s_2 \xrightarrow{\langle C^+_{g,h}, w\rangle} \;\wedge\; f\!:\!(j, l_i) \in M \;\Longrightarrow$$
$$(s_1, s_2, M) \xrightarrow{C^-_{f,j}} (s_1', s_2, M - \{f\!:\!(j, l_i)\})$$

and also the symmetric rules $\mathbf{1_r}, \mathbf{3_r}, \mathbf{4_r}$ *referring to the local transitions of* $\mathcal{G}_2$*, which are obtained from the rules* $\mathbf{1_l}, \mathbf{3_l}, \mathbf{4_l}$ *by exchanging the roles of states* s_1 *(*s_1'*) and* s_2 *(*s_2'*) and by replacing* l_i *with* r_i*, hold true.*

- $(s_{0,1}, s_{0,2}, \emptyset) \in \Sigma$ *the initial state* ■

Each state $s \in \Sigma$ of the composed model is represented by a triple including a pair of states ($s_1 \in \Sigma_1$ and $s_2 \in \Sigma_2$) and an auxiliary memory M containing all the mappings currently active in such a state. Rules **1** (**2**) describe the behavior of the composed model in the case of a standard action α performed by one (or both, via a synchronization) well-named IGSMPs, when $\alpha \notin S$ ($\alpha \in S$). Rules **3** and **4** define the behavior of the composed model in the case of delays locally performed by components. When in $\mathcal{G}_1$ ($\mathcal{G}_2$) occurs a transition labeled with $\langle C^+_{f,i}, w\rangle$, denoting the beginning of a delay with duration distribution f, then the new index $n(M_f)$ is determined for identifying the action at the level of the composed model, and the new mapping $f : (n(M_f), l_i)$ ($f : (n(M_f), r_i)$) is added to M. Conversely, when in $\mathcal{G}_1$ ($\mathcal{G}_2$) occurs a transition labeled with $C^-_{f,i}$, denoting the termination of a clock with duration distribution f, the particular clock with index j associated to l_i (r_i) in M_f terminates at the level of the composed model, and the index j becomes available. Note that the negative clauses in the premises enforce the maximal progress and the urgency of delays assumptions.

Example 6.1 Call $\mathcal{G}$ the well-named IGSMP of Fig. 6.4 which repeatedly executes delays with distribution f, we depict in Fig. 6.5 the well-named IGSMP $\mathcal{G} \,\|_\emptyset\, \mathcal{G}$. In Fig. 6.4 and 6.5 and in the following we consider C^+_n to be a shorthand for $<C^+_n, 1>$. ■

Theorem 6.1 *Let* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *be two well-named IGSMPs. Then for each* $S \subseteq Act - \{\tau\}$*,* $\mathcal{G}_1 \,\|_S\, \mathcal{G}_2$ *is a well-named IGSMP.*

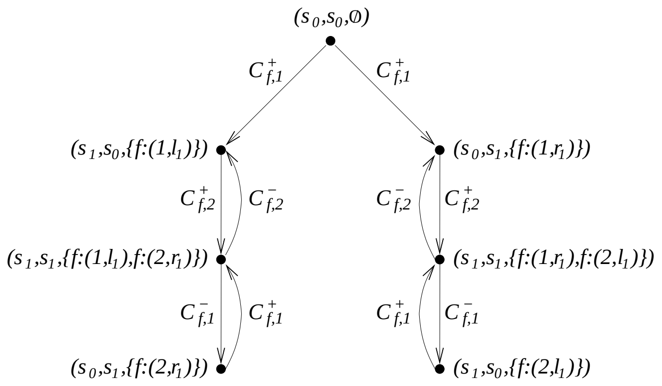


Figure 6.5: Example of Parallel Composition of Well-Named IGSMPs

Proof *Just a trivial consequence of the fact that the maximal progress and urgency of delays assumptions are enforced when composing well-named IGSMPs.* ■

6.2.4 Hiding of Well-Named IGSMPs

Now, we address the problem of defining hiding of well-named IGSMPs, where the standard actions of a given set L are turned into invisible τ actions.

As we already explained, the capability of hiding actions make it possible to turn visible "incomplete" actions into invisible "complete" ones, thus giving the possibility of building a complete system from several system components. In particular while a visible action transition (as long as it is enabled) can delay indefinitely before being performed, when such an action is turned into an invisible action it must be executed in zero time.

Now we formally define the hiding $\mathcal{G}/L$ of a well-named IGSMP $\mathcal{G}$, where the set L of the visible actions to be hidden is a subset of $Act - \{\tau\}$.

Definition 6.6 *The hiding $\mathcal{G}/L$ of a well-named IGSMP $\mathcal{G} = (\Sigma, Act, T_{+,1}, T_{-,1}, T_{a,1}, s_0)$ with L being the set of visible actions to be hidden is the tuple $(\Sigma, Act, T_+, T_-, T_a, s_0)$ where $T_+ \subseteq (\Sigma \times \mathcal{C}^+ \times \Sigma)$, $T_- \subseteq (\Sigma \times \mathcal{C}^- \times \Sigma)$, and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ are the least set of transitions, such that $\forall s \in \Sigma$.* [3]

1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

3 $s \xrightarrow{\theta}_1 s' \wedge\ \not\exists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{\theta} s'$ ■

[3]In order to distinguish transition of $T_{+,1}$, $T_{-,1}$ and $T_{a,1}$ from transitions of T_+, T_- and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

Rules **1** and **2** are standard. Rule **3** says that the effect of the hiding operator over states of $\mathcal{G}$ which enable standard actions in L is to preempt all clock related transitions according to the maximal progress assumption.

Theorem 6.2 *Let $\mathcal{G}$ be a well-named IGSMP. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{G}/L$ is a well-named IGSMP.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption is enforced when hiding well-named IGSMPs.* ■

6.2.5 Equivalence of Well-Named IGSMPs

Now we will introduce a notion of strong and weak bisimulation over well-named IGSMPs. In particular weak bisimulation matches start and termination events of clocks with the same duration distribution by using strong bisimulation, deals with probabilistic choices similarly as in [117], and abstracts from standard τ actions similarly as in [122]. In the following Chapter 7 we will show weak bisimulation to be a congruence with respect to both parallel composition and hiding. In our context we express cumulative probabilities by aggregating weights.

Definition 6.7 *Let $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ be a well-named IGSMTS.The function $TW : \Sigma \times PDF^+ \times \mathcal{P}(\Sigma) \longrightarrow \mathbb{R}^+ \cup \{0\}$, which computes the aggregated weight that a state $s \in \Sigma$ reaches a set of states $I \in \mathcal{P}(\Sigma)$ by starting a delay with duration distribution $f \in PDF^+$ is defined as:* [4]

$$TW(s, f, I) = \sum \{\!| w \mid \exists i \in \mathbb{N}^+, s' \in I.\ s \xrightarrow{\langle C^+_{f,i}, w\rangle} s' |\!\}$$

■

We are now in a position to define the notion of strong bisimilarity for well-named IGSMPs. Let $NPAct = Act \cup \mathcal{C}^-$, the set of non-probabilistic actions, be ranged over by σ.

Definition 6.8 *Let $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ be a well-named IGSMTS. An equivalence relation β on Σ is a strong bisimulation iff $s_1 \ \beta \ s_2$ implies*

- *for every $\sigma \in NPAct$,*

  $s_1 \xrightarrow{\sigma} s'_1$ *implies* $s_2 \xrightarrow{\sigma} s'_2$ *for some* s'_2 *with* $s'_1 \ \beta \ s'_2$,

- *for every $f \in PDF^+$ and equivalence class I of β,*

  $TW(s_1, f, I) = TW(s_2, f, I)$

[4] We use $\{\!|$ and $|\!\}$ to denote multiset parentheses. The summation of an empty multiset is assumed to yield 0. Since the method for computing the new index of a delay f that starts in a state P is fixed, we have that several transitions f^+ leaving P have all the same index i.

Two states s_1 *and* s_2 *are strongly bisimilar, denoted by* $s_1 \sim s_2$*, iff* (s_1, s_2) *is included in some strong bisimulation. Two well-named IGSMPs* $(\mathcal{G}_1, s_{0,1})$ *and* $(\mathcal{G}_2, s_{0,2})$ *are strongly bisimilar, if their initial states* $s_{0,1}$ *and* $s_{0,2}$ *are strongly bisimilar in the well-named IGSMTS obtained with the disjoint union of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$*.* ■

The definition of weak bisimilarity is an adaptation of that presented in [105, 99] to our context.

Let $\stackrel{\sigma}{\Longrightarrow}$ denote $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\sigma}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$, i.e. a sequence of transitions including a single σ transition and any number of τ transitions. Moreover, we define $\stackrel{\hat{\sigma}}{\Longrightarrow} \; = \; \stackrel{\sigma}{\Longrightarrow}$ if $\sigma \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow} \; = (\stackrel{\tau}{\longrightarrow})^*$, i.e. a possibly empty sequence of τ transitions. Moreover we let I^τ denote the set of processes that may silently evolve into an element of I, i.e. $I^\tau = \{s_1 \mid \exists s_2 \in I : s_1 \stackrel{\hat{\tau}}{\Longrightarrow} s_2\}$.

Definition 6.9 *Let* $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ *be a well-named IGSMTS. An equivalence relation* β *on* Σ *is a weak bisimulation iff* $s_1 \; \beta \; s_2$ *implies*

- *for every* $\sigma \in NPAct$ *and* $s'_1 \in \Sigma$*,*

  $s_1 \stackrel{\hat{\sigma}}{\Longrightarrow} s'_1$ *implies* $s_2 \stackrel{\hat{\sigma}}{\Longrightarrow} s'_2$ *for some* s'_2 *with* $s'_1 \; \beta \; s'_2$*,*

- *for every* $s'_1 \in \Sigma$*,*

  $s_1 \stackrel{\hat{\tau}}{\Longrightarrow} s'_1$ *implies* $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s'_2$ *for some* s'_2 *such that, for every* $f \in PDF^+$ *and equivalence class* I *of* β*,*

  $$TW(s'_1, f, I^\tau) = TW(s'_2, f, I^\tau)$$

Two states s_1 *and* s_2 *are weakly bisimilar, denoted by* $s_1 \approx s_2$*, iff* (s_1, s_2) *is included in some weak bisimulation. Two well-named IGSMPs* $(\mathcal{G}_1, s_{0,1})$ *and* $(\mathcal{G}_2, s_{0,2})$ *are weakly bisimilar, if their initial states* $s_{0,1}$ *and* $s_{0,2}$ *are weakly bisimilar in the well-named IGSMTS obtained with the disjoint union of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$*.* ■

As for IWMCs (see Chapter 4), in the definition of equivalence we do not express conditions about the stability of bisimilar states as done in [105, 99] because, as we will see in Chapter 7 where we introduce the calculus of IGSMPs, we will be interested in obtaining a congruence result only for strongly guarded processes of the calculus.

Similarly as in [105, 99] it is possible to reformulate weak bisimilarity in the following way, which is simpler but less intuitive.

Theorem 6.3 *Let* $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a)$ *be a well-named IGSMTS. An equivalence relation* β *on* Σ *is a weak bisimulation iff* $s_1 \; \beta \; s_2$ *implies*

- *for every* $\sigma \in NPAct$ *and* $s'_1 \in \Sigma$*,*

  $s_1 \stackrel{\sigma}{\longrightarrow} s'_1$ *implies* $s_2 \stackrel{\hat{\sigma}}{\Longrightarrow} s'_2$ *for some* s'_2 *with* $s'_1 \; \beta \; s'_2$*,*

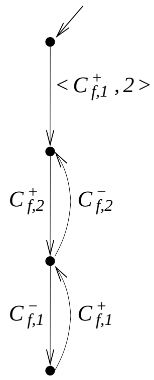


Figure 6.6: Minimal Well-Named IGSMP

- $s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s'_2$ *for some* s'_2 *such that, for every* $f \in PDF^+$ *and equivalence class* I *of* β,

  $TW(s_1, f, I) = TW(s'_2, f, I)$ ■

The proof that this reformulation is correct derives from that given in [99], simply by substituting rates of exponential distributions with weights.

Example 6.2 In Fig. 6.6 we depict the minimal semantic model for the well-named IGSMP of Fig. 6.5, which is obtained by merging weakly bisimilar states. The weight 2 of the initial transition derives from the aggregation of the weights of the two initial transitions in the model of Fig. 7.3. However since in the initial state there is no alternative to such a transition, its weight is not relevant for the actual behavior (in isolation) of the IGSMP. ■

6.3 Deriving the Performance Model from an IGSMP

In this section we show how to formally derive a GSMP from an IGSMP system specification. In particular this transformation is possible only if the specification of the system is *complete* both from the *interaction* and from the *performance* point of view.

An IGSMP specification is complete from the interaction viewpoint if the system specified is not a part of a larger system which may influence its behavior, hence when every standard action appearing in the IGSMP is an *internal* τ action.

An IGSMP specification is complete from the performance viewpoint if all the choices in which the specified system may engage are quantified probabilistically. This means that the IGSMP must not include silent states actually leading to a non-deterministic choice among different future behaviors. In other words a silent state either must have only one outgoing τ transition, or all its outgoing τ transitions must lead to equivalent behaviors. This notion can be formally defined as follows: An IGSMP is complete w.r.t.

performance if it can be reduced, by aggregating weakly bisimilar states (see Sect. 6.2.5), to a model without silent states.

Provided that a Well-Named IGSMP $\mathcal{G} = (\Sigma, Act, T_+, T_-, T_a, s_0)$ satisfies these two conditions, we now present a formal procedure for deriving the *GSMP* representing the performance behavior of $\mathcal{G}$.

6.3.1 Elimination of τ Actions

The first phase is to *minimize* the state space Σ of the Well-Named IGSMP $\mathcal{G}$ by aggregating states that are equivalent according to the notion of weak bisimulation defined in Sect. 6.2.5. Since we supposed that the $\mathcal{G}$ satisfies the two conditions above, a side effect of this minimization is that all τ actions disappear from $\mathcal{G}$.

We denote the resulting Well-Named IGSMP with $\mathcal{G}_m = (\Sigma_m, Act, T_{+,m}, T_{-,m}, T_{a,m}, s_0)$, where m stands for "minimal". We have $T_{a,m} = \emptyset$, hence Σ_m includes probabilistic states, timed states and waiting states only.

6.3.2 Solution of Probabilistic Trees

The second phase is the transformation of every *probabilistic tree* present in the semantic model into a single probabilistic choice. A probabilistic tree is formed by the possible probabilistic paths that go from a given probabilistic state (the root of the tree) to a timed or waiting state (a leaf of the tree). Note that such trees cannot include loops composed of one or more transitions, because after each clock start the number of clocks in execution strictly increases. To be precise, such trees are directed acyclic graphs (*DAGs*) with root, since a node may have multiple incoming arcs. The probabilistic trees are flattened into a single choice that goes directly from the root to the leaves of the tree, with the following inductive procedure.

Initially (at step 0) we transform our semantic model by turning all weights into the corresponding probability values. We denote the resulting *LTS* with $(S_{\mathcal{G},p,0}, L_p, \longrightarrow_{\mathcal{G},p,0}, s_{\mathcal{G},0})$, where p stands for "probabilistic", where:

- $S_{\mathcal{G},p,0} = \Sigma_m$ is the set of states
- $L_p = \mathbb{R}_{]0,1]} \cup TAct^-$ is the set of transition labels, where positive real numbers represent probabilities
- $\longrightarrow_{\mathcal{G},p,0} \; = (T_{-,m} \cup \{(s, prob, s') \mid \exists f, i, w : (s, <f_i^+, w>, s') \in T_{+,m} \wedge prob = w/TW(s)\}$ is the transition relation, where:
$$TW(s) = \sum \{\!|\, w \mid \exists f, i, s' : (s, <f_i^+, w>, s') \in T_{+,m} \,|\!\}$$
- $s_{\mathcal{G},0} = s_0$ is the initial state

Hence now we have a semantic model with clock termination transitions and *probabilistic* transitions labeled by a probability *prob*. Note that clock start events are removed from transition labels. The occurrence of such events becomes implicit in the representation of system behavior similarly as in *GSMPs*.

At the k-th step, beginning from the *LTS* $(S_{\mathcal{G},p,k-1}, L_p, \longrightarrow_{\mathcal{G},p,k-1}, s_{\mathcal{G},0})$ we eliminate a node in a probabilistic tree, thus reducing its size. This is done by considering a probabilistic state $s \in S_{\mathcal{G},p,k-1}$ with incoming probabilistic transitions. Such transitions are removed and replaced by a new set of probabilistic transitions which are determined in the following way. Each incoming probabilistic transition is divided into multiple transitions, one for each probabilistic transition that leaves the state s. Its probability is distributed among the new transitions in parts that are proportional to the probabilities of the transitions that leave s. Moreover, if s has no incoming clock termination transitions, then s is eliminated together with its outgoing probabilistic transitions. Therefore the resulting *LTS* $(S_{\mathcal{G},p,k}, L_p, \longrightarrow_{\mathcal{G},p,k}, s_{\mathcal{G},0})$, is defined by:

- $S_{\mathcal{G},p,k} = \begin{cases} S_{\mathcal{G},p,k-1} & \text{if } \exists f, i, s' : s' \xrightarrow{f_i^-}_{\mathcal{G},p,k-1} s, \\ S_{\mathcal{G},p,k-1} - \{s\} & \text{otherwise} \end{cases}$

- $\longrightarrow_{\mathcal{G},p,k} = (\longrightarrow_{\mathcal{G},p,k-1} \cap S_{\mathcal{G},p,k-1} \times \mathit{TAct}^- \times S_{\mathcal{G},p,k-1}) \cup$
  $\{(s', \mathit{prob}, s'') \mid s', s'' \in S_{\mathcal{G},p,k} \wedge s'' \neq s \wedge$
  $\mathit{MultiProb}_s(s', s'') \neq \emptyset \wedge \mathit{prob} = \sum \mathit{MultiProb}_s(s', s'')\}$

  where:

  $$\begin{aligned} \mathit{MultiProb}_s(s', s'') = \{\!|\, \mathit{prob} \mid\ & s' \xrightarrow{\mathit{prob}}_{\mathcal{G},p,k-1} s'' \vee \\ & (\exists \mathit{prob}', \mathit{prob}'' : s' \xrightarrow{\mathit{prob}'}_{\mathcal{G},p,k-1} s \wedge \\ & s \xrightarrow{\mathit{prob}''}_{\mathcal{G},p,k-1} s'' \wedge \mathit{prob} = \mathit{prob}' \cdot \mathit{prob}'')\, |\!\} \end{aligned}$$

The algorithm terminates when we reach k for which $\nexists s \in S_{\mathcal{G},p,k-1}$ with s probabilistic state with incoming probabilistic transitions. Since at every step we eliminate a node in a probabilistic tree of the initial semantic model, thus reducing its size, we are guaranteed that this will eventually happen. Let t be such k.

The *LTS* that results from this second phase is denoted by $(S_{\mathcal{G},p}, L_p, \longrightarrow_{\mathcal{G},p}, s_{\mathcal{G},0}) = (S_{\mathcal{G},p,t}, L_p, \longrightarrow_{\mathcal{G},p,t}, s_{\mathcal{G},0})$.

If the nodes of a probabilistic tree are eliminated by following a breadth-first visit from the root, it can be easily seen that the time complexity of the algorithm above is just linear in the number of probabilistic transitions forming the tree (the *DAG*). This because by following this elimination ordering, each node to be eliminated has one ingoing probabilistic transition only.

6.3.3 Derivation of the GSMP

Now we show how to derive a generalized semi-Markov process $(\mathcal{S}, El, ElPDF, ElSt, Dec, \longrightarrow, Pr, P_{init})$, see Sect. 2.2.6, from the semantic model obtained at the end of the previous phase. With respect to the general definition of a *GSMP* given in Sect. 2.2.6, we have that in an *IGSMP* all elements (clocks) decay at rate 1, i.e. they all advance uniformly with time at the same speed.

The *performance model* $\mathcal{P}[\![\mathcal{G}]\!]$ of a Well-Named IGSMP $\mathcal{G}$ is derived from the *LTS* $(S_{\mathcal{G},p}, L_p, \longrightarrow_{\mathcal{G},p}, s_{\mathcal{G},0})$ as follows.

The elements of the *GSMP* are the clocks $C_{f,i}$ labeling the transitions of $\longrightarrow_{\mathcal{G},p}$. The states of the *GSMP* are the timed and waiting states of $S_{\mathcal{G},p}$. A transition leaving a state of the *GSMP* is derived beginning from a clock termination transition leaving the corresponding timed state of $S_{\mathcal{G},p}$ and, in the case this transition leads to a probabilistic state, from a probabilistic transition leaving this state. The timed or waiting state of $S_{\mathcal{G},p}$ reached in this way is the state of the *GSMP* the derived transition leads to. Note that we are certain to reach a timed or waiting state because all probabilistic trees have been solved and, consequently (see Sect. 6.3.2) probabilistic states cannot have incoming probabilistic transitions. Each transition of the *GSMP* is labeled by the element $C_{f,i}$ terminating in the corresponding termination transition. The probability associated with a transition of the *GSMP* (function *Pr*) is the probability of the corresponding probabilistic transition (or probability 1 if the transition is derived from a clock termination transition leading directly to a timed or waiting state).

The performance semantics of a Well-Named *IGSMP* $\mathcal{G}$ is defined as follows.

Definition 6.10 The *performance semantics* $\mathcal{P}[\![\mathcal{G}]\!]$ of a Well-Named *IGSMP* $\mathcal{G}$ is a *GSMP* represented by the tuple:

$$\mathcal{P}[\![\mathcal{G}]\!] = (\mathcal{S}, El, ElPDF, ElSt, Dec, \longrightarrow, Pr, P_{init})$$

where:

- $\mathcal{S} = \{s \in S_{\mathcal{G},p} \mid \nexists prob, s' : s \xrightarrow{prob}_{\mathcal{G},p} s'\}$
- $El = \{C_{f,i} \mid \exists s, s' : s \xrightarrow{C^-_{f,i}}_{\mathcal{G},p} s'\}$
- $\forall C^-_{f,i} \in El.\, ElPDF(C^-_{f,i}) = f$
- $\forall s \in \mathcal{S}.\, ElSt(s) = \{C^-_{f,i} \mid \exists s' : s \xrightarrow{C^-_{f,i}}_{\mathcal{G},p} s'\}$
- $\forall s \in \mathcal{S}, \forall C^-_{f,i} \in ElSt(s).\;\; Dec(C^-_{f,i}, s) = 1$
- We let $\longrightarrow \; = \; \longrightarrow_1 \cup \longrightarrow_2$ where:
  - $\longrightarrow_1 \; = \{(s, C^-_{f,i}, s') \mid s \xrightarrow{C^-_{f,i}}_{\mathcal{G},p} s' \wedge s' \in \mathcal{S}\}$
  - $\longrightarrow_2 \; = \{(s, C^-_{f,i}, s') \mid \exists s'', prob : s \xrightarrow{C^-_{f,i}}_{\mathcal{G},p} s'' \wedge s'' \xrightarrow{prob}_{\mathcal{G},p} s'\}$ [5]

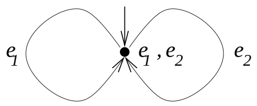
[5] The transition relations $\longrightarrow_1$ and $\longrightarrow_2$ are disjoint. This because the only transition in $\longrightarrow_{\mathcal{G},p}$ that leaves state s and is labeled by a given element $C^-_{f,i}$, leads either to a probabilistic state or to a timed/waiting state.

- $\forall s, s' \in \mathcal{S}; C^-_{f,i} \in ElSt(s)$ such that $s \xrightarrow{C^-_{f,i}} s'$
  - if $s \xrightarrow{C^-_{f,i}}_1 s'$ let $Pr(s, C^-_{f,i}, s') = 1$
  - if $s \xrightarrow{C^-_{f,i}}_2 s'$ let $Pr(s, C^-_{f,i}, s')$ be the unique *prob* such that:

$$\exists s'' : s \xrightarrow{C^-_{f,i}}_{\mathcal{G},p} s'' \wedge s'' \xrightarrow{prob}_{\mathcal{G},p} s'$$

- P_{init} is defined as follows:
  - if $s_{\mathcal{G},0} \in \mathcal{S}$ then:
    * $P_{init}(s_{\mathcal{G},0}) = 1$
    * $P_{init}(s) = 0 \quad \forall s \in \mathcal{S}, s \neq s_{\mathcal{G},0}$
  - if $s_{\mathcal{G},0} \notin \mathcal{S}$ then $\forall s \in \mathcal{S}$:
    * $P_{init}(s) = prob \quad$ if $s_{\mathcal{G},0} \xrightarrow{prob}_{\mathcal{G},p} s$
    * $P_{init}(s) = 0 \quad$ if $\nexists prob : s_{\mathcal{G},0} \xrightarrow{prob}_{\mathcal{G},p} s$ ■

Example 6.3 In Fig. 6.7 we show the *GSMP* derived, by applying the formal translation we have presented, from the Well-Named IGSMP of Fig. 6.5. In particular the *GSMP* is obtained, as described above, from the minimal model of Fig. 6.6. Since such model does not include standard action transitions the system considered is complete both from the interactive and the performance viewpoints. In the *GSMP* of Fig. 6.7 the states are labeled by the active elements and the transitions with the terminating elements. The probability Pr associated to each transition of the *GSMP* (see Chapter 2) that is shown in the picture is 1. Moreover P_{init} is 1 for the unique state of the *GSMP* (it is pointed by the arrow to point out the fact that it is the initial state). The elements e_1 and e_2 represent the clocks $C_{f,1}$ and $C_{f,2}$ respectively, and the probability distribution function of both is given by function f. ■



$$ElPDF(e_1) = f, ElPDF(e_2) = f$$

Figure 6.7: Derived GSMP

6.4 Interactive Stochastic Timed Transition Systems

In this section we introduce Interactive Stochastic Timed Transition Systems (ISTTSs) that will be used in the next section to define a semantics for IGSMPs. We first briefly introduce some basic notions about probability spaces.

6.4.1 Probability Spaces

In this section we recall some basic notions related to measure theory and we introduce some notation that will be used in the rest of the chapter.

Definition 6.11 *A σ-algebra on a set Ω, denoted by $\mathcal{F}$, is a family of subsets of Ω that contains Ω and is closed under complementation and countable union. The elements of a σ-algebra $\mathcal{F}$ are called measurable sets. The pair $(\Omega, \mathcal{F})$ is called a measurable space.* ■

Definition 6.12 *The Borel σ-algebra on a set Ω with a topology, denoted by $\mathcal{B}(\Omega)$, is defined to be the σ-algebra generated by the open subsets (or equivalently, by the closed subsets) of Ω.* ■

Definition 6.13 *A finite measure μ on a measurable space $(\Omega, \mathcal{F})$ is a function that assigns a non-negative real value to each element of $\mathcal{F}$, such that $\mu(\emptyset) = 0$ and, supposing $\{C_i\}_{i \in I}$, with $I \subseteq \mathbb{N}$, to be a family of disjoint elements of $\mathcal{F}$, $\mu(\cup_{i \in I} C_i) = \sum_{i \in I} \mu(C_i)$. The triple $(\Omega, \mathcal{F}, \mu)$ is called a measure space. If we have in addition that $\mu(\Omega) = 1$, then the triple $(\Omega, \mathcal{F}, \mu)$ is also called a probability space.* ■

We now define some operations over measure spaces and probability spaces.

Definition 6.14 *Let $(\Omega, \mathcal{F}, \mu)$ be a measure space and let f be a function defined on Ω, then $f(\Omega, \mathcal{F}, \mu)$ denotes the triple $(f(\Omega), \{C \subseteq f(\Omega) \mid f^{-1}(C) \in \mathcal{F}\}, \mu'\})$, where $\forall C \subseteq f(\Omega).\ \mu'(C) = \mu(f^{-1}(C))$.* ■

It is easy to verify that, since $(\Omega, \mathcal{F}, \mu)$ is a measure space, $f(\Omega, \mathcal{F}, \mu)$ is a measure space as well. Such a measure space is called the measure space *induced* by f from $(\Omega, \mathcal{F}, \mu)$. Moreover if $(\Omega, \mathcal{F}, \mu)$ is a probability space, then $f(\Omega, \mathcal{F}, \mu)$ is a probability space as well.

Definition 6.15 *Let $(\Omega, \mathcal{F}, \mu)$ be a measure space and let p be a positive real number, then $p \cdot (\Omega, \mathcal{F}, \mu)$ denotes the triple $(\Omega, \mathcal{F}, \mu')$, where $\forall C \subseteq \Omega.\ \mu'(C) = p \cdot \mu(C)$.* ■

It is easy to verify that, since $(\Omega, \mathcal{F}, \mu)$ is a measure space, $p \cdot (\Omega, \mathcal{F}, \mu)$ is a measure space as well.

Definition 6.16 *Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be two measure spaces, then $(\Omega', \mathcal{F}', \mu') + (\Omega'', \mathcal{F}'', \mu'')$ denotes the triple $(\Omega' \cup \Omega'', \{C \subseteq \Omega' \cup \Omega'' \mid C \cap \Omega' \in \mathcal{F}' \wedge C \cap \Omega'' \in \mathcal{F}''\}, \mu)$, where $\forall C \subseteq \Omega' \cup \Omega''.\ \mu(C) = \mu'(C \cap \Omega') + \mu''(C \cap \Omega'')$.* ■

It is easy to verify that, since $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ are measure spaces, $(\Omega', \mathcal{F}', \mu') + (\Omega'', \mathcal{F}'', \mu'')$ is a measure space as well. Moreover, considering a set of probability spaces $\{(\Omega_i, \mathcal{F}_i, \mu_i)\}_{i \in I}$, with $I \subseteq \mathbb{N}$ finite index set, and a set $\{p_i\}_{i \in I}$ of positive real numbers such that $\sum_{i \in I} p_i = 1$, we have that $\sum_{i \in I} p_i \cdot (\Omega_i, \mathcal{F}_i, \mu_i)$ is a probability space.

Definition 6.17 *Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be two measure spaces, then $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ denotes the triple $(\Omega' \times \Omega'', \{C' \times C'' \mid C' \in \mathcal{F}' \wedge C'' \in \mathcal{F}''\}, \mu')$, where $\forall C' \subseteq \Omega', C'' \subseteq \Omega''.\ \mu'(C' \times C'') = \mu'(C') \cdot \mu''(C'')$.* ■

It is easy to verify that, since $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ are measure spaces, $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ is a measure space as well. Moreover, if $(\Omega, \mathcal{F}, \mu)$ is a probability space, then $(\Omega', \mathcal{F}', \mu') \cdot (\Omega'', \mathcal{F}'', \mu'')$ is a probability space as well.

We now show that a probability distribution function on real numbers defines a unique probability space over the Borel σ-algebra of real numbers.

Theorem 6.4 *Let $F \in PDF$ be a probability distribution function on $\mathbb{R}$. There is a unique probability measure P on $\mathcal{B}(\mathbb{R})$ such that $\forall a, b \in \mathbb{R}, a < b.\ P(\,(a, b]\,) = F(b) - F(a)$.* ■

Finally, we present a notion of equivalence over measure spaces which relates measure spaces with different domains by assuming that they assign measure 0 to the every set of elements not included in their domain.

Definition 6.18 *Let $(\Omega', \mathcal{F}', \mu')$ and $(\Omega'', \mathcal{F}'', \mu'')$ be measure spaces. We say that $(\Omega', \mathcal{F}', \mu')$ is equivalent to $(\Omega'', \mathcal{F}'', \mu'')$, written $(\Omega', \mathcal{F}', \mu') \approx (\Omega'', \mathcal{F}'', \mu'')$, if $\forall C' \in \mathcal{F}'.\ C' \cap \Omega'' \in \mathcal{F}'' \wedge \mu''(C' \cap \Omega'') = \mu'(C')$ and $\forall C'' \in \mathcal{F}''.\ C'' \cap \Omega' \in \mathcal{F}' \wedge \mu'(C'' \cap \Omega') = \mu''(C'')$.* ■

Note that the definition above implies that if $(\Omega', \mathcal{F}', \mu') \approx (\Omega'', \mathcal{F}'', \mu'')$ then both $\Omega' - \Omega'' \in \mathcal{F}'$ with $\mu'(\Omega' - \Omega'') = 0$ and $\Omega'' - \Omega' \in \mathcal{F}''$ with $\mu''(\Omega'' - \Omega') = 0$.

6.4.2 The ISTTS Model

In this section we formally introduce Interactive Stochastic Timed Transition Systems (ISTTS) which include three type of transitions: *standard action transitions*, representing the interactive behavior of a system component, *probabilistic transitions* (expressed by means of probability spaces) representing (infinitely branching) probabilistic choices and *numeric time transitions* representing a fixed temporal delay.

As far as standard actions are concerned they have exactly the same behavior as in IGSMPs. In ISTTS non-deterministic choices can arise not only from transitions labeled with standard visible actions (like in IGSMPs), but also from transitions representing the passage of time. As usual in the real time literature (see e.g. [132]), several timed transition leaving a state offer the possibility to the observer to choose the amount of time after which he wants to observe the status of the system.

In ISTTS we have two different kinds of state:

- *silent states* which are exactly like in IGSMPs.
- *probabilistic states* enabling probabilistic transitions, expressed by a probability space PS, and (possibly) visible action transitions a only. In such states the ISTTS just chooses a new state in zero time according to the probability space and may potentially interact with the environment through one of its visible actions (see e.g. Fig. 6.8.a).
- *timed states* enabling numeric timed transitions t and (possibly) visible action transitions a only. In such states the ISTTS just performs a non-deterministic choice among the numeric timed transitions (which cause the amount of time labeling the transition to pass) and may potentially interact with the environment through one of its visible actions (see e.g. Fig. 6.8.b).

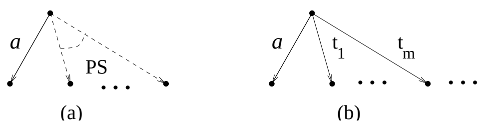


Figure 6.8: Some examples of possible states of an ISTTS

In the following we present the formal definition of Interactive Stochastic Timed Transition System (ISTTS), then we will define Rooted Interactive Stochastic Timed Transition Systems as ISTTSs possessing an initial state. Formally, given a time domain $TD \subseteq \mathbb{R}^+ \cup \{0\}$, we use $t, t', \ldots$, representing time values, to range over TD.

Definition 6.19 *An Interactive Stochastic Timed Transition System (ISTTS) is a tuple* $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a)$ *with*

- Σ *a set of possibly infinite states,*
- TD *a time domain, i.e. the set of possible values over which the labels of the numeric timed transitions range,*
- Act *a set of standard actions,*
- $P : \Sigma' \to PS(\Sigma - \Sigma')$*, where* $\Sigma' \subset \Sigma$ *and* $PS(\Sigma'')$ *denotes the family of probability spaces* $(\Sigma''', \mathcal{F}, \mu)$ *over sets of states* $\Sigma''' \subseteq \Sigma''$*, the probabilistic transition relation which associates a probability space with some of the states of the ISTTS; and* $T_t \subseteq (\Sigma \times TD \times \Sigma)$ *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *two transition relations representing time passage and action execution, respectively.* P*,* T_t *and* T_a *must be such that* $\forall s \in \Sigma$.

  - $s \xrightarrow{\tau} \implies s \notin dom(P) \wedge \nexists t. s \xrightarrow{t}$
  - $s \in dom(P) \implies \nexists t. s \xrightarrow{t}$

- $s \xrightarrow{\tau} \vee \exists t. s \xrightarrow{t} \vee s \in dom(P)$ ∎

Definition 6.20 *A Rooted Interactive Stochastic Timed Transition System (RISTTS) is a tuple* $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a, s_0)$*, where* $s_0 \in \Sigma$ *is the initial state and* $(\Sigma, TD, Act, P, T_t, T_a)$ *is an ISTTS.* ∎

The meaning of the constraints over transition relations is the following. The first requirement says that (similarly as in IGSMPs) if a state can perform internal τ actions then it can perform neither probabilistic transitions nor timed transitions (*maximal progress* assumption). The second requirement says that (similarly as in IGSMPs) if a state can perform probabilistic transitions then it cannot perform timed transitions (*urgency of choices* assumption). The third requirement says that (similarly as in IGSMPs) we cannot have states where time is not allowed to pass (time deadlocks).

6.4.3 Parallel of Rooted ISTTSs

Now we define, similarly as for IGSMPs, the parallel composition à la CSP of RISTTSs.

In such a parallel composition the discrete timed transitions of the composed RISTTSs are constrained to synchronize, so that the same amount of time passes for both systems, i.e. when time advances for one RISTTS it must also advance for the other RISTTS.

Definition 6.21 *The parallel composition* $\mathcal{D}_1 \,\|_S\, \mathcal{D}_2$ *of two RISTTSs* $\mathcal{D}_1 = (\Sigma_1, TD, Act, P_1, T_{t,1}, T_{a,1}, s_{0,1})$ *and* $\mathcal{D}_2 = (\Sigma_2, TD, Act, P_2, T_{t,2}, T_{a,2}, s_{0,2})$*, with* $S \subset Act - \{\tau\}$ *being the synchronization set, is the tuple* $(\Sigma, TD, Act, P, T_t, T_a, (s_{0,1}, s_{0,2}))$ *with:*

- $\Sigma = \Sigma_1 \times \Sigma_2$ *the set of states*
- P *the partial function defined over* $\Sigma_1 \times \Sigma_2$ *obtained from* P_1 *and* P_2 *as follows:* $\forall s_1 \in \Sigma_1, s_2 \in \Sigma_2$.

  $P(s_1, s_2) = Id^1_{s_2}(P_1(s_1))$ *if* $s_1 \in dom(P_1) \wedge s_2 \xrightarrow{t}$

  $P(s_1, s_2) = Id^2_{s_1}(P_2(s_2))$ *if* $s_2 \in dom(P_2) \wedge s_1 \xrightarrow{t}$

  $P(s_1, s_2) = P(s_1) \cdot P(s_2)$ *if* $s_1 \in dom(P_1) \wedge s_2 \in dom(P_2)$

  $P(s_1, s_2)$ *is not defined otherwise*

  with $Id^1_{s_2} : \Sigma_1 \longrightarrow (\Sigma_1 \times \{s_2\})$ *defined by* $\forall s \in \Sigma_1 . Id_{s_2}(s) = (s, s_2)$ *and* $Id^2_{s_1} : \Sigma_2 \longrightarrow (\{s_1\} \times \Sigma_2)$ *defined by* $\forall s \in \Sigma_2 . Id_{s_1}(s) = (s_1, s)$.
- $T_t \subseteq (\Sigma \times TD \times \Sigma)$ *and* $T_a \subseteq (\Sigma \times Act \times \Sigma)$ *the least transition relations, such that*

  1l $s_1 \xrightarrow{\alpha} s_1',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$

  1r $s_2 \xrightarrow{\alpha} s_2',\ \alpha \notin S \implies (s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')$

  2 $s_1 \xrightarrow{a} s_1' \wedge s_2 \xrightarrow{a} s_2',\ a \in S \implies (s_1, s_2) \xrightarrow{a} (s_1', s_2')$

  3 $s_1 \xrightarrow{t} s_1' \wedge s_2 \xrightarrow{t} s_2' \implies (s_1, s_2) \xrightarrow{t} (s_1', s_2')$

- $(s_{0,1}, s_{0,2}) \in \Sigma$ *the initial state.* ■

When evaluating the probability spaces associated by function P to the states of the composed model we make use of induced probability spaces (see Definition 6.14) and we enforce the maximal progress assumption. Moreover we produce a single "global" probability space whenever both RISTTSs engage in probabilistic choices (we assume that choices are performed independently). When evaluating action transitions we just make use of standard rules. Finally we require timed transitions to synchronize.

Theorem 6.5 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two RISTTSs. Then for each $S \subseteq Act - \{\tau\}$, $\mathcal{D}_1 \,\|_S\, \mathcal{D}_2$ is a RISTTS.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption is enforced when composing RISTTSs and that timed transitions must synchronize in order to be executed.* ■

6.4.4 Hiding of Rooted ISTTSs

Now we define, similarly as for IGSMPs, the hiding of RISTTSs.

Definition 6.22 *The hiding $\mathcal{D}/L$ of a RISTTS $\mathcal{D}_1 = (\Sigma, TD, Act, P_1, T_{t,1}, T_{a,1}, s_0)$, with $L \subset Act - \{\tau\}$ being the set of visible actions to be hidden, is the tuple $(\Sigma, TD, Act, P, T_t, T_a, s_0)$, with:*

- *P the partial function obtained from P_1 by removing from its domain those states (and the associated probability spaces) which enable at least one transition labeled with an action in L*
- *$T_t \subseteq (\Sigma \times TD \times \Sigma)$ and $T_a \subseteq (\Sigma \times Act \times \Sigma)$ the least transition relations, such that $\forall s \in \Sigma$.* [6]

  1 $s \xrightarrow{\alpha}_1 s',\ \alpha \notin L \implies s \xrightarrow{\alpha} s'$

  2 $s \xrightarrow{a}_1 s',\ a \in L \implies s \xrightarrow{\tau} s'$

  3 $s \xrightarrow{t}_1 \wedge\ \nexists a \in L.\, s \xrightarrow{a}_1 \implies s \xrightarrow{t}$ ■

Similarly as for IGSMPs, in the definition of the hiding operator in addition to standard rules we make use of rules which enforce the maximal progress assumption.

Theorem 6.6 *Let $\mathcal{D}$ be a RISTTS. Then for each $L \subseteq Act - \{\tau\}$, $\mathcal{D}/L$ is a RISTTS.*

Proof *Just a trivial consequence of the fact that the maximal progress assumption is enforced when hiding RISTTSs.* ■

[6] In order to distinguish transition of $T_{t,1}$ and $T_{a,1}$ from transitions of T_t and T_a we denote the former with "$\longrightarrow_1$" and the latter simply with "$\longrightarrow$".

6.4.5 Equivalence of Rooted ISTTSs

Now we introduce a notion of weak bisimulation for RISTTSs which constitutes an extension of the approach of [117] to probability spaces and abstracts from standard τ actions similarly as in [122].

Given an equivalence relation β on a set Σ and a set $I \subseteq \Sigma$, we first define the function $EC_{I,\beta} : I \to \Sigma/\beta$ which maps each state $s \in I$ into the corresponding equivalence class $[s]_\beta$ in Σ.

Definition 6.23 *Let $\mathcal{D} = (\Sigma, TD, Act, P, T_t, T_a)$ be an ISTTS. An equivalence relation β on Σ is a weak bisimulation iff $s_1 \; \beta \; s_2$ implies*

- *for every $\alpha \in Act$,*

  $$s_1 \xrightarrow{\alpha} s_1' \text{ implies } s_2 \stackrel{\hat{\alpha}}{\Longrightarrow} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta \; s_2',$$

- *for every $t \in TD$,*

  $$s_1 \xrightarrow{t} s_1' \text{ implies } s_2 \xrightarrow{t} s_2' \text{ for some } s_2' \text{ with } s_1' \; \beta \; s_2',$$

- *$s_2 \stackrel{\hat{\tau}}{\Longrightarrow} s_2'$ for some s_2' such that, denoted $P(s_1) = (\Sigma_1, \mathcal{F}_1, \mu_1)$ and $P(s_2') = (\Sigma_2, \mathcal{F}_2, \mu_2)$, we have that $EC_{\Sigma_1,\beta}(P(s_1)) \approx EC_{\Sigma_2,\beta}(P(s_2'))$*

Two states s_1 and s_2 are weakly bisimilar, denoted by $s_1 \approx s_2$, iff (s_1, s_2) is included in some weak bisimulation. Two RISTTSs $(\mathcal{D}_1, s_{0,1})$ and $(\mathcal{D}_2, s_{0,2})$ are weakly bisimilar, if their initial states $s_{0,1}$ and $s_{0,2}$ are weakly bisimilar in the ISTTS obtained with the disjoint union of $\mathcal{D}_1$ and $\mathcal{D}_2$. ■

In the last item we exploit induced probability spaces (see Definition 6.14) and equivalence between probability spaces (see Definition 6.18) to check that states s_1 and s_2' have the same aggregated probability to reach the same equivalence classes.

6.5 A Semantics for Interactive Generalized Semi-Markov Processes

In this section we present a semantics for well-named interactive generalized semi-Markov processes which maps them onto interactive stochastic timed transition systems. Such a semantics explicitely represents the passage of time by means of transitions labeled with numeric time delays and turns probability distributions of durations into infinitely branching probabilistic choices which lead to states performing numeric time delays with a different duration. In particular, differently from [66] where a technique based on *residual lifetimes of clocks* is used, the states of the semantics of an interactive generalized semi-Markov process encode the *spent lifetimes of clocks*. This means that, in a timed state of the IGSMP where several clocks $C_{n_1} \dots C_{n_k}$ are in execution, the time delay originated by a clock C_{n_i} is determined according to its residual distribution of duration which is evaluated from (i) its associated duration distribution and (ii) its spent lifetime. Once we have sampled a time value t_i from the residual duration distribution of each clock C_{n_i}, we just take the minimum $t_{\min}$ of the sampled values and we consider the clock $C_{n_{\min}}$ which sampled such a time value. Such a "winning clock" is the clock that terminates in the timed state of the IGSMP. After this event the other clocks (which are still in execution) carry over their spent lifetimes, which now is given by $t_i' = t_i + t_{\min}$. Since, according to this approach, the residual duration of a clock is re-sampled in each IGSMP state until it terminates, an adversary (or scheduler) which resolves non-deterministic choices in an IGSMP cannot gain information about the future behavior of the system on which to base its decisions.

Example 6.4 Let us consider the IGSMP depicted in Fig. 6.9, where three temporal delays are started by activating three clocks C_{n_1}, C_{n_2}, and C_{n_3}. In particular, we concentrate on the case in which C_{n_2} is the first clock to terminate.

In Fig. 6.10 we show the semantics of the IGSMP of Fig. 6.9 obtained by following an approach similar to that of [66], which encodes in each state the *residual lifetimes of clocks*. Each state is enriched with the set of active clocks together with their residual lifetimes. In state $\langle s_0, \emptyset \rangle$ (where no clock is active) three numeric time delays t_1, t_2, and t_3 are sampled and associated with the lifetime of the clocks C_{n_1}, C_{n_2}, and C_{n_3}, respectively. Depending on which is the clock $C_{n_{\min}}$ sampling the minimum time value t_{min} in state $\langle s_0, \emptyset \rangle$, we move to one of three different classes of states, one for each possible winning clock. Afterwards, a temporal transition labeled with a numeric time value t between 0 and t_{min} is taken, and each residual duration is accordingly modified by subtracting t_{min} from the residual lifetime of each clock. For the sake of readability in Fig. 6.10 we just depict one trace leading from s_0 to a state s_1 which belongs to the class of states for which C_{n_2} is the winning clock (i.e. t_2 is t_{min}), and then from s_1 to the state s_2 via the transition labeled with the time value t_2, so that in s_2 the clock C_{n_2} is terminated. In state s_2 the residual lifetimes of the remaining active clocks C_{n_1} and C_{n_3} are $t_1 - t_{min}$ and $t_3 - t_{min}$ respectively. By exploiting this information an adversary may already know which clock between C_{n_1} and C_{n_3} will terminate first and consequently guide the nondeterministic choice in state s_2.

In Fig. 6.11 we show the semantics of the IGSMP of Fig. 6.9 obtained by following the approach that we adopt in this paper, which is based on the *spent lifetimes of clocks*. Each state is enriched with: (i) the set of active clocks together with their spent lifetimes, and (ii) a pair $C_n : t$ containing the time value sampled by the winning clock in a timed state of the IGSMP and the clock name. The latter field is set to "$-$" whenever the IGSMP is not in a timed state. The sampling executed in state $\langle s_0, \emptyset, - \rangle$ leads to a state where the three starting clocks are associated with the spent lifetime 0 (because the corresponding transition does not represent a passage of time but simply the result of the sampling), and the winning clock C_n and its sampled value are reported too. As in the case of Fig. 6.10, in Fig. 6.11 we just report one trace leading from s_0 to a state s_1 which belongs to the class of states for which C_{n_2} is the winning clock (i.e. C_{n_2} is $C_{n_{min}}$ and t_2 is its sampled value), and then from s_1 to the state s_2 via the transition labeled with the value t_2, so that in s_2 the clock C_{n_2} is terminated. In state s_2 the spent lifetimes of the remaining active clocks C_{n_1} and C_{n_3} are both equal to t_2, and their residual durations depend on both such a value and the duration distribution

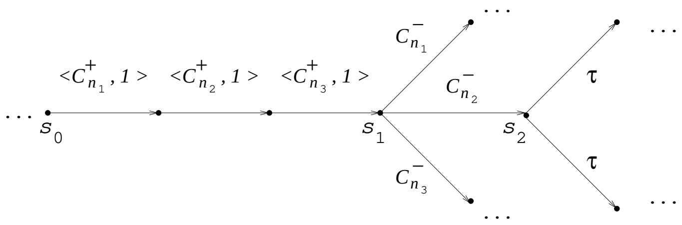


Figure 6.9: Example of an IGSMP

associated with the clocks. Since, according to this approach, the time to termination of clocks C_{n_1} and C_{n_3} is re-sampled, an adversary cannot gain in advance any information about the future behavior of the system and he cannot exploit this information when resolving the nondeterministic choice in state s_2.

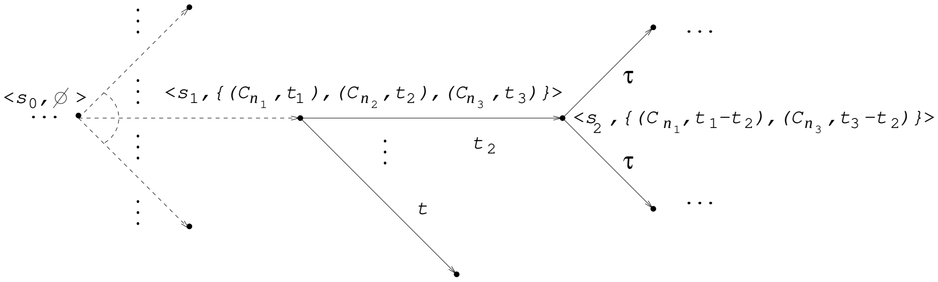


Figure 6.10: Example of semantics based on residual lifetimes

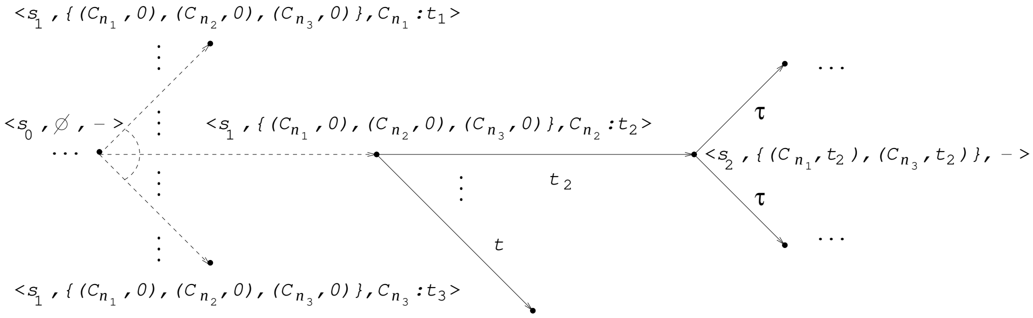


Figure 6.11: Example of semantics based on spent lifetimes

In the following we introduce some preliminary definitions which are needed to define the semantics of IGSMPs.

Definition 6.24 *Given a duration probability distribution $f \in PDF^+$ and a time value $t \in \mathbb{R}^+$, we denote by $[f \mid t]$ the residual duration distribution of a clock C_n with duration distribution f which, after t time units from when it started, has not terminated yet (t is its spent lifetime). More formally, if T is a random variable with distribution f, i.e. $\forall t' \in \mathbb{R}.\ f(t') = P(T \leq t')$, then $[f \mid t]$ is the probability distribution defined* *as follows. For all $t' \in \mathbb{R}$ we have that:*

$$[f \mid t](t') = P(T \leq t' + t \mid T > t)$$

■

Theorem 6.7 *Given $f \in PDF^+$ and $t \in \mathbb{R}^+$, we have that for all $t' \in \mathbb{R}^+ \cup \{0\}$:*

$$[f \mid t](t') = \frac{f(t+t') - f(t)}{1 - f(t)}$$

Proof *It is just sufficient to observe that $\forall t' \in \mathbb{R}^+ \cup \{0\}$. $[f \mid t](t') = P(T \leq t' + t \mid T > t) = P(T \leq t' + t \wedge T > t)/P(T > t) = (f(t+t') - f(t))/(1 - f(t))$* ■

Consider a family of probability distribution functions $f_1, \ldots, f_k \in PDF$. We denote by $\mathcal{R}(f_1, \ldots, f_k)$ the probability space $\prod_{i=1\ldots k}(\mathbb{R}, \mathcal{B}(\mathbb{R}), P_i)$, where P_i is the unique probability measure on $\mathcal{B}(\mathbb{R})$ obtained from f_i (see Theorem 6.4).

Definition 6.25 *Let the residual duration distribution of the set of clocks $C_{n_1}, \ldots, C_{n_k}$ in execution in an IGSMP state be $f_1, \ldots, f_k$, i.e. the probability that a certain tuple of residual durations $(t_1, \ldots, t_k)$ is sampled from the clocks is described by the probability space $\mathcal{R}(f_1, \ldots, f_k)$. For each $I \subseteq \{1, \ldots, k\}$ such that $|I| \geq 2$, the event $Term(I)$ of contemporaneous termination of the clocks $\{C_{n_i} \mid i \in I\}$ in execution is the following measurable subset of the sample space $\mathbb{R}^k$:*

$$Term(I) = \{(t_1, \ldots, t_k) \mid \exists t.\ (\forall i \in I.\ t_i = t) \wedge (\forall i \notin I.\ t_i > t\}$$

■

Since in an IGSMP clocks in execution in a state cannot terminate at the same time instant (see Sect. 2.2.6) we have that each event $Term(I)$ of contemporaneous termination of a subset $\{C_{n_i} \mid i \in I\}$ of the clocks in execution $C_{n_1}, \ldots, C_{n_k}$ occurs with probability 0. More formally, we have that in each state of an IGSMP, if $(\mathbb{R}^k, \mathcal{F}, P)$ is the probability space $\mathcal{R}(f_1, \ldots, f_k)$ expressing the residual duration of the clocks $C_{n_1}, \ldots, C_{n_k}$ in execution in the state, for each $I \subseteq \{1, \ldots, k\}$ such that $|I| \geq 2$, we have $P(Term(I)) = 0$. We exploit this fact in order to reduce the domain of the probability space for a set of active clocks. In particular instead of considering the entire $\mathcal{R}(f_1, \ldots, f_k)$ we can just restrict to consider $\breve{\mathcal{R}}(f_1, \ldots, f_k)$ defined as follows.

Definition 6.26 *$\breve{\mathcal{R}}(f_1, \ldots, f_k)$ is the triple $(\breve{\mathbb{R}}^k, \breve{\mathcal{F}}, \breve{P})$ defined as follows. Let $(\mathbb{R}^k, \mathcal{F}, P)$ be the probability space $\mathcal{R}(f_1, \ldots, f_k)$, then we have:*

- $\breve{\mathbb{R}}^k = \mathbb{R}^k - \bigcup_{I \subseteq \{1,\ldots,k\}, |I| \geq 2} Term(I)$
- $\breve{\mathcal{F}} = \{E \subseteq \mathcal{F} \mid E \subseteq \breve{\mathbb{R}}^k\}$

- $\breve{P} = \{(E,p) \mid E \in \breve{\mathcal{F}} \wedge (E,p) \in P\}$ ■

Theorem 6.8 *Let $(\mathbb{R}^k, \mathcal{F}, P)$ be the probability space $\mathcal{R}(f_1, \ldots, f_k)$. If $\forall I \subseteq \{1, \ldots, k\}.\ |I| \geq 2 \Rightarrow P(Term(I)) = 0$, then $\breve{\mathcal{R}}(f_1, \ldots, f_k)$ is a probability space.*

Proof *Just a trivial consequence of the definition of probability space and of the fact that $Term(I)$ is a measurable set with measure 0.* ■

We are now in a position to formally define the semantics of an IGSMP.

Definition 6.27 *The semantics of an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$ is the RISTTS $[\![\mathcal{G}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, P, T_t, T'_a, s'_0)$ where:*

- *$\Sigma' = (\Sigma \times Spent \times Sample)$ is the set of states of the RISTTS, where Spent, ranged over by v, is the set of partial functions from $\mathcal{C}$ to $\mathbb{R}^+ \cup \{0\}$, expressing the time already spent in execution by the clocks currently in execution in the IGSMP (clocks in the domain of Spent), and Sample, ranged over by sample, is the set $(\mathcal{C} \times (\mathbb{R}^+ \cup \{0\})) \cup \{-\}$, where a pair (C_n, t), also written $C_n : t$, denotes that the IGSMP is currently executing a set of clocks and that clock C_n has sampled the minimum residual time delay with t being the value of such a delay; while "$-$" denotes that started clocks are not under execution (e.g. the IGSMP is in a choice state or in a silent state).*
- *$\mathbb{R}^+ \cup \{0\}$ is the time domain: we consider continuous time.*
- *Act is the set of standard actions considered in the IGSMP.*
- *P, which associates a probability space (expressing next state probability) to some of the states in Σ', is defined to be the least partial function on Σ' satisfying the operational rules in the first part of Table 6.2.*
- *T_t is the set of timed transitions which are defined as the least relation over $\Sigma' \times (\mathbb{R}^+ \cup \{0\}) \times \Sigma'$ satisfying the operational rules in the second part of Table 6.2.*
- *T'_a is the set of action transitions which are defined as the least relation over $\Sigma' \times Act \times \Sigma'$ satisfying the operational rules in the third part of Table 6.2.*
- *$s'_0 = \langle s_0, \emptyset, - \rangle$ is the initial state of the RISTTS, where the IGSMP is in the initial state and no clock is in execution.* ■

In Table 6.2 we make use of the following notation. Given $v \in Spent$, we define $v - C_n$ to be the partial function obtained from v by removing C_n (and the associated value) from its domain. We define $v + t$, with $t \in \mathbb{R}^+ \cup 0$, to be the partial function obtained from v by adding t to the time value associated with each clock in the domain of v. We use the notation $\{\mathbf{n_i}\}$ to stand for $\{\mathbf{n_i}\}_{\mathbf{i=1\ldots k}}$, representing the sequence of names $n_1, \ldots n_k$ (in Table 6.2 the length k of the sequence is always clarified by the context in which $\{\mathbf{n_i}\}$ is used). Finally in the fourth part of Table 6.2 we define two auxiliary functions. The function $TW : \Sigma \longrightarrow \mathbb{R}^+ \cup \{0\}$

$$(\mathbf{P1})\quad \frac{(\exists C_n.\ s \xrightarrow{C_n^-}) \ \wedge\ \{C_{n_1},\ldots,C_{n_k}\} = dom(v)}{P(\langle s,v,-\rangle) = Sample_{s,v}^{\{\mathbf{n_i}\}}(\ \breve{\mathcal{R}}([D(C_{n_1}) \mid v(C_{n_1})],\ldots,[D(C_{n_k}) \mid v(C_{n_k})])\ )}$$

$$(\mathbf{P2})\quad \frac{(\exists C_n, w.\ s \xrightarrow{<C_n^+,w>}\ ) \ \wedge\ Pr = \{\ (\ <C_n,s'>,\ w/TW(s)\ )\ \mid\ s \xrightarrow{<C_n^+,w>} s'\ \}}{P(\langle s,v,-\rangle) = \sum_{<C_n,s'>\in dom(Pr)} Pr(<C_n,s'>) \cdot P(\langle s', v \cup \{(C_n,0)\}, -\rangle)}$$

$$(\mathbf{T1})\quad \langle s,v,C_n\!:\!t\rangle \xrightarrow{t'} \langle s, v+t', -\rangle \qquad 0 \le t' < t$$

$$(\mathbf{T2})\quad \frac{s \xrightarrow{C_n^-} s'}{\langle s,v,C_n\!:\!t\rangle \xrightarrow{t} \langle s', (v - C_n) + t, -\rangle}$$

$$(\mathbf{T3})\quad \frac{(\not\exists\theta.\ s \xrightarrow{\theta}) \ \wedge\ s \not\xrightarrow{\tau}}{\langle s,\emptyset,-\rangle \xrightarrow{t} \langle s,\emptyset,-\rangle} \qquad t \ge 0$$

$$(\mathbf{A1})\quad \frac{s \xrightarrow{\alpha} s'}{\langle s,v,-\rangle \xrightarrow{\alpha} \langle s',v,-\rangle} \qquad\qquad (\mathbf{A2})\quad \frac{s \xrightarrow{a} s'}{\langle s,v,C_n\!:\!t\rangle \xrightarrow{a} \langle s',v,-\rangle}$$

$$TW(s) = \sum\{\!|\ w \mid \exists C_n.\ s \xrightarrow{<C_n^+,w>}\ |\!\}$$

$$Sample_{s,v}^{\{\mathbf{n_i}\}}(t_1,\ldots,t_k) = \langle s,v,C_{n_{min}}\!:\!t_{min}\rangle$$

where min is the only index i such that: $\mathrm{t_i} = \min\limits_{\mathrm{j}\in\{1,\ldots,\mathrm{k}\}} \mathrm{t_j}$

Table 6.2: Semantic rules for IGSMPs

computes the overall weight of the clock start transitions leaving a state of an IGSMP. Moreover, given a state of the IGSMP $s \in \Sigma$, a partial function mapping active clock into their spent lifetimes $v \in \mathit{Spent}$, and a sequence $\{n_1, \ldots, n_k\}$ of clock indexes, the function $\mathit{Sample}^{\{\mathbf{n_i}\}}_{s,v}$ maps a tuple $(t_1, \ldots, t_k)$ of time values sampled by clocks into the corresponding state $\langle s, v, C_{n_{\min}} : t_{\min}\rangle$ reached in the RISTTS, where min is the index of the clock which sampled the least time value. Note that function $\mathit{Sample}^{\{\mathbf{n_i}\}}_{s,v}$ is used in Table 6.2 for deriving (via induction, see Definition 6.14) a probability space over the states of the RISTTS from the probability space $\breve{\mathcal{R}}([D(C_{n_1}) \mid v(C_{n_1})], \ldots, [D(C_{n_k}) \mid v(C_{n_k})])$ over residual durations sampled by active clocks in a state of the IGSMP. Finally, we assume that, for every distribution f, $[f|0]$ is defined so that it yields the distribution f itself.

Theorem 6.9 *Let $\mathcal{G}'$, $\mathcal{G}''$ be two well-named IGSMPs. If $\mathcal{G}' \approx \mathcal{G}''$ then $[\![\mathcal{G}']\!] \approx [\![\mathcal{G}'']\!]$.*

Proof *It is just sufficient to show that from a weak probabilistic bisimulation β between the two well-named IGSMPs $\mathcal{G}'$ and $\mathcal{G}''$ we can derive a weak probabilistic bisimulation β' between the two RISTTSs $[\![\mathcal{G}']\!]$ and $[\![\mathcal{G}'']\!]$. In particular β' is obtained by letting $(s_1, v, sample)\ \beta'\ (s_2, v, sample)$ whenever $s_1 \beta s_2$ and $v \in Spent$, $sample \in Sample$. The fact that β' is a weak probabilistic bisimulation simply derives from the fact that equivalent well-named IGSMP states always have the same set of clocks under execution (with the same name and therefore also with the same duration distribution).* ■

The following theorems show that the semantics of well-named IGSMPs is indeed compositional.

Theorem 6.10 *Let $\mathcal{G}'$, $\mathcal{G}''$ be two well-named IGSMPs. For each $S \subseteq Act - \{\tau\}$ we have $[\![\mathcal{G}']\!] \,\|_S [\![\mathcal{G}'']\!] \approx [\![\mathcal{G}' \,\|_S\, \mathcal{G}'']\!]$.*

Proof *It is just sufficient to show that we can build a weak probabilistic bisimulation β between the two RISTTSs $[\![\mathcal{G}']\!] \,\|_S [\![\mathcal{G}'']\!]$ and $[\![\mathcal{G}' \,\|_S\, \mathcal{G}'']\!]$. In particular β is obtained by letting $((s', v', sample'), (s'', v'', sample''))\ \beta\ ((s', s'', M), v,\ sample)$ whenever (i) the spent lifetimes of clocks in v' and v'' are the same as those in v once clocks are renamed according to the mapping function M, and (ii) either both $sample'$ and $sample''$ are equal to "$-$" and $sample$ is also equal to "$-$", or they are equal to $(C_{n'}, t')$ and $(C_{n''}, t'')$, respectively, with $t' \neq t''$ and $sample$ is $(C_{n'}, t')$ if $t' \leq t''$, while is $(C_{n''}, t'')$ otherwise. The fact that β' is a weak probabilistic bisimulation derives from the fact that the set of clocks executable in a state of an IGSMP obtained as the parallel composition of two IGSMPs is the union of the clocks executable by the two IGSMPs. Therefore the probability space generated from such a state of the composed model turns out to be equivalent to the product of the probability spaces independently generated from the states of the two IGSMPs. This because for each fixed $s', v', C'_n, t', s'', v''$ all states $((s', v', C_{n'} : t'), (s'', v'', C_{n''} : t''))$ such that $t'' > t'$ turn out to be equivalent due to the synchronization of timed transitions in parallel composition of RISTTSs. Moreover, when composing in parallel IGSMPs τ transitions pre-empt performance-related transitions and clock start transitions pre-empt clock termination transitions just as in RISTTSs τ transitions pre-empt performance-related transitions and probabilistic transitions pre-empt timed transitions.* ■

Theorem 6.11 *Let $\mathcal{G}$ be a well-named IGSMP. For each $L \subseteq Act - \{\tau\}$ we have $[\![\mathcal{G}]\!]/L \approx [\![\mathcal{G}/L]\!]$.*

Proof *It is just sufficient to show that we can build a weak probabilistic bisimulation β between the two RISTTSs $[\![\mathcal{G}]\!]/L$ and $[\![\mathcal{G}/L]\!]$. In the case of the hiding operator the structure of the states of the two RISTTSs turns out to be the same and β is just obtained by considering the identity relation, i.e. we relate each state $(s, v, sample)$ to itself. The fact that β' is a weak probabilistic bisimulation simply derives from the fact that the only effect of hiding standard actions is to preempt the execution of time delays both in IGSMPs and in RISTTSs. In particular the fourth constraint of Definition 6.1 guarantees that in an IGSMP a sequence of clock start events leading to clock execution is either globally preempted or not preempted at all by an hiding operation.* ■

6.6 Conclusion

Dealing with non-determinism in probabilistic systems with general distributions raises a series of problems, including the correct management of the residual durations of generally distributed delays in system states and the interplay of non-deterministic choices and probabilistic behaviors of temporal delays. The former problem can be solved by representing the temporal behavior of a system by using clocks (as in Timed Automata [132]) or elements (as in GSMPs [121]) whose durations are associated with generally distributed random variables. In particular we can correctly manage the residual durations of active clocks (elements) in system states with two different approaches which are borrowed from the theory of GSMPs: one based on spent clock lifetimes and one based on residual clock lifetimes. In this chapter we have shown how to apply the former approach to the specification and analysis of concurrent systems including generally distributed delays, instead of using the latter approach as previously done in the literature [66]. In the former approach (similarly as in Timed Automata [132]) states are enriched with spent lifetimes of clocks and for each timed state where a clock C_n is active, the residual duration of C_n is sampled depending on both its spent lifetime and its associated duration distribution. In the latter approach (similarly as in classical discrete event simulation [54]), the lifetime of each clock is sampled all at once at the clock start event, and states are enriched with residual lifetimes of clocks directly determining their residual duration. As we have shown, the drawback of the approach based on residual lifetimes, with respect to the one we propose, is that an adversary which is in charge of solving non-determinism may get information about the future system behavior since the duration of delays is decided a priori.

Our approach has been formalized by introducing (well-named) Interactive GSMPs and Interactive Stochastic Timed Transition Systems (ISTTSs), which are both closed with respect to CSP parallel and hiding operators. Then we have used ISTTSs to define a compositional semantics for IGSMPs which realizes the approach based on spent lifetimes mentioned above.

Chapter 7

Calculus of Interactive Generalized Semi-Markov Processes

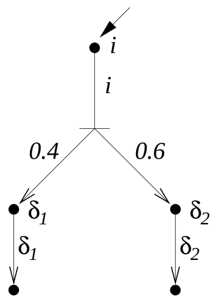
In this chapter we introduce the calculus of interactive generalized semi-Markov processes, a stochastic process algebra which can express probabilistic timed delays with general distributions and synchronizable actions with zero duration. The calculus of *IGSMPs* is equipped with a structural operational semantics which generates semantic models in the form of well-named *IGSMPs*, the interactive extension of *GSMPs* presented in the previous Chapter 6. This is obtained by expressing the concurrent execution of delays through a simple probabilistic extension of ST semantics based on dynamic names (see Chapter 3). We also show that the notion of observational congruence over *IGMSP*, obtained by modifying weak bisimulation over *IGSMPs* (introduced in the previous Chapter 6) in the standard way [122], is a congruence for all the operators of the calculus over strongly guarded processes and we produce an axiomatization for this equivalence which is complete over finite-state strongly guarded processes. Finally, we present a case study on queuing systems $G/G/1/q$.

7.1 Introduction

Some previous work has been done in order to extend the expressiveness of Markovian process algebras to probabilistic time with general distributions [87, 7, 49, 90, 139, 113]. The main point in doing this is to understand how to define the algebra operational semantics and semantic reasoning, e.g. the definition of an adequate notion of bisimulation based equivalence.

7.1.1 The Basic Idea

Once recognized that we have to produce a system representation like that of the IGSMP model presented in Chapter 6 when dealing with general distributions, an important issue is how to develop a calculus suitable for generating IGSMPs, so to have a fully compositional approach. The problem of developing a semantics for



$$PDF(\delta_1) = f, \quad PDF(\delta_2) = g, \quad PDF(i) = Det(0)$$

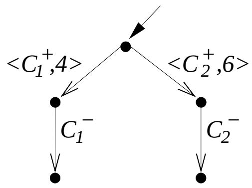
Figure 7.1: Choice of Generally Distributed Delays

a process algebra with generally distributed delays (expressed, e.g., by $f.\_$ prefixes, where f is a probability distribution) is essentially the problem of representing the execution of a temporal delay as the combination of a start and a termination event in such a way that pair of events derived from the execution of a delay are somehow related, e.g. by deriving a unique clock name for the delay (see Chapter 6). As we recognized in [39] such problem is not new in the literature, but exactly corresponds to consider classical *ST semantics* (see Chapter 3) for delays. With respect to the definition of ST semantics the "type" of a delay is simply its duration distribution f and what we observe of a system is its ability of performing delays of certain types f. In particular *identification* of delays by means of clock names is obtained if we use a technique for expressing ST semantics based on names (like the static and dynamic name techniques presented in Chapter 3) and not, e.g., on pointers [52]. The use of such a semantics cause clock names to be generated for delays by the semantics, thus obtaining models with clocks like IGSMPs.

Another important issue to address when developing a semantics for a process algebra with generally distributed delays is how to interpret a choice "$\_ + \_$" between delays. Our claim (introduced in [41]) is that, while in the "Markovian world" the most natural solution is to adopt a *race policy* where the choice is solved in favor of the delay which terminates first, when we consider general distributions a convenient solution is to adopt a *preselection policy* where, first one of the delays is selected according to a probabilistic choice, and then the selected delay is executed. A justification for this claim can be found in the difference between the structure of GSMPs, where probabilistic choices are explicitely represented, and CTMCs, where probabilistic choices are implicitly expressed through races between exponential delays. For example an intuitive semantics of $f.\underline{0} + g.\underline{0}$, where f and g are generally distributed delays, should generate a GSMP like that of Fig. 7.1, where the element i is an auxiliary dummy element with zero duration which causes the process to leave immediately the first state. [1]

The GSMP of Fig. 7.1 first performs a probabilistic choice between f and g (e.g. according to probabilities

[1] An equivalent representation of the GSMP of Fig. 7.1 which allows us to leave out the element i can be obtained by associating directly with the states labeled with δ_1 and δ_2 the probability of being the initial state (see Chapter 2).



$$PDF(C_1) = f, \qquad PDF(C_2) = g$$

Figure 7.2: Event-Based Representation of Choice of Delays

0.4 and 0.6) and then executes the selected delay. This corresponds to using the *preselection policy* instead of the race policy to solve choices. Obviously we must somehow syntactically express the probability associated with delays in a choice (in the GSMP above 0.4 and 0.6 for f and g, respectively). This can be done by using delay prefixes $<f, w>.\_$ in the algebra, so that the weight w determines the probability of selecting the delay (similarly as for weight prefixes of IWMCs in Chapter 4). In this way the GSMP would be obtained from, e.g., $<f,4>.\underline{0} + <g,6>.\underline{0}$. Notably, such a preselection policy is naturally obtained from ST semantics applied to $<f, w>.\_$ prefixes by associating the weight w with the transition representing the delay start. For instance in the case of $<f,4>.\underline{0} + <g,6>.\underline{0}$ we would obtain the IGSMP of Fig. 7.2 whose performance model (see Chapter 6) is the GSMP of Fig. 7.1.

Some previous efforts [87, 7, 139] have been made in order to try to adapt the interleaving semantics to deal with general distributions, instead of splitting delays in starting and terminating events as in the ST semantics. When doing this, the basic idea is considering actions as starting in the first state they become enabled. In order for the starting point of actions to be observable the interleaving semantics had to be enriched with additional information: in [87] transitions are enriched with *start references*, in [139] transitions are enriched with information about causality relations among actions, and in [7] actions must be differently identified with indexes before the semantic rules are applied. As a matter of fact these semantics are not actually interleaving since, even if they consider choices to be resolved with race policy, the expansion law $a.\underline{0} \,||\, b.\underline{0} = a.b.\underline{0} + b.a.\underline{0}$ is no longer valid, due to the fact that distributions are not memoryless. Hence there is no advantage in trying to keep the semantics in an interleaving atomic form and to preserve the rather tricky race policy for the alternative composition operator "$\_ + \_$".

On the contrary, if we use ST semantics we also obtain an expansion law for generally distributed delays at the level of clock start and clock termination events which allow us to produce a complete axiomatization over finite-state algebraic specifications of the same kind of that presented for IWMCs in Chapter 4.

7.1.2 A Semantics for Timed Concurrent Systems with General Distributions

Similarly as for the calculus of IWMCs (see Chapter 4), the process algebra that we will consider in this chapter (called calculus of IGSMPs) is an extension of the basic process algebra introduced in Chapter 2 with prefixes "$<f, w>.\_$", representing timed delays with general probability duration distribution f and associated weight w, which produces well-named IGSMPs (see Chapter 6) as semantic models of its terms.

As we already explained in Sect. 7.1.1, we define the operational semantics of a delay f through a realization of ST semantics based on names. In Chapter 3 we have introduced two name based techniques for expressing ST semantics. With the first one ST semantics is expressed by assigning *static names* to delays, i.e. names determined according to their syntactical position in the system, while in the second one we use *dynamic names*, i.e. names computed dynamically while the system evolves. When defining the semantics of the calculus of IGSMPs, we use the dynamic name technique which, with respect to the static name technique, has the advantage of generating semantic models such that ST bisimulation can be simply established via the standard notion of observational congruence [122]. In particular, the use of the dynamic name technique allows us to derive semantic models in the form of well-named IGSMPs, for which equivalence is defined as a simple extension of probabilistic bisimulation [117] (so that existing results and tools can be exploited). On the contrary, using static names for the semantics of generally distributed delays (as we will do in Chapter 13) requires a more complex definition of bisimulation which associates the names of the delays of one process with the names of the corresponding delays used by the other one.

A delay is represented in semantic models as a combination of the event of start of the delay f^+ and the event of termination of the delay f^-. Moreover, according to the dynamic name technique, we dynamically assign indexes i to delays so that the execution of a delay is represented by the two events f_i^+ and f_i^- and no confusion arises (in the connection between delay starts and delay terminations) when multiple delays with the same distribution f are concurrently executed. We resolve choices among several delays by means of preselection policy. In a choice a delay is selected with probability proportional to its associated weight. For instance $<f, w>.\underline{0} + <g, w'>.\underline{0}$ represents a system which performs a delay of distribution f with probability $w/(w + w')$ and a delay of distribution g with probability $w'/(w + w')$. Choices are expressed in semantic models by associating weights to transitions f_i^+ representing the start of a delay. With respect to the definition of well-named IGSMPs of Chapter 6, in this chapter we will use "$<f_i^+, w>$" and "f_i^-" as shorthands for "$<C_{f,i}^+, w>$" and "$C_{f,i}^-$", respectively, in transition labels.

The semantics of standard actions a (including internal actions τ) in *IGSMP* is, instead, just the standard interleaving semantics. This reflects the fact that these actions have zero duration and can be considered as being executed atomically.

In general, the semantics of terms of the calculus of IGSMPs is simply defined via SOS by extending the standard interleaving semantics for action prefixes presented in Chapter 2 with ST semantics via dynamic names for delay prefixes "$<f, w>$". This can be done easily with the technique introduced in Chapter 3 which is compositional. Moreover, the interrelation among the three kind of derived transitions is captured in the semantics by: (i) applying a notion of priority of delay start transitions over delay termination transitions due to the assumption that probabilistic choices are resolved urgently, and (ii) applying a notion of priority of τ transitions over performance related transitions due to the maximal progress assumption. We also show that the notion of observational congruence over terms of the calculus, obtained by modifying weak bisimulation over well-named *IGSMPs* defined in Chapter 6 in the standard way [122], is a congruence for all the operators of the calculus over strongly guarded processes and we produce an axiomatization for this equivalence which is complete over finite-state strongly guarded processes.

As an example of semantic reasoning with the calculus of *IGSMPs*, in this chapter we will consider Queuing Systems $G/G/1/q$, i.e. queuing systems with one server and a FIFO queue with q-1 seats, where interarrival time and service time are generally distributed. Moreover we show how to derive the performance model of such queuing systems (a *GSMP*) by applying the formal procedure which turns IGSMPs into GSMPs presented in Chapter 6.

7.1.3 Chapter Outline

The chapter is structured as follows. In Sect. 7.2 we present the calculus of Interactive *GSMPs* and its operational semantics. In Sect. 7.3 we present the notion of observational congruence and its complete axiomatization. In Sect. 7.4 we present the example of Queuing Systems $G/G/1/q$. Finally, in Sect. 7.5 we report some concluding remarks including comparison with related work.

7.2 The Calculus of Interactive GSMPs

7.2.1 Syntax of Terms and Informal Semantics of Operators

The calculus of interactive *GSMPs* is an extension of the basic process algebra with operators of CCS/CSP [122, 109] introduced in Chapter 2, [2] which allows us to express priority, probabilistic choices and probabilistic delays with arbitrary distributions. This is done by including into the calculus, in addition to standard actions, a special kind of actions representing delays. Delays are represented as $<f, w>$ and are characterized by a *weight* w and a *duration distribution* f. The weight w determines the probability of choosing the delay in a choice among several delays. The set of weights is $\mathbb{R}^+$, ranged over by $w, w', \ldots$. The duration distribution f denotes the probability distribution function of the delay duration. The set of duration probability distribution functions is PDF^+, i.e. the set of probability distribution functions f such that $f(x) = 0$ for $x < 0$, ranged over by f, g, h. The possibility of expressing priority derives from the interrelation of delays and standard τ actions. In particular we make the maximal progress assumption: the system cannot wait if it has something internal to do. Therefore we assume that, in a choice, τ actions have priority over delays, i.e. $\tau.P + <f, w>.Q$ behaves as $\tau.P$.

Let Act be the set of action types containing a distinguished type τ representing an internal computation. Act is ranged over by $\alpha, \alpha', \ldots$. $Act - \{\tau\}$ is ranged over by $a, b, \ldots$. Let $TAct = \{<f, w> \mid f \in PDF^+ \wedge w \in$

[2] To be precise, with respect to the algebra considered in Chapter 2, in the initial part of this section we use the recursion operator "$recX.\_$", instead of process constants, to denote recursion.

$\mathbb{R}^+\}$ be the set of delays. [3] Let *Var* be a set of process variables ranged over by X, Y, Z. Let $ARFun = \{\varphi : Act \longrightarrow Act \mid \varphi(\tau) = \tau \wedge \varphi(Act - \{\tau\}) \subseteq Act - \{\tau\}\}$ be a set of *action relabeling functions*, ranged over by φ.

Definition 7.1 *We define the language IGSMP as the set of terms generated by the following syntax*

$$P ::= \underline{0} \mid X \mid <f,w>.P \mid \alpha.P \mid P + P \mid P/L \mid P[\varphi] \mid P \parallel_S P \mid recX.P$$

where $L, S \subseteq Act - \{\tau\}$. *An IGSMP process is a closed term of IGSMP. We denote by* $IGSMP_g$ *the set of strongly guarded terms of IGSMP.* [4] ■

"$\underline{0}$" denotes a process that cannot move. The operators "." and "+" are the CCS prefix and choice. The choice among delays is carried out through the preselection policy by giving each of them a probability proportional to its weight. Note that alternative delays are not executed concurrently, first one of them is chosen probabilistically and then the selected delay is executed. Moreover τ actions have priority over delays in a choice. "$/L$" is the hiding operator which turns into τ the actions in L, "$[\varphi]$" is the relabeling operator which relabels visible actions according to φ. "$\parallel_S$" is the CSP parallel operator, where synchronization over actions in S is required. Finally "$recX$" denotes recursion in the usual way.

Similarly as for the calculus of IWMCs (see Chapter 4), in this chapter we will consider only terms of $IGSMP_g$, i.e. strongly guarded terms of our calculus. We will discuss the treatment of weakly guarded processes in Chapter 9.

7.2.2 Operational Semantics

As explained in Sect. 7.1, we employ the dynamic name technique of Chapter 3 for giving semantics to generally distributed delays f.

The problem of preserving the relationship between starts and terminations of delays arises, like in the ST semantics of standard process algebras, when several delays of the same type f are being executed in parallel. When a delay f terminates (event f^-) we need some information for establishing which event of delay start (f^+) it refers to. By using the dynamic name technique introduced in Chapter 3 this is obtained by generating a fresh name f_i for each starting delay f, where $i \in \mathbb{N}$ is the minimum index not already used by the other delays with distribution f that have started but not yet terminated, thus obtaining names for clocks in the format required by well-named IGSMPs. In particular, due to the adoption of preselection policy (see Sect. 7.1.1) starting events generated by delay prefixes $<f,w>$ are represented in semantic models by a transitions labeled by $<f_i^+,w>$, where i is determined as explained above. The termination of a delay prefix $<f,w>$ is simply represented by a transition labeled by f_i^-, where the "identifier" i uniquely determines which delay f is terminating.

[3] In the following we consider f to be a shorthand for $<f,1>$ when this is clear from the context.

[4] We consider the delay $<f,w>$ as being a guard in the definition of strong guardedness.

Moreover the dynamic name technique introduced in Chapter 3 allows us to dynamically assign names to delays, according to the rule formerly described, via SOS semantics (hence in a *compositional way*) through the idea of *levelwise renaming*. In order to obtain structural compositionality it is necessary to determine at each syntactical level of a term, e.g. in the case of the parallel composition operator, the computations of $P \,\|\, Q$ from the computations of P and Q. This is done, as described in Chapter 6 when composing in parallel well-named IGSMPs, by suitably renaming delays occurring in the computations of P and Q. In particular, in the operational semantics the mapping M recording delay renamings for a certain parallel operator (see Sect. 6.2.3) is expressed as an additional parameter of that parallel operator in state terms. For every delay f started by $P \,\|_{S,M}\, Q$, M records the association between the name f_i, generated according to the well-naming rule for identifying f at the level of $P \,\|_{S,M}\, Q$, and the name f_j (which in general is different from f_i), generated according to the well-naming rule for identifying the same delay f inside P (or Q). In this way when, afterwards, such a delay f terminates in P (or Q) the name f_j can be re-mapped to the correct name f_i at the level of $P \,\|_{S,M}\, Q$, by exploiting the information included in M. As in Sect. 6.2.3, in M the delay f of $P \,\|_{S,M}\, Q$ which gets index i is uniquely identified by expressing the unique name j it gets in P or in Q and the "location" of the process that executes it: *left* if P, *right* if Q. Such an association is represented inside M by the triple (f, i, loc_j) with $f \in \mathit{PDF}^+$, indices $i, j \in \mathbb{N}^+$ and location $loc \in \mathit{Loc} = \{l, r\}$, where "$l$" stands for left and "$r$" for right. In the following we use $f : (i, loc_j)$ to stand for $(f, i, loc_j) \in M$.

In order to define the operational semantics for the processes of IGSMP, we need a richer syntax to represent states. Let $\mathit{TAct}^+ = \{<f_i^+, w> \mid f \in \mathit{PDF}^+ \wedge i \in \mathbb{N}^+ \wedge w \in \mathbb{R}^+\}$ be the set of delay starts, where $<f_i^+, w>$ represents the beginning of the delay $<f, w>$ identified by i. [5] Besides let $\mathit{TAct}^- = \{f_i^- \mid f \in \mathit{PDF}^+ \wedge i \in \mathbb{N}^+\}$ be the set of delay terminations, where f_i^-, represents the termination of the delay with duration distribution f identified by i. η ranges over $\mathit{Act} \cup \mathit{TAct} \cup \mathit{TAct}^+ \cup \mathit{TAct}^-$. We denote an *index association*, whose elements are associations (i, loc_j), with *iassoc* which ranges over the set *IAssoc* of partial bijections from $\mathbb{N}^+$ to $\mathit{Loc} \times \mathbb{N}^+$. Finally a mapping M is a relation from PDF^+ to $\mathbb{N}^+ \times (\mathit{Loc} \times \mathbb{N}^+)$ such that $\forall f \in \mathit{PDF}^+ . M_f \in \mathit{IAssoc}$ [6], i.e. M is a set including an independent index association for each different duration distribution.

The set IGSMP_s of state terms of IGSMP is generated by the following syntax:

$$P ::= \underline{0} \mid X \mid \eta.P \mid P + P \mid P/L \mid P[\varphi] \mid P \,\|_{S,M}\, P \mid \mathit{rec}X.P$$

We denote by IGSMP_{sg} the set of strongly guarded terms of IGSMP_s. We consider the operators "$\|_S$" occurring in a IGSMP term P as being "$\|_{S,\emptyset}$" when P is regarded as a state.

The semantics of state terms produces a transition system labeled over $\mathit{Act} \cup \mathit{TAct}^+ \cup \mathit{TAct}^-$, ranged over by $\gamma, \gamma', \ldots$. Such a transition system is defined as being the well-named *IGSMTS* [7] (see Chapter 6) $\mathcal{G} = (\mathit{IGSMP}_{sg}, \mathit{Act}, T_+, T_-, T_a)$, where: T_a is the least subset of $\mathit{IGSMP}_{sg} \times \mathit{Act} \times \mathit{IGSMP}_{sg}$ satisfying the standard

[5] In the following we consider f_i^+ to be a shorthand for $<f_i^+, 1>$ when this is clear from the context.

[6] Given a relation M from A to B, we denote with M_a the set $\{b \in B \mid (a, b) \in M\}$.

[7] We recall that we consider f_i^+ and f_i^- as being shorthands for $C^+_{f,i}$ and $C^-_{f,i}$ in the transitions of the IGSMTS.

$$\alpha.P \xrightarrow{\alpha} P$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \parallel_{S,M} Q \xrightarrow{\alpha} P' \parallel_{S,M} Q}\ \alpha \notin S \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel_{S,M} Q \xrightarrow{\alpha} P \parallel_{S,M} Q'}\ \alpha \notin S$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q'}{P \parallel_{S,M} Q \xrightarrow{a} P' \parallel_{S,M} Q'}\ a \in S$$

$$\frac{P \xrightarrow{a} P'}{P/L \xrightarrow{\tau} P'/L}\ a \in L \qquad \frac{P \xrightarrow{\alpha} P'}{P/L \xrightarrow{\alpha} P'/L}\ a \notin L$$

$$\frac{P \xrightarrow{\alpha} P'}{P[\varphi] \xrightarrow{\varphi(\alpha)} P'[\varphi]} \qquad \frac{P\{recX.P/X\} \xrightarrow{\alpha} P'}{recX.P \xrightarrow{\alpha} P'}$$

Table 7.1: Standard Rules

$$<f,w>.P \xrightarrow{<f_i^+,w>} f_i^-.P \qquad <f_i^+,w>.P \xrightarrow{<f_i^+,w>} P$$

$$\frac{P \xrightarrow{<f_i^+,w>} P' \wedge Q \not\xrightarrow{\tau}}{P + Q \xrightarrow{<f_i^+,w>} P'} \qquad \frac{Q \xrightarrow{<f_i^+,w>} Q' \wedge P \not\xrightarrow{\tau}}{P + Q \xrightarrow{<f_i^+,w>} Q'}$$

$$\frac{P \xrightarrow{<f_i^+,w>} P' \wedge Q \not\xrightarrow{\tau}}{P \parallel_{S,M} Q \xrightarrow{<f_{n(M_f)}^+,w>} P' \parallel_{S,M\cup\{f:(n(M_f),l_i)\}} Q} \qquad \frac{Q \xrightarrow{<f_i^+,w>} Q' \wedge P \not\xrightarrow{\tau}}{P \parallel_{S,M} Q \xrightarrow{<f_{n(M_f)}^+,w>} P \parallel_{S,M\cup\{f:(n(M_f),r_i)\}} Q'}$$

$$\frac{P \xrightarrow{<f_i^+,w>} P' \wedge \nexists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{<f_i^+,w>} P'/L} \qquad \frac{P \xrightarrow{<f_i^+,w>} P'}{P[\varphi] \xrightarrow{<f_i^+,w>} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{<f_i^+,w>} P'}{recX.P \xrightarrow{<f_i^+,w>} P'}$$

Table 7.2: Rules for Start Moves

$$f_i^-.P \xrightarrow{f_i^-} P$$

$$\frac{P \xrightarrow{f_i^-} P' \,\wedge\, Q \stackrel{\tau}{\not\longrightarrow} \,\wedge\, Q \stackrel{<f_k^+,w>}{\not\longrightarrow}}{P + Q \xrightarrow{f_i^-} P'} \qquad \frac{Q \xrightarrow{f_i^-} Q' \,\wedge\, P \stackrel{\tau}{\not\longrightarrow} \,\wedge\, P \stackrel{<f_k^+,w>}{\not\longrightarrow}}{P + Q \xrightarrow{f_i^-} Q'}$$

$$\frac{P \xrightarrow{f_i^-} P' \,\wedge\, Q \stackrel{\tau}{\not\longrightarrow} \,\wedge\, Q \stackrel{<f_k^+,w>}{\not\longrightarrow}}{P \,\|_{S,M}\, Q \xrightarrow{f_j^-} P' \,\|_{S,M-\{f:(j,l_i)\}}\, Q}\; f:(j,l_i) \in M \qquad \frac{Q \xrightarrow{f_i^-} Q' \,\wedge\, P \stackrel{\tau}{\not\longrightarrow} \,\wedge\, P \stackrel{<f_k^+,w>}{\not\longrightarrow}}{P \,\|_{S,M}\, Q \xrightarrow{f_j^-} P \,\|_{S,M-\{f:(j,r_i)\}}\, Q'}\; f:(j,r_i) \in M$$

$$\frac{P \xrightarrow{f_i^-} P' \wedge \not\exists a \in L.\, P \xrightarrow{a}}{P/L \xrightarrow{f_i^-} P'/L} \qquad \frac{P \xrightarrow{f_i^-} P'}{P[\varphi] \xrightarrow{f_i^-} P'[\varphi]}$$

$$\frac{P\{recX.P/X\} \xrightarrow{f_i^-} P'}{recX.P \xrightarrow{f_i^-} P'}$$

Table 7.3: Rules for Termination Moves

operational rules of Table 7.1, T_+ is obtained from the least multiset over $IGSMP_{sg} \times TAct^+ \times IGSMP_{sg}$ satisfying the operational rules of Table 7.2 (similarly to [107], we consider a transition to have arity m if and only if it can be derived in m possible ways from the operational rules) by summing the weights of the multiple occurrencies of the same transition, and T_- is the least subset of $IGSMP_{sg} \times TAct^- \times IGSMP_{sg}$ satisfying the operational rules of Table 7.3. In Tables 7.2 and 7.3 we use $P \xrightarrow{a}$ to stand for $\exists P' : P \xrightarrow{a} P'$, $P \stackrel{\tau}{\not\longrightarrow}$ to stand for $\not\exists Q : P \xrightarrow{\tau} Q$ and $P \stackrel{<f_k^+,w>}{\not\longrightarrow}$ to stand for $\not\exists f, w, k, Q : P \xrightarrow{<f_k^+,w>} Q$.

The rules of Table 7.2 define the transitions representing the start of a delay, by taking into account the priority of "τ" actions over delays. In particular the rules for parallel composition and hiding are exactly as those defined in Chapter 6. As far as the operator $P \,\|_{S,M}\, Q$ is concerned, if P performs f_i^+ then a new index $n(M_f)$ is determined for identifying the delay f at the level of "$\|_{S,M}$" and the new association $f:(n(M_f), l_i)$ is added to M. The function $n : IAssoc \longrightarrow \mathbb{N}^+$ computes the new index to be used for identifying the delay f that is starting execution by choosing the minimum index not used by the other delays f already in execution: $n(iassoc) = min\{k \mid k \notin dom(iassoc)\}$. A symmetric mechanism takes place for a move f_i^+ of Q.

The rules of Table 7.3 define the transitions representing the termination of a delay, by taking into account the priority of "τ" actions over delay related transitions and the priority of delay starts over delay terminations. In particular the rules for parallel composition and hiding are exactly as those defined in Chapter 6. As far as the operator $P \,\|_{S,M}\, Q$ is concerned, when P performs f_i^- the delay f with index j associated to l_i in M terminates at the level of the parallel operator. A symmetric mechanism takes place for a move f_i^- of Q.

Note that even if the operational rules in Tables 7.2 and 7.3 include negative premises, this does not

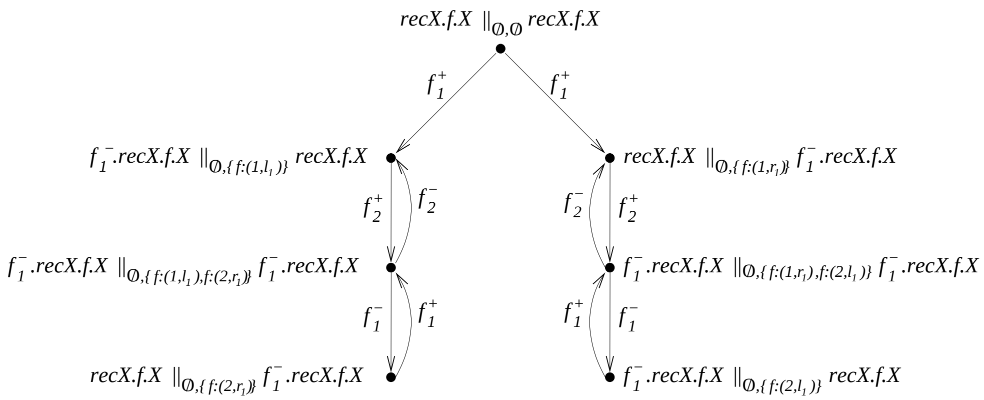


Figure 7.3: Example of recursive system

cause inconsistencies because when applying such rules for deriving the moves of a term P, the negative premises always refer to the moves of a subterm of P (and not of P itself), hence the operational semantics is stratifiable [89].

We are now in a position to define the well-named *IGSMP* (see Chapter 6) obtained as the semantic model of a process of the calculus.

Definition 7.2 *The semantic model $\mathcal{G}[\![P]\!]$ of $P \in IGSMP_g$ is the well-named IGSMP defined by:*

$$\mathcal{G}[\![P]\!] = (S_P, Act, T_{+,P}, T_{-,P}, T_{a,P}, P)$$

where:

- *S_P is the least subset of $IGSMP_{sg}$ such that:*
  - $P \in S_P$
  - *if $P' \in S_P$ and $P' \xrightarrow{\gamma} P''$, then $P'' \in S_P$*
- *$T_{+,P}, T_{-,P}$ and $T_{a,P}$ are the restriction of T_+, T_- and T_a to $S_P \times Act \times S_P$, $S_P \times TAct^+ \times S_P$ and $S_P \times TAct^- \times S_P$.* ■

Example 7.1 In Fig. 7.3 we depict the semantic model of $recX.f.X \parallel_\emptyset recX.f.X$. As expected, we obtain the same well-named IGSMP as that derived in Chapter 6 via parallel composition of two well-named IGSMPs repeatedly executing f delays (see Fig. 6.5). ■

In the following theorem, where we consider "P/L", "$P[\varphi]$", and "$P \parallel_S P$" to be *static* operators [122], we show that finite semantic models are obtained for a wide class of recursive systems.

Theorem 7.1 *Let P be a $IGSMP_g$ process such that for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator. Then P is a finite state process.*

Proof The proof of this theorem derives from the fact that the number of states of the semantics of P which differ only for the contents of mappings M parameterizing parallel operators, are always finite, because the maximum index a delay may assume is bounded by the maximum number of processes that may run in parallel in a state. ■

Note that the class of processes considered in this corollary includes strictly the class of *nets of automata*, i.e. terms where no static operator occurs in the scope of any recursion.

7.3 Observational Congruence for IGSMP

In this section we introduce the notion of observational congruence over $IGSMP_{sg}$ processes, and we will show it to be a congruence with respect to all the operators of our calculus. Such a notion is defined, starting from the notion of weak bisimulation over Well-Named IGSMPs we defined in Chapter 6, according to the classical notion of observational congruence [122] and probabilistic bisimulation [117]. In the definition of observational congruence we make use of function TW defined in Chapter 6.

Definition 7.3 *Two closed terms P, Q of $IGSMP_{sg}$ are observational congruent, written $P \simeq Q$, iff:*

- *for every $\sigma \in NPAct$ and $P' \in IGSMP_{sg}$,*

  $P \xrightarrow{\sigma} P'$ *implies* $Q \stackrel{\sigma}{\Longrightarrow} Q'$ *for some* Q' *with* $P' \approx Q'$,

- *for every $\sigma \in NPAct$ and $Q' \in IGSMP_{sg}$,*

  $Q \xrightarrow{\sigma} Q'$ *implies* $P \stackrel{\sigma}{\Longrightarrow} P'$ *for some* P' *with* $P' \approx Q'$,

- *for every $f \in PDF^+$ and equivalence class I of β,*

  $TW(P, f, I) = TW(Q, f, I)$ ■

We consider $\simeq$ as being defined also on the open terms of $IGSMP_{sg}$ by extending observational congruence with the standard approach of [122].

Theorem 7.2 *$\simeq$ is a congruence w.r.t. all the operators of IGSMP, including recursion.*

Proof The proof of this theorem follows the lines of the similar proof in [122] that is adapted to our setting. The only relevant case is that of parallel composition operator. It suffices to show that $\{(P_1 \,\|_{S,M}\, Q, P_2 \,\|_{S,M}\, Q) \mid P_1 \simeq P_2\}$ is a (weak) bisimulation. ■

7.3.1 Axiomatization

In this section we present an axiom system which is complete for $\simeq$ on finite state $IGSMP_{sg}$ terms.

The axiom system $\mathcal{A}_{IGSMP}$ for $\simeq$ on $IGSMP_{sg}$ terms is formed by the axioms presented in Fig. 7.4. In this figure "$\lfloor\!\lfloor$" and "$|$" denote, respectively, the left merge and synchronization merge operators (see e.g. [1]). Moreover θ ranges over $TAct^+ \cup TAct^-$. We recall from Sect. 7.2.2 that $\gamma, \gamma', \ldots$ range over $Act \cup TAct^+ \cup TAct^-$.

The axioms $(Pri1)$ and $(Pri2)$ express the two kinds of priorities of $IGSMP$, respectively, priority of τ actions over (semi-)delays and priority of delay starts over delay terminations. The axiom (Par) is the standard one except that when the position of processes P and Q is exchanged we must invert left and right inside M. The inverse $\overline{M}$ of a mapping M is defined by $\overline{M} = \{f:(i,r_j) \mid f:(i,l_j) \in M\} \cup \{f:(i,l_j) \mid f:(i,r_j) \in M\}$. Axioms $(LM4)$ and $(LM5)$ just reflect the operational rules of the parallel operator for a delay move of the left-hand process. The axioms $(Rec1-3)$ handle strongly guarded recursion in the standard way [124].

If we consider the obvious operational rules for "$\lfloor\!\lfloor_{S,M}$" and "$|_{S,M}$" that derive from those we presented for the parallel operator [8] then the axioms of $\mathcal{A}_{IGSMP}$ are sound.

A sequential state is defined to be one which includes "$\underline{0}$", "X" and operators ".", "+", "$recX$" only; leading to the following theorem.

Theorem 7.3 *If an $IGSMP_{sg}$ process P is finite state, then $\exists P' : \mathcal{A}_{IGSMP} \vdash P = P'$ with P' sequential state.*

Proof Let $s_1 \ldots s_n$ be the states of the operational semantics of P, $s_n \equiv P$. It can be easily seen that for each $i \in \{1 \ldots n\}$, there exist J_i and $\theta_{i,j}$, $k_{i,j}$ with $j \in J_i$ such that $\mathcal{A}_{IGSMP} \vdash s_i = \sum_{j \in J_i} \theta_{i,j}.s_{k_{i,j}}$ where $\sum_\emptyset \equiv \underline{0}$. Then for each i, from 1 to n, we do the following. If i is such that $\exists j \in J_i : k_{i,j} = i$ we have, by applying $(Rec3)$, that $s_i = recX.(\sum_{j \in J_i : k_{i,j} \neq i} \theta_{i,j}.s_{k_{i,j}} + \sum_{j \in J_i : k_{i,j} = i} \theta_{i,j}.X)$. Then we replace each subterm s_i occurring in the equations for $s_{i+1} \ldots s_n$ with its equivalent term. When, in the equation for $s_n \equiv P$, we have replaced s_{n-1}, we are done. ■

For sequential states the axioms of $\mathcal{A}_{IGSMP}$ involved are just the standard axioms of [124], and the axioms for priority and probabilistic choice. From Theorem 7.3 and by resorting to arguments similar to those presented in [124] and [99] we derive the completeness of $\mathcal{A}_{IGSMP}$.

Theorem 7.4 *$\mathcal{A}_{IGSMP}$ is complete for $\simeq$ over finite state $IGSMP_{sg}$ processes.*

Proof The proof of this theorem follows the lines of the proof of [124]. In particular weights are treated as rates of exponential distributions in the proof of [99]. ■

[8]The definition of the operational rule for "$|_{S,M}$" must allow for actions "τ" to be skipped [1], as reflected by axiom $(SM4)$.

$(A1)$	$P + Q = Q + P$	
$(A2)$	$(P + Q) + R = P + (Q + R)$	
$(A3)$	$\alpha.P + \alpha.P = \alpha.P$	
$(A4)$	$P + \underline{0} = P$	
$(Tau1)$	$\gamma.\tau.P = \gamma.P$	
$(Tau2)$	$P + \tau.P = \tau.P$	
$(Tau3)$	$\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$	
$(TAct)$	$<f, w>.P = <f_1^+, w>.f_1^-.P$	
$(Prob)$	$<f_i^+, w>.P + <f_i^+, w'>.P = <f_i^+, w + w'>.P$	
$(Pri1)$	$\tau.P + \theta.Q = \tau.P$	
$(Pri2)$	$<f_i^+, w>.P + g_j^-.Q = <f_i^+, w>.P$	
$(Hi1)$	$\underline{0}/L = \underline{0}$	
$(Hi2)$	$(\gamma.P)/L = \gamma.(P/L)$	$\gamma \notin L$
$(Hi3)$	$(a.P)/L = \tau.(P/L)$	$a \in L$
$(Hi4)$	$(P + Q)/L = P/L + Q/L$	
$(Rel1)$	$\underline{0}[\varphi] = \underline{0}$	
$(Rel2)$	$(\alpha.P)[\varphi] = \varphi(\alpha).(P[\varphi])$	
$(Rel3)$	$(\theta.P)[\varphi] = \theta.(P[\varphi])$	
$(Rel4)$	$(P + Q)[\varphi] = P[\varphi] + Q[\varphi]$	
(Par)	$P \parallel_{S,M} Q = P \lfloor\!\lfloor_{S,M} Q + Q \lfloor\!\lfloor_{S,\overline{M}} P + P \mid_{S,M} Q$	
$(LM1)$	$\underline{0} \lfloor\!\lfloor_{S,M} P = \underline{0}$	
$(LM2)$	$(a.P) \lfloor\!\lfloor_{S,M} Q = \underline{0}$	$a \in S$
$(LM3)$	$(\alpha.P) \lfloor\!\lfloor_{S,M} Q = \alpha.(P \parallel_{S,M} Q)$	$\alpha \notin S$
$(LM4)$	$(<f_i^+, w>.P) \lfloor\!\lfloor_{S,M} Q = <f^+_{n(M_f)}, w>.(P \parallel_{S,M \cup \{f:(n(M_f),l_i)\}} Q)$	
$(LM5)$	$(f_i^-.P) \lfloor\!\lfloor_{S,M} Q = f_j^-.(P \parallel_{S,M - \{f:(j,l_i)\}} Q)$	$f:(j, l_i) \in M$
$(LM6)$	$(P + Q) \lfloor\!\lfloor_{S,M} R = P \lfloor\!\lfloor_{S,M} R + Q \lfloor\!\lfloor_{S,M} R$	
$(SM1)$	$P \mid_{S,M} Q = Q \mid_{S,M} P$	
$(SM2)$	$\underline{0} \mid_{S,M} P = \underline{0}$	
$(SM3)$	$(\gamma.P) \mid_{S,M} (\gamma'.Q) = \underline{0}$	$(\gamma \notin S \ \vee \ \gamma \neq \gamma') \wedge \tau \notin \{\gamma, \gamma'\}$
$(SM4)$	$(\tau.P) \mid_{S,M} Q = P \mid_{S,M} Q$	
$(SM5)$	$(a.P) \mid_{S,M} (a.Q) = a.(P \parallel_{S,M} Q)$	$a \in S$
$(SM6)$	$(P + Q) \mid_{S,M} R = P \mid_{S,M} R + Q \mid_{S,M} R$	
$(Rec1)$	$recX.P = recY.(P\{Y/X\})$	provided that Y is not free in *rec*X.P
$(Rec2)$	$recX.P = P\{recX.P/X\}$	
$(Rec3)$	$Q = P\{Q/X\} \Rightarrow Q = recX.P$	provided that X is strongly guarded in P

Figure 7.4: Axiomatization for IGSMP

Example 7.2 Let us consider the system $recX.f.X \,\|_{\emptyset}\, recX.f.X$ of the previous example. In the following we show how this process can be turned into a sequential process by applying the procedure presented in the proof of Theorem 7.3. In the following we let f_i^+ stand for $<f_i^+, 1>$ and we abbreviate $\mathcal{A}_{IGSMP} \vdash P = Q$ with $P = Q$. Moreover we let $P \equiv recX.f.X$ and $P' \equiv f_1^-.recX.f.X$. Initially we note that $P = f.recX.f.X = f_1^+.f_1^-.recX.f.X$, by applying $(Rec2)$ and $(TAct)$. We start the procedure of the proof of Theorem 7.3 with the initial state $P \,\|_{\emptyset,\emptyset}\, P$. We have:

$$P \,\|_{\emptyset,\emptyset}\, P = f_1^+.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P) + f_1^+.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P)$$

by applying (Par), $(LM4)$ and $(SM3)$. From this equation we derive:

$$P \,\|_{\emptyset,\emptyset}\, P = <f_1^+, 2>.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P)$$

by applying $(Prob)$. Then, we have:

$$P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P = f_2^+.(P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P')$$

by applying (Par), $(LM4)$, $(LM5)$, $(SM3)$ and $(Pri2)$. Then, we have:

$$P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P' = f_1^-.(P \,\|_{\emptyset,\{f:(2,r_1)\}}\, P') + f_2^-.(P \,\|_{\emptyset,\{f:(1,r_1)\}}\, P')$$

by applying (Par), $(LM5)$ and $(SM3)$. From this equation we derive:

$$P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P' = f_1^-.(P \,\|_{\emptyset,\{f:(2,r_1)\}}\, P') + f_2^-.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P)$$

by applying (Par), $(A1)$ and $(SM1)$ to $P \,\|_{\emptyset,\{f:(1,r_1)\}}\, P'$. Finally we have:

$$P \,\|_{\emptyset,\{f:(2,r_1)\}}\, P' = f_1^+.(P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P')$$

by applying (Par), $(LM4)$, $(LM5)$, $(SM3)$ and $(Pri2)$. Now we perform the second part of the procedure where we generate recursive processes and we substitute states with equivalent terms. We start with $P \,\|_{\emptyset,\{f:(2,r_1)\}}\, P'$. Since the state does not occur in its equivalent term we do not have to generate any recursion. Substituting the state with its equivalent term in the other equations generates the new equation:

$$P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P' = f_1^-.f_1^+.(P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P') + f_2^-.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P).$$

Then we consider the state $P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P'$. We change its equation by generating a recursion as follows:

$$P' \,\|_{\emptyset,\{f:(1,l_1),f:(2,r_1)\}}\, P' = recY.(f_1^-.f_1^+.Y + f_2^-.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P)).$$

Substituting the state with its equivalent term in the remaining equations generates the new equation:

$$P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P = f_2^+.recY.(f_1^-.f_1^+.Y + f_2^-.(P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P)).$$

Now we consider the state $P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P$. We change its equation by generating a recursion as follows:

$$P' \,\|_{\emptyset,\{f:(1,l_1)\}}\, P = recX.(f_2^+.recY(f_1^-.f_1^+.Y + f_2^-.X)).$$

Substituting the state with its equivalent term in the remaining equations generates the new equation:

$$P \,\|_{\emptyset,\emptyset}\, P = <f_1^+, 2>.recX.(f_2^+.recY(f_1^-.f_1^+.Y + f_2^-.X)).$$

Therefore we have turned our initial system $recX.f.X \,\|_\emptyset\, recX.f.X$ into the recursive sequential process $<f_1^+, 2>.recX.(f_2^+.recY(f_1^-.f_1^+.Y + f_2^-.X))$. Note that the operational semantics of this process generates the labeled transition system of Fig. 6.6 derived in Chapter 6 as the minimal version of the IGSMP in Fig. 7.3. ■

7.4 Example: Queuing Systems G/G/1/q

In this section we present an example of specification with *IGSMP*. In particular we concentrate on Queuing Systems (*QSs*) $G/G/1/q$, i.e. *QSs* which have one server and a FIFO queue with q-1 seats and serve a population of unboundedly many customers. In particular the *QS* has an interarrival time which is generally distributed with distribution f and a service time which is generally distributed with distribution g.

Such a system can be modeled with *IGSMP* as follows. Let a be the action representing the fact that a new customer arrives at the queue of the service center, d be the action representing that a customer is delivered by the queue to the server. The process algebra specification is the following one: [9]

- $QS_{G/G/1/q} \triangleq Arrivals \,\|_{\{a\}} (Queue_0 \,\|_{\{d\}}\, Server)$

  - $Arrivals \triangleq f.a.Arrivals$
  - $Queue_0 \triangleq a.Queue_1$

    $Queue_h \triangleq a.Queue_{h+1} + d.Queue_{h-1} \qquad 0 < h < q-1$

    $Queue_{q-1} \triangleq a.Queue_{q-1} + d.Queue_{q-2}$
  - $Server \triangleq d.g.Server$

We have specified the whole system as the composition of the arrival process, the queue and the server which communicate via action types a and d. Then we have separately modeled the arrival process, the queue, and the server. As a consequence if we want to modify the description by changing the interarrival time distribution f or the service time distribution g , only component *Arrivals* or *Server* needs to be modified while component *Queue* is not affected. Note that the role of actions a and d is defining interactions among the different system components. Such actions have zero duration and they are neglected from the performance viewpoint.

In Fig. 7.5 we show $\mathcal{G}[\![QS_{G/G/1/q}]\!]$. In this picture A stands for *Arrivals*, A' stands for $f^-.a.Arrivals$, A'' stands for $a.Arrivals$. Similarly, S stands for *Server*, S' stands for $g.Server$, S'' stands for $g^-.Server$.

[9] In the specification we use process constants (as defined in Chapter 2), instead of the operator "$recX$", to denote recursion. The reason being that the use of constants is suitable for doing specifications, while the use of operator "$recX$" is preferable when dealing with axiomatizations. The two constructs are shown to be completely equivalent in [122].

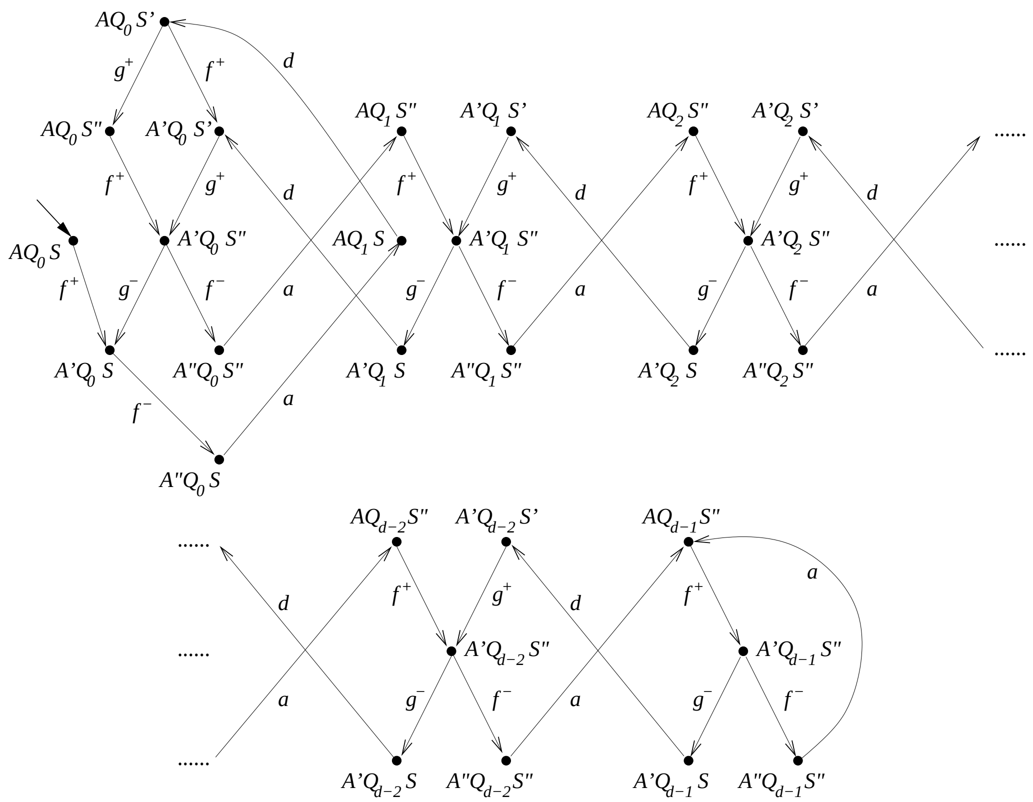


Figure 7.5: Semantic Model

Moreover, Q_h stands for $Queue_h$, for any h. We omit parallel composition operators in terms, so, e.g., AQ_hS stands for $Arrivals \parallel_{\{a\}} (Queue_h \parallel_{\{d\}} Server)$.

In order to derive the performance model of the system $QS_{G/G/1/q}$ we have to make sure that it is complete both from the interaction and the performance viewpoints. In Fig. 7.5 we have visible actions a and d, therefore the behavior of the system can be influenced by interaction with the environment and is not complete. We make it complete by considering $QS_{G/G/1/q}/\{a, d\}$ so that every action in the semantic model of Fig. 7.5 becomes a τ action.

As far as completeness w.r.t. performance is concerned, we present in Fig. 7.6 the minimal version of $\mathcal{G}[\![QS_{G/G/1/q}/\{a, d\}]\!]$, obtained by aggregating weakly bisimilar states (see Sect. 7.3). Since in the minimal model there are no longer internal τ actions, we have that our system is complete also w.r.t. performance.

By applying the formal procedure defined in Sect. 6.3, hence by solving choice trees in the minimal model of Fig. 7.6, we finally obtain the *GSMP* of Fig. 7.7. The probability Pr associated to each transition of the *GSMP* is 1. P_{init} is 1 for the state pointed by the arrow and 0 for all the other states. The elements e_1 and e_2 represent the delays f and g.

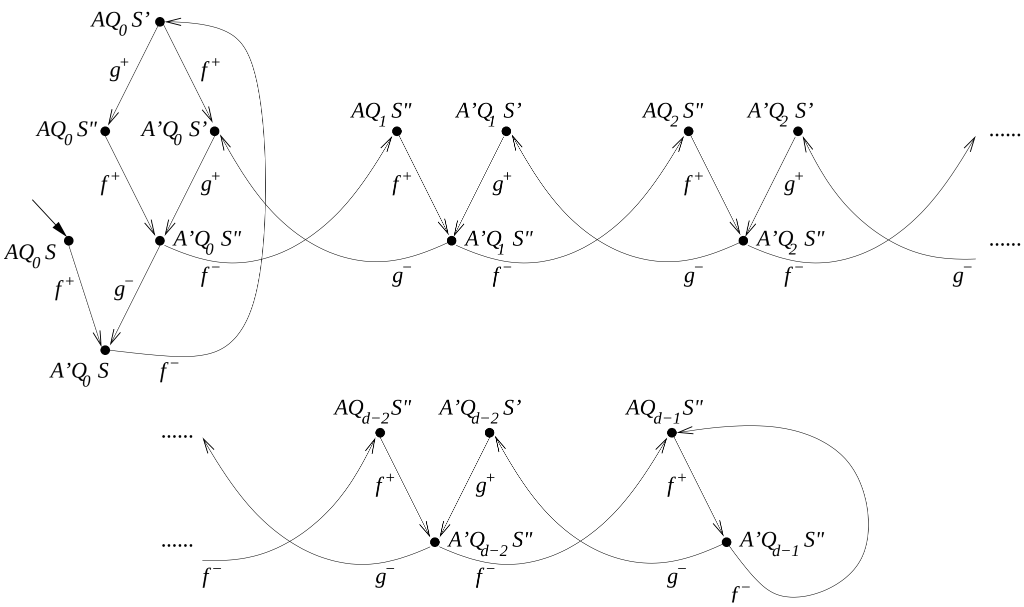
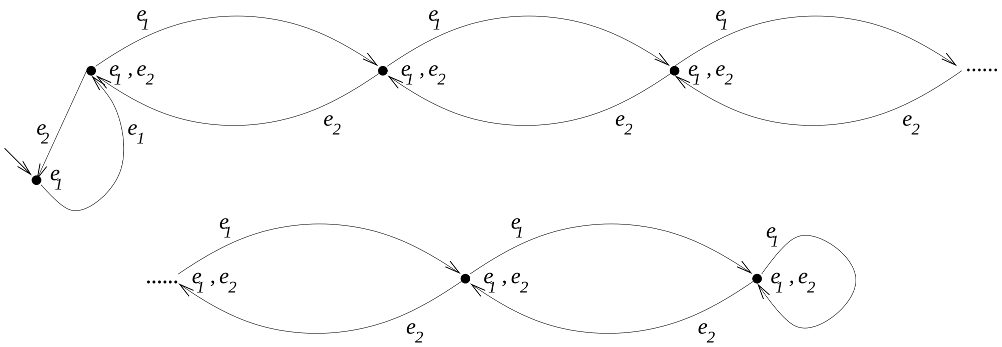


Figure 7.6: Minimal Semantic Model



$$ElPDF(e_1) = f, \qquad ElPDF(e_2) = g$$

Figure 7.7: Derived GSMP

7.5 Conclusion

Several algebraic languages which express generally distributed durations like the calculus of *IGSMPs* have been previously developed. In this section we present related work by starting from languages that follow a completely different approach in representing the behavior of systems.

In [49] a *truly concurrent approach* to modeling systems is proposed which employs general distributions. From a term of the algebra presented in [49] a *truly concurrent* semantic model (a stochastic extension of a bundle event structure) is derived that represents statically the concurrency of the system by expressing the components of the system and the causal relationships among them. Therefore the behavior of the system is not described by representing explicitly all possible global system states as it happens in labeled transition systems. In this way a very concise semantic model is obtained where duration distributions can be statically associated with durational actions. The drawback of this approach is that the semantic models produced must nevertheless be translated to a transition system form before their performance can be evaluated. This because in *GSMPs* the evolution of a stochastic process is represented in such a form. Another algebraic approach to modeling systems with general distributions is the *discrete event simulation approach* [90, 69]. For example in [90] an algebra is developed that extends CCS with temporal and probabilistic operators in order to formally describe discrete event simulations. Such algebra employs actions with null duration (events) and delays similarly as in *IGSMP*. The states produced by the operational semantics include explicitly the residual durations of delays (as real numbers), hence the semantic models of systems are *not* finite. This approach excludes a priori the possibility of making mathematical analysis of such models by means of established theoretical results such as analytical solution methods for (insensitive) *GSMPs* [121].

Other languages have been previously developed, that are more similar to the calculus of *IGSMPs*, in that they represent the behavior of specified systems via labeled transition systems which are (in many cases) finite. According to these approaches [87, 139, 113] the execution of durational actions is represented in an abstract way, as in *IGSMP*, without including explicitly their residual durations (as real numbers) in states. In [87] the technique of "start reference" is employed in order to have a pointer to the system state where an action begins its execution. In [139], instead, information about causality relations among actions is exploited in order to establish the starting point of actions. In a recent work [113] a methodology for obtaining finite semantic models from the algebra of [90] is defined, which is based on symbolic operational semantics. Such semantics generates symbolical transition systems which abstract from time values by representing operations on values as symbolic expressions. The drawback of these approaches is that the structure of the semantic models generated is very different from that of *GSMPs*. It is therefore not always clear how to derive a performance model for a specified system and [113] only provides a (quite involved) procedure for deriving a *GSMP* from systems belonging to a certain class.

The languages that are closest to *IGSMP*, in that they produce semantic models which represent probabilistic durations as in *GSMPs* are those of [7, 69]. In particular such semantic models represent the performance behavior of systems by means of (some kind of) clocks with probabilistic duration which can be easily seen as the elements of a *GSMP*. With the language of [7], performance models are derived from terms specifying systems by applying to them a preliminary procedure that gives a different name to each durational action of the term. In this way, each name represents a different clock in the semantic model of the system. In the approach of [7] the events of action starts are not explicitly expressed in the semantic models and choices are resolved via the race policy (alternative actions are executed in parallel and the first action that terminates wins) instead of the preselection policy as in *IGSMP*. The approach of [69], which has been developed concurrently with our work on general distributions (the first papers [70] and [41] on the two approaches appeared at the same workshop), is the most similar to ours. The language presented in [69] is endowed with an abstract semantics which may generate finite intermediate semantic models (from these models it is then possible to derive the infinite models which are used for discrete event simulation). With this language clock names must be explicitly expressed in the term that specify the system and the fact that a different name is used for each clock is ensured by imposing syntactical restrictions in terms. As in *IGSMP* the execution of a clock is represented by the events of clock start and clock termination, but here these two events must be explicitly expressed in the term specifying a system and they are not automatically generated by the operational semantics. Moreover, the language of [69] can only express choices between events of clock terminations (which are resolved through race policy) and cannot express probabilistic choices which are a basic ingredient of *GSMPs*. A drawback of the approaches of [7, 69] w.r.t. the calculus of *IGSMP* is that there is no easy way to decide equivalence of systems (hence to minimize their state space). This is because in order to establish the equivalence of two systems it is necessary to associate in some way the names of the clocks used by one system with the names of the corresponding clocks used by the other one. As we will see in Chapter 13, trying to extend the notion of bisimulation in this way turns out to be rather complex especially in the presence of probabilistic choices. In *IGSMP*, instead, names of clocks are dynamically generated by the operational semantics with a fixed rule. In this way equivalent systems get the same names for clocks and there is no need to associate names of clocks for establishing equivalence. We can, therefore, rely on standard (probabilistic) bisimulation and we have the opportunity to reuse existing results and tools.

Chapter 8

Mapping IGSMPs onto Pure Markovian and Real-Time Processes

In this chapter we present the two formal mappings from IGSMPs, representing the stochastic and real-time behavior of a concurrent system in an integrated way, into IWMCs, representing the pure stochastic (Markovian) behavior of the system, and into ITA, representing the pure real-time behavior of the system. Both mappings are compositional and preserve equivalence of systems.

8.1 Introduction

In Chapter 1 we showed that formal specification languages expressing generally distributed time can be seen as representing the real-time and stochastic time aspects of a concurrent system in an integrated fashion. In this chapter we show how to derive from such an integrated specification of a concurrent system a pure stochastic (Markovian) specification and a pure real-time specification. The capability of performing such mappings is important because it allows us to use, in the analysis of stochastic real-time systems, besides the "integrated" techniques introduced in Chapters 6 and 7, also the results and software tools already developed for Markovian systems and real-time systems.

In particular we will map IGSMP integrated specifications into IWMC specifications, representing the pure stochastic (Markovian) behavior of the system, and into ITA, representing the pure real-time behavior of the system. The former mapping is obtained by approximating generally distributed durations with phase-type durations. Technically, such mapping is performed compositionally at the algebraic level by replacing each delay prefix $<f, w>$ occurring in an algebraic term of an IGSMP specification with an IWMC term $w.P$, where P is the algebraic representation of a phase-type distribution (see Chapter 2) approximating f. In this way we map a term of the calculus of IGSMPs into a term of IWMC. The latter mapping is obtained by abstracting from probability related information. Such mapping is still performed compositionally, but at the level of models (not at the level of algebraic terms). In particular we define how to derive an ITA from an IGSMP by turning probabilistic choices into non-deterministic choices and by considering the support of the distribution of a clock, i.e. the set of time values that may happen with probability (density) greater than 0, as the set of possible values for its duration. Moreover we show that such mapping is compositional, i.e. is preserved by CSP parallel composition and hiding. If every distribution used in the GSMP has a support which is a finite collection of intervals, then the derived ITA is analyzable with existing techniques and tools (see Chapter 5).

We present in Sect. 8.2 the formal mapping from IGSMP terms to IWMC terms and in Sect. 8.3 the formal mapping from IGSMPs to ITA.

8.2 Deriving the Pure Markovian Process

Given an IGSMP term $P \in \mathit{IGSMP}_g$, as defined in Chapter 7, we derive an IWMC term $Q \in \mathit{IWMCg}$, as defined in Chapter 4, by approximating general distribution with phase-type distributions. Since phase-type distributions are defined as the time to absorption in a CTMC with one absorbing state, any phase-type distribution *pht* can be represented by some term P_{pht} of IWMC, made up of only weighted prefixes "$w.\_$", exponentially timed prefixes "$\lambda.\_$", choice operators "$\_ + \_$", recursion operators "$\mathit{rec}Y.\_$", and occurrences of a free variable X representing the absorbing state. Note that weight prefixes "$w.\_$" are necessary to model probabilistic initial states in CTMCs. We assume that only phase-type distribution are considered such that the probability of the absorbing state to be the initial state is zero. On the other hand this is consistent with the idea of using phase-type distributions as approximating distributions which are continuous and have an infinite support. Given this constraint it is not restrictive to assume that terms P_{pht} are such that each occurrence of X appears in P_{pht} in the form $\lambda.X$ for some λ.

Given a function $\mathit{approx} : \mathit{PDF}^+ \twoheadrightarrow \mathit{PhT}$, which associates with each general distribution f occurring in an IGSMP specification P its approximating phase-type distribution *pht*, term $Q \in \mathit{IWMCg}$ is obtained as follows. Denoted with $R[R'/X]$ the term obtained from a term R by replacing R' for X inside R, we just replace each occurrence of a subterm $<f, w>.P'$ in P with $w.(P_{approx(f)}[P'/X])$.

Definition 8.1 *Given $P \in \mathit{IGSMP}_g$ and a function $\mathit{approx} : \mathit{PDF}^+ \twoheadrightarrow \mathit{PhT}$, which associates with each general distribution occurring in P an approximating phase-type distribution, we define $\mathcal{M}[\![P, \mathit{approx}]\!] \in \mathit{IWMCg}$ to be the term obtained by replacing each occurrence of a subterm $<f, w>.P'$ in P with $w.(P_{approx(f)}[P'/X])$.* ■

The following theorem, where we denote by $\mathit{approx}(P)$ the term of IGSMP_g obtained from $P \in \mathit{IGSMP}_g$ by replacing distributions f in prefixes $<f, w>$ according to *approx*, shows the correctness of the mapping from IGSMP to IWMC terms (performance measures are preserved).

Theorem 8.1 *Given $P \in \mathit{IGSMP}_g$ and $\mathit{approx} : \mathit{PDF}^+ \twoheadrightarrow \mathit{PhT}$, we have that, for every fixed adversary resolving non-deterministic choices, the stochastic process underlying $\mathit{approx}(P)$ is the same as that underlying $\mathcal{M}[\![P, \mathit{approx}]\!]$ (provided that in $\mathcal{M}[\![P, \mathit{approx}]\!]$ we do not consider states that are distinguished just* *because they enable different derivatives of terms $P_{approx(f)}$, for the same distribution f, as being different states in the underlying stochastic process).*

Proof: It is sufficient to show that we can create a mapping from the states of $approx(P)$ onto the states of $\mathcal{M}[\![P, approx]\!]$ (modulo not considering as being different states the states of $\mathcal{M}[\![P, approx]\!]$ that are distinguished just because they enable different derivatives of terms $P_{approx(f)}$, for the same distribution f) such that, if we abstract from renaming of $approx(P)$ states, the stochastic process underlying $approx(P)$ and $\mathcal{M}[\![P, approx]\!]$ (for a given adversary) are exactly the same. In particular such a correspondance is obtained as follows. We put a state P' of $approx(P)$ in correspondance with any state of $\mathcal{M}[\![P, approx]\!]$ obtained from $\mathcal{M}[\![P', approx]\!]$ by replacing occurrences of prefixes $f_i^-.R$ with arbitrary derivatives $P'_{f_i} \neq X$ of $P_{approx(f)}$ (when doing this variables X in P'_{f_i} are replaced by the "continuation" R of the prefix). The fact that we obtain identical stochastic processes derives from the following considerations. From the definition of the semantics of IGSMP and IWMC terms, it is easy to see that, given a state P' of $approx(P)$ we have the following correspondance between single transitions leaving P' and (disjoint) group of transitions leaving a state P'' of $\mathcal{M}[\![P, approx]\!]$ corresponding to P'.

- For every action transition α leaving P' and leading to Q' there is a single corresponding action transition α leaving P'' which leads to a state Q'' that corresponds to Q' and is such that it preserves the derivatives P''_{f_i} used in P'' in correspondance to prefixes f_i^- occurring in P'.
- For every delay start transition $<f_i^+, w>$ leaving P' and leading to Q' there is a single probabilistic transition w leaving P'' that leads to a state Q'' that corresponds to Q' and is such that it preserves the derivatives P''_{f_i} used in P'' in correspondance to prefixes f_i^- occurring in P'.
- For every delay termination transition f_i^- leaving P' and leading to Q' there is:
  - Either a group of probabilistic transitions such that each probabilistic transition w leaving P'' leads to a state Q'' that corresponds to P' and is such that: (i) for each prefix g_j^- occurring in P' with $g \neq f$ and $j \neq i$ it preserves the derivative P''_{g_j} used in P'' in correspondance to prefix g_j^-; (ii) in correspondance to prefix f_i^- it, instead, makes use of a derivative Q''_{f_i} such that, if P''_{f_i} is derivative used in P', P''_{f_i} reaches Q''_{f_i} via a w transition.
  - Or a group of exponentially timed transitions such that each exponentially timed transition λ leaving P'':
    * Either leads to a state Q'' that corresponds to P' and is such that: (i) for each prefix g_j^- occurring in P' with $g \neq f$ and $j \neq i$ it preserves the derivative P''_{g_j} used in P'' in correspondance to prefix g_j^-; (ii) in correspondance to prefix f_i^- it, instead, makes use of a derivative Q''_{f_i} such that, if P''_{f_i} is derivative used in P', P''_{f_i} reaches Q''_{f_i} via a λ transition.
    * Or leads to a state Q'' that corresponds to Q' and is such that for each prefix g_j^- occurring in P' with $g \neq f$ and $j \neq i$ it preserves the derivative P''_{g_j} used in P'' in correspondance to prefix g_j^-.

Moreover, if P' is a probabilistic state, the state P'' of $\mathcal{M}[\![P, approx]\!]$ corresponding to P' also exibits, for each probabilistic derivative P''_{f_i} used in P'' in correspondance to a prefix f_i^- occurring in P', a group of probabilistic transitions such that each probabilistic transition w leaving P'' leads to a state Q'' that corresponds to P' and is such that: (i) for each prefix g_j^- occurring in P' with $g \neq f$ and $j \neq i$ it preserves the derivative P''_{g_j} used in P'' in correspondance to prefix g_j^-; (ii) in correspondance to prefix f_i^- it, instead, makes use of a derivative Q''_{f_i} such that, if P''_{f_i} is derivative used in P', P''_{f_i} reaches Q''_{f_i} via a w transition.

Therefore, since (for a given adversary) the stochastic processes underlying $approx(P)$ and $\mathcal{M}[\![P, approx]\!]$ exibit the same state to state transitions (modulo: not considering states of $\mathcal{M}[\![P, approx]\!]$ corresponding to the same $approx(P)$ state as being different, and renaming states of $approx(P)$ according to the correspondance) and that $\mathcal{M}[\![P, approx]\!]$ correctly represents (due to the memoryless property of exponential distribution and to the fact that weight prefixes have zero duration) via derivatives of P'_{f_i} of $P_{approx(f)}$ the residual distributions of generally distributed delays f_i in states of $approx(P)$, the stochastic process underlying $approx(P)$ is the same as that underlying $\mathcal{M}[\![P, approx]\!]$. □

The following theorem shows that, thanks to the fact that the semantics of IGSMP delays are defined by means of an ST semantics, observational equivalence is preserved when delays are refined by means of phase-type distributions. We denote with $\simeq_{IGSMP}$ observational equivalence over IGSMP terms (defined in Chapter 7), while we denote with $\simeq_{IWMC}$ observational equivalence over IWMC terms (defined in Chapter 4).

Theorem 8.2 *Given $P, Q \in IGSMP_g$ and a function $approx : PDF^+ \twoheadrightarrow PhT$, we have that $P \simeq_{IGSMP} Q$ implies $\mathcal{M}[\![P, approx]\!] \simeq_{IWMC} \mathcal{M}[\![Q, approx]\!]$.*

Proof: It is sufficient just to show that we can build a weak bisimulation β' between states of the semantic models of IWMC terms $\mathcal{M}[\![P, approx]\!]$ and $\mathcal{M}[\![Q, approx]\!]$ (see Chapter 4) from a weak bisimulation β between states of the semantic models of IGSMP terms P and Q (see Chapter 6). In particular β' is obtained as follows. Given P', Q' such that $P'\beta Q'$ and considering, for each delay f_i such that f_i^- occurs in P' (or equivalently, since $P'\beta Q'$, in Q'), an arbitrary derivative P'_{f_i} of $P_{approx(f)}$, we must have that the terms obtained from $\mathcal{M}[\![P', approx]\!]$ and $\mathcal{M}[\![Q', approx]\!]$ by replacing occurrences of prefixes $f_i^-.R$ with terms P'_{f_i} (when doing this variables X in P'_{f_i} are replaced by the "continuation" R of the prefix) are related by β'. The fact that β' is a weak bisimulation derives from the fact that transitions f_i^- of equivalent IGSMP state terms P' and Q' are matched according to bisimulation and occurrences of f_i^- prefixes are replaced with the same derivative of of $P_{approx(f)}$ in both $\mathcal{M}[\![P', approx]\!]$ and $\mathcal{M}[\![Q', approx]\!]$. Moreover, since in the calculus of IGSMPs we use ST semantics, we cannot have two delays in execution with the same name f_i, hence in a state of an IGSMP term we have at most one occurrence of a prefix f_i^-. □

The simple mapping above from IGSMP terms into IWMC terms is significant from a pure performance viewpoint in that it shows process algebra to provide exactly the machinery necessary for approximating GSMPs with CTMCs through phase-type distributions. This because, while directly transforming at the model level a GSMP into a CTMC via phase-type approximation is really cumbersome due to the interleaving of the exponential phases, when using process algebra we just have to approximate general distributions at the term level and then the parallel operator automatically computes the interleaving of exponential phases for us. Finally, such a mapping confirms ST semantics to be the adequate semantics for generally distributed time in that approximation of activity durations with phase-type distributions is a form of action refinement.

8.3 Deriving the Pure Real-Time Process

Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, as defined in Chapter 6, we derive an ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, as defined in Chapter 5, by turning probabilistic choices into non-deterministic choices and by considering the support of the distribution of a clock as the set of possible values for its duration. In particular, clock start transitions C_i^+ are turned into reset transitions C_i, while clock termination transitions C_i^- are turned into clock bound transitions $C_i \in T$, where T is the support of the distribution $D(C_i)$. Note that a technique like this, which is based on the idea that we introduced in [32] of considering support of distributions as constraints over clocks, was also used in [50] for deriving timed automata from the stochastic automata model of [66]. Subsequently, in [67] it was shown that a more complex technique, which generates new states for each interval composing the domain of the support of the probability distribution of clocks, is actually needed for correctly deriving timed automata from the model of [66]. This because it can be seen that in such a model the direct transformation of clock termination transitions into transitions requiring clocks to assume values in the support of their distributions causes timed automata which behave differently from the original system to be derived. This is due to the fact that in the model of [66] it may happen that a clock termination transition is executed some time after the clock the transition refers to actually terminates. Since such a phenomenon cannot happen in IGSMPs, our simple technique which does not increase the system state space, can be correctly applied.

Now we present the precise definition of support of a probability distribution that we need for the translation. We follow the idea of [67] of defining the support (therein called "useful domain") in such a way that, if a time value is in the support set, then either it has non-zero measure, or it is internal, i.e. it belongs to an open interval which is all included in the support set (and which must have non-zero measure). This avoids considering traces containing action orderings which in the original IGSMP occur with zero probability (see [67]).

Definition 8.2 *Given a probability distribution f over $\mathbb{R}$, the support of f, denoted by $supp(f)$, is the set obtained from the least closed subset of $\mathbb{R}$ with measure 1 by eliminating non-internal values with measure 0.* ■

It is trivial to verify that for each probability distribution f, $supp(f)$ has measure 1 (hence that the definition is correct).

Definition 8.3 *Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, we define $\mathcal{T}[\![\mathcal{G}]\!]$ to be the ITA $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$, where T_r and T_b are given by*

- $T_r = \{(s, C_i, s') \mid (s, C_i^+, s') \in T_+\}$
- $T_b = \{(s, C_i \in T, s') \mid (s, C_i^-, s') \in T_- \wedge T = supp(D(C_i))\}$

■

In order to show the correctness of the mapping from IGSMP to ITA, we define the concepts of *supported transitions* of IGSMPs and of *possible transitions* of ITA. Supported transitions of an IGSMP are timed transitions $t \in \mathbb{R}^+ \cup 0$ and action transitions $\alpha \in Act$ executable by the IGSMP according to its semantics (see Chapter 6) when we consider as possible values sampled for a clock with distribution f the time values in $supp(f)$ only. Similarly a possible transition of an ITA starting is a timed transitions $t \in \mathbb{R}^+ \cup 0$ and action transitions $\alpha \in Act$ executable by the ITA according to its semantics (see Chapter 5). Both transition relation are defined over a common set of states representing scenarios in significant (and corresponding) moments of the behavior of the IGSMP and ITA. In particular, states are of the form $[s, v]$, where $s \in \Sigma$ (the set of IGSMP or ITA states) and $v : \mathcal{C} \rightarrow \mathbb{R}^+ \cup \{0\}$ is a valuation function assigning a spent time value to each of the clocks in $\mathcal{C}$ (the set of IGSMP or ITA clocks). $[s, v]$ represents a scenario in which clocks have reached the values in v and clocks with non-zero spent time value have already fired every clock related transition (termination transitions for IGSMPs and clock bound transitions for ITA) to be fired in the instant of time v refers to.

Before defining supported transitions for IGSMPs we introduce some notation. Let $[\![\mathcal{G}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, P, T_t, T'_a, s'_0)$ be the RISTTS obtained from the IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$. We denote by $pr_{\langle s,v \rangle}$ the measure function defined by the probability space $P(s, v, -)$ and by $sampled_{\langle s,v \rangle, \langle s'',v'',C_n \rangle}$ the distribution of the residual duration of clock C_n, given that $\langle s'', v'' \rangle$ is reached from $\langle s, v \rangle$ and that clock C_n samples the least time value. Such a distribution is defined by the probability space $time_{s'',v'',C_n}(P(s, v, -)) \, / \, pr(\{\langle s'', v'', C_n : t'' \rangle \mid t'' \in \mathbb{R}\})$, where $time_{s'',v'',C_n}$ is the partial function from Σ' to $\mathbb{R}$ defined by $time_{s'',v'',C_n}(s'', v'', C_n : t'') = t''$ (we consider a partial function f applied to a measure space P, where the domain D of f is a set which is measurable in P, to yield the space induced by f from the "subspace" of P obtained by restricting its domain to D).

Definition 8.4 *Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$ and denoted the RISTTS $[\![\mathcal{G}]\!]$ (the semantics of the IGSMP $\mathcal{G}$) by $[\![\mathcal{G}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, P, T_t, T'_a, s'_0)$, we have $[s, v] \xrightarrow{t}_{\mathcal{G}} [s', v']$ iff there exist $v_1, v_2, v'_1, v'_2 : \mathcal{C} \twoheadrightarrow \mathbb{R}^+ \cup \{0\}$ and $CS, CS' \subseteq \mathcal{C}$, with $v_1 = v|_{CS}$, $v_2 = v|_{\mathcal{C}-CS}$, $v'_1 = v'|_{CS'}$ and $v'_2 = v'|_{\mathcal{C}-CS'}$, such that:* [1]

- $\langle s, v_1, - \rangle \xrightarrow{t} \langle s', v'_1, - \rangle$ *and $v'_2 = v_2 + t$ (i.e. s is a waiting state and it turns out that $s = s'$ and $v_1 = v'_1 = \emptyset$).*

[1] We denote by $f|_S$ the restriction of function f to the set S and by $\twoheadrightarrow$ a partial function.

- $\langle s, v_1, - \rangle \in dom(P)$ *(i.e. s is a probabilistic or a timed state of the IGSMP) and there exists* $\langle s'', v'', C_n : t' \rangle$, *with* $pr_{\langle s,v_1 \rangle}(\{\langle s'', v'', C_n : t'' \rangle \mid t'' \in \mathbb{R}\}) > 0$ *and* $t' \in supp(sampled_{\langle s,v_1 \rangle, \langle s'', v'', C_n \rangle})$, *such that* $\langle s'', v'', C_n : t' \rangle \xrightarrow{t} \langle s', v'_1, - \rangle$. *Moreover* $v'_2|_{\mathcal{C} - dom(v'')} = v_2|_{\mathcal{C} - dom(v'')} + t$ *and if* $t = t'$ *then* $v'_2|_{dom(v'')} = \{(C_n, v''(C_n) + t)\}$.

Moreover we have $[s, v] \xrightarrow{\alpha}_{\mathcal{G}} [s', v']$ *iff there exist* $v_1, v_2, v'_1, v'_2 : \mathcal{C} \twoheadrightarrow \mathbb{R}^+ \cup \{0\}$ *and* $CS, CS' \subseteq \mathcal{C}$, *with* $v_1 = v|_{CS}$, $v_2 = v|_{\mathcal{C} - CS}$, $v'_1 = v'|_{CS'}$ *and* $v'_2 = v'|_{\mathcal{C} - CS'}$, *such that:*

- $\langle s, v_1, - \rangle \xrightarrow{\alpha} \langle s', v'_1, - \rangle$ *and* $v'_2 = v_2$ *(it turns out* $v_1 = v'_1$*).*

- $\langle s, v_1, - \rangle \in dom(P)$ *and there exists* $\langle s'', v'', C_n : t' \rangle$, *with* $pr_{\langle s,v_1 \rangle}(\{\langle s'', v'', C_n : t'' \rangle \mid t'' \in \mathbb{R}\}) > 0$ *and* $t' \in supp(sampled_{\langle s,v_1 \rangle, \langle s'', v'', C_n \rangle})$, *such that* $\langle s'', v'', C_n : t' \rangle \xrightarrow{\alpha} \langle s', v'_1, - \rangle$. *Moreover* $v'_2 = v_2|_{\mathcal{C} - dom(v'')}$ *(it turns out* $v'_1 = v''$ *and that* α *is a visible action).* ■

Note that, from the semantics of IGSMPs, it can be easily inferred that the condition in the last item is satisfied whenever there exists $\langle s'', v'' \rangle$, with $pr_{\langle s,v \rangle}(\{\langle s'', v'', C_n : t'' \rangle \mid C_n \in \mathcal{C} \wedge t'' \in \mathbb{R}\}) > 0$, such that for any C_n and t'' we have $\langle s'', v'', C_n : t'' \rangle \xrightarrow{\alpha} \langle s', v', - \rangle$.

Definition 8.5 *Given an ITA* $\mathcal{T} = (\Sigma, \mathcal{C}, Act, T_r, T_b, T_a, s_0)$ *and denoted the RIPTTS* $[\![\mathcal{T}]\!]$ *(the semantics of the ITA* $\mathcal{T}$*) by* $[\![\mathcal{T}]\!] = (\Sigma', \mathbb{R}^+ \cup \{0\}, Act, T_p, T_t, T'_a, s'_0)$ *we have* $[s, v] \xrightarrow{t}_{\mathcal{T}} [s', v']$ *iff:*

- $t = 0$ *and* $\langle s, v \rangle \xrightarrow{2}{}^* \langle s', v' \rangle$ *and there exists* $t' \in \mathbb{R}^+$ *such that* $\langle s', v' \rangle \xrightarrow{t'}$. [2]

- $t = 0$ *and there exists* s'' *such that* $\langle s, v \rangle \xrightarrow{2}{}^* \langle s'', v' \rangle$ *and* $\langle s'', v' \rangle \xrightarrow{1} \langle s', v' \rangle$ *such that there exists* ϕ, *expressed in terms of clocks* C_n *with* $v'(C_n) = 0$ *only, such that* $s'' \xrightarrow{\phi} s'$ *in the ITA* $\mathcal{T}$.

- *there exists* $\langle s'', v'' \rangle$ *such that* $\langle s, v \rangle \xrightarrow{2}{}^* \langle s'', v'' \rangle$ *and* $\langle s'', v'' \rangle \xrightarrow{t} \langle s', v' \rangle$ *and there exists* $t' \in \mathbb{R}^+$ *such that* $\langle s', v' \rangle \xrightarrow{t'}$.

- *there exist* $\langle s'', v'' \rangle, \langle s''', v''' \rangle$ *such that* $\langle s, v \rangle \xrightarrow{2}{}^* \langle s'', v'' \rangle$ *and* $\langle s'', v'' \rangle \xrightarrow{t} \langle s''', v''' \rangle$ *and* $\langle s''', v''' \rangle \xrightarrow{1} \langle s', v' \rangle$. ■

Moreover we have $[s, v] \xrightarrow{\alpha}_{\mathcal{T}} [s', v']$ *iff:*

- $\langle s, v \rangle \xrightarrow{\alpha} \langle s', v' \rangle$.

- *there exists* $\langle s'', v'' \rangle$ *such that* $\langle s, v \rangle \xrightarrow{2}{}^* \langle s'', v'' \rangle$ *and* $\langle s'', v'' \rangle \not\xrightarrow{2}$ *and* $\langle s'', v'' \rangle \xrightarrow{\alpha} \langle s', v' \rangle$.

[2]We denote by $\xrightarrow{l}{}^*$ a possibly empty sequence of $\xrightarrow{l}$ transitions.

Theorem 8.3 *Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$ we have that for each state $s \in \Sigma$ and valuation function $v : \mathcal{C} \rightarrow \mathbb{R}^+ \cup \{0\}$:*

$$[s,v] \xrightarrow{\;l\;}_{\mathcal{G}} [s',v'] \Leftrightarrow [s,v] \xrightarrow{\;l\;}_{\mathcal{T}[\![\mathcal{G}]\!]} [s',v']$$

where l ranges over $Act \cup (\mathbb{R}^+ \cup \{0\})$.

Proof: We start by showing that $[s,v] \xrightarrow{\;t\;}_{\mathcal{G}} [s',v'] \Leftrightarrow [s,v] \xrightarrow{\;t\;}_{\mathcal{T}[\![\mathcal{G}]\!]} [s',v']$ for $t \in \mathbb{R}^+ \cup \{0\}$.

- If s is a waiting state or a silent state of the IGSMP $\mathcal{G}$ then the statement trivially holds for every v.
- If s a timed state then, denoted with CS the set of clocks labeling its outgoing termination transitions, we can derive t transitions either from the occurrence of clock related transitions or just from time passage in a state.
  - We start by clock related transitions. In the IGSMP $\mathcal{G}$, according to Definition 8.4, we have a $[s,v] \xrightarrow{\;t\;}_{\mathcal{G}} [s',v+t]$ transition for every t and s' such that there exists C_n, with $s \xrightarrow{\;C_n^-\;} s'$ in $\mathcal{G}$, such that $t \in supp(sampled_{\langle s,v|_{CS}\rangle,\langle s,v|_{CS},C_n\rangle})$. It is easy to see that $supp(sampled_{\langle s,v|_{CS}\rangle,\langle s,v|_{CS},C_n\rangle}) = supp([D(C_n) \mid v(C_n)]) - N$, where N is the set of time values which cannot be sampled by clocks in CS conditioned on valuation v. Assumed that $m = \min_{C_{n'} \in CS} \sup(supp([D(C_{n'}) \mid v(C_{n'})]))$, N is determined as follows. $N =]m,\infty[$ if for every $C_{n'} \in CS$ such that $\sup(supp([D(C_{n'}) \mid v(C_{n'})])) = m$ we have $m \in supp([D(C_{n'} \mid v(C_{n'})])$; $N = [m, \infty[$ otherwise.
    Similarly, in the ITA $\mathcal{T}[\![\mathcal{G}]\!]$, according to Definition 8.5, we have a $[s,v] \xrightarrow{\;t\;}_{\mathcal{T}[\![\mathcal{G}]\!]} [s',v+t]$ transition for every t and s' such that there exists C_n, with $s \xrightarrow{\;C_n^-\;} s'$ in $\mathcal{G}$, such that, $t > 0$ if $v(C_n) > 0$, and it holds $\bigwedge\{\ \exists t' \geq t : v(C_{n'}) + t' \in supp(D(C_{n'})) \mid C_{n'} \in CS\}$ and $v(C_n) + t \in supp(D(C_n))$. Since for any distribution f it holds that $t \in supp([f|t']) \Leftrightarrow t > 0 \wedge t + t' \in supp(f)$, it easily derives that the two sets of time values above coincide.
  - We now consider t transition derived just from time passage in a timed state s. In the IGSMP $\mathcal{G}$, according to Definition 8.4, we have a $[s,v] \xrightarrow{\;t\;}_{\mathcal{G}} [s,v+t]$ transition for every t such that there exists $C_n \in CS$ and $t' > t$ such that $t' \in supp(sampled_{\langle s,v|_{CS}\rangle,\langle s,v|_{CS},C_n\rangle})$. It is easy to see that the set of valus t satisfying the condition above is given by $(\ \bigcup_{C_{n'} \in CS} supp([D(C_{n'}) \mid v(C_{n'})])\ ) - [m, \infty[$, where $m = \min_{C_{n'} \in CS} \sup(supp([D(C_{n'}) \mid v(C_{n'})]))$.
    Similarly, in the ITA $\mathcal{T}[\![\mathcal{G}]\!]$, according to Definition 8.5, we have a $[s,v] \xrightarrow{\;t\;}_{\mathcal{T}[\![\mathcal{G}]\!]} [s,v+t]$ transition for every t such that: (i) either $t = 0$ or it holds $\bigwedge\{\ \exists t' \geq t : v(C_{n'}) + t' \in supp(D(C_{n'})) \mid C_{n'} \in CS\}$; and (ii) there exists $t'' > 0$ such that it holds $\bigwedge\{\ \exists t' \geq t'' : v(C_{n'}) + t + t' \in supp(D(C_{n'})) \mid C_{n'} \in CS\}$.
    Again it is immediate to verify that the two sets of time values above coincide.
- If s a probabilistic state then we prove the statement as follows. In the IGSMP $\mathcal{G}$, according to Definition 8.4, we have a $[s,v] \xrightarrow{\;t\;}_{\mathcal{G}} [s',v']$ transition whenever there exists a timed state s'' which

can be reached from s via a sequence of clock start transitions in the IGSMP $\mathcal{G}$ (we donote by CS the set of clocks started in this sequence) and, considered $v'' = v|_{\mathcal{C}-CS} \cup \{(C_n, 0) \mid C_n \in CS\}$, we have $[s'', v''] \xrightarrow{t}_{\mathcal{G}} [s', v'' + t]$ and $v' = v'' + t$.

It is trivial to verify that for the ITA $\mathcal{T}[\![\mathcal{G}]\!]$, according to Definition 8.5, we have exactly the same scenario for transitions $[s, v] \xrightarrow{t}_{\mathcal{T}[\![\mathcal{G}]\!]} [s', v']$.

We now show that $[s, v] \xrightarrow{\alpha}_{\mathcal{G}} [s', v'] \Leftrightarrow [s, v] \xrightarrow{\alpha}_{\mathcal{T}[\![\mathcal{G}]\!]} [s', v']$ for $\alpha \in Act$.

- If s is a waiting state or a silent state of the IGSMP $\mathcal{G}$ then the statement trivially holds for every v.

- If s a timed state then we prove the statement as follows. In the IGSMP $\mathcal{G}$, according to Definition 8.4, we have a $[s, v] \xrightarrow{\alpha}_{\mathcal{G}} [s', v']$ transition whenever $s \xrightarrow{\alpha} s'$ and $v = v'$. Note that the additional α transitions for $[s, v]$ derived from α transitions executable in the IGSMP $\mathcal{G}$ semantics after sampling clocks in s give rise to the same transition as above. For the ITA $\mathcal{T}[\![\mathcal{G}]\!]$ we trivially have the same scenario.

- If s a probabilistic state then we prove the statement as follows. In the IGSMP $\mathcal{G}$, according to Definition 8.4, we have a $[s, v] \xrightarrow{\alpha}_{\mathcal{G}} [s', v']$ transition whenever: either (i) we have $s \xrightarrow{\alpha} s'$ in the IGSMP $\mathcal{G}$ and $v = v'$; or (ii) there exists a timed state s'' which can be reached from s via a sequence of clock start transitions in the IGSMP $\mathcal{G}$ (we donote by CS the set of clocks started in this sequence) and we have $s'' \xrightarrow{\alpha} s'$ in the IGSMP $\mathcal{G}$ and $v' = v|_{\mathcal{C}-CS} \cup \{(C_n, 0) \mid C_n \in CS\}$.

  It is immediate to verify that, in the ITA $\mathcal{T}[\![\mathcal{G}]\!]$, according to Definition 8.5, we have the same scenario.

□

Theorem 8.4 *Given an IGSMP $\mathcal{G} = (\Sigma, \mathcal{C}, D, Act, T_+, T_-, T_a, s_0)$, we have that for each state s and valuation function v associating a time value to the clocks of $\mathcal{G}$ (belonging to $\mathcal{C}$) the set of all supported executions of $\mathcal{G}$ starting in $[s, v]$ is equal to the set of all possible executions of the $\mathcal{T}[\![\mathcal{G}]\!]$ starting in $[s, v]$.*

Proof: An immediate consequence of Theorem 8.3. □

The following theorem shows that weak bisimulation equivalence is preserved when well-named IGSMPs are mapped into ITA. We denote with $\approx_{IGSMP}$ weak bisimulation over well-named IGSMPs (defined in Chapter 6), while we denote with $\approx_{ITA}$ weak bisimulation over ITA (defined in Chapter 5).

Theorem 8.5 *Given two well-named IGSMPs $\mathcal{G}'$ and $\mathcal{G}''$, we have that $\mathcal{G}' \approx_{IGSMP} \mathcal{G}''$ implies $\mathcal{T}[\![\mathcal{G}']\!] \approx_{ITA} \mathcal{T}[\![\mathcal{G}'']\!]$.*

Proof: It is sufficient just to show that we can build a weak bisimulation family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ between the two ITA $\mathcal{T}[\![\mathcal{G}']\!]$ and $\mathcal{T}[\![\mathcal{G}'']\!]$ (see Chapter 5) from a weak bisimulation β between the two IGSMPs $\mathcal{G}'$ and $\mathcal{G}''$ (see Chapter 6). In particular $\mathbf{B}$ is trivially obtained by letting $s_1 \beta_H s_2$ whenever $s_1 \beta s_2$ and H is the identity relationship over the union of the clocks of $\mathcal{G}'$ and $\mathcal{G}''$. □

In the following we show that the mapping of IGSMPs into ITA is indeed compositional.

Theorem 8.6 *Let $\mathcal{G}'$ and $\mathcal{G}''$ be two well-named IGSMPs. For each $S \subseteq Act - \{\tau\}$ we have $\mathcal{T}[\![\mathcal{G}']\!] \,\|_S\, \mathcal{T}[\![\mathcal{G}'']\!] \approx_{ITA} \mathcal{T}[\![\mathcal{G}' \,\|_S\, \mathcal{G}'']\!]$.*

Proof: It is sufficient just to show that we can build a weak bisimulation family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ between the two ITA $\mathcal{T}[\![\mathcal{G}']\!] \,\|_S\, \mathcal{T}[\![\mathcal{G}'']\!]$ and $\mathcal{T}[\![\mathcal{G}' \,\|_S\, \mathcal{G}'']\!]$. In particular $\mathbf{B}$ is obtained by letting $(s_1, s_2)\beta_H(s_1, s_2, M)$ where H includes $\{(C_{f_i,l}, C_{f_j}) \mid f : (j, l_i) \in M\} \cup \{(C_{f_i,r}, C_{f_j}) \mid f : (j, r_i) \in M\}$. The fact that B is a weak bisimulation family simply derives from the fact that when composing in parallel IGSMPs τ transitions pre-empt timed-related transitions and clock start transitions pre-empt clock termination transitions just as in ITA τ transitions pre-empt timed-related transitions and clock reset transitions pre-empt clock bound transitions. □

Theorem 8.7 *Let $\mathcal{G}$ be a well-named IGSMP. For each $L \subseteq Act - \{\tau\}$ we have $\mathcal{T}[\![\mathcal{G}]\!]/L \approx_{ITA} \mathcal{T}[\![\mathcal{G}/L]\!]$.*

Proof: It is sufficient just to show that we can build a weak bisimulation family $\mathbf{B} = \{\beta_H \subseteq \Sigma \times \Sigma \mid H \in \mathcal{H}\}$ between the two ITA $\mathcal{T}[\![\mathcal{G}]\!]/L$ and $\mathcal{T}[\![\mathcal{G}/L]\!]$. In particular $\mathbf{B}$ is trivially obtained by letting $(s_1, s_2)\beta_H(s_1, s_2)$ where H is the identity relationship over the clocks of $\mathcal{G}$. The fact that B is a weak bisimulation family simply derives from the fact that the only effect of hiding standard actions is to pre-empt the execution of time-related transitions both in IGSMPs and in ITA. □

Chapter 9

Dealing with Timed Weakly Guarded Processes

In this chapter we face the problem of dealing with "Zeno" processes (technically, weakly guarded processes) in interactive timed process algebras, so that the integrated approach presented in this part of the thesis (Chapters 4-8) can be extended also to these kind of processes. In particular we consider a generic interactive timed process algebra and we show how to develop a complete axiomatization for observational congruence [122] over finite-state weakly guarded (or fully unguarded) processes. As a consequence of the new insight given by considering weakly guarded processes, we will also show that a definition of parallel operator similar to that of [98], different with respect to that used in the calculi of IWMCs and IGSMPs, is preferable. In particular it is needed in order to obtain the congruence property for weakly guarded processes. The introduction of such an operator makes interactive timed calculi somehow more complex, but more well-founded.

9.1 Introduction

In the last years the expressiveness of classical process algebras have been extended in several ways, e.g. by introducing a notion of pre-emption, real time or stochastic time. As in the case of the "interactive" calculi presented in Chapters 4 and 7 where we assume maximal progress [133], often such extensions have led to the necessity of introducing, explicitly or implicitly, a notion of priority among actions. The problems arising from expressing priority have been previously studied in the context of prioritized process algebras, see e.g. [60, 128, 61] and [62] for a survey. One of the open questions in this context is finding a complete axiomatization for observational congruence. In the literature complete axiomatizations of Milner's standard notion of observational congruence for algebras with priority have been presented only for recursion free processes [128]. For this class of processes a notion of priority of silent τ actions over actions of a type δ (representing, e.g., a time delay) is simply axiomatized by adding the axiom $\tau.E + \delta.F = \tau.E$ to the standard Milner's complete proof system for observational congruence [122, 124].

When we consider also recursive processes (which are essential, e.g., if we want to do performance analysis), if we simply try to extend the Milner's proof system for observational congruence [122] with the axiom above, we do not obtain a correct axiomatization. This because the axiom:

$$recX.(\tau.X + E) = recX.\tau.E \qquad (*)$$

which is part of the proof system of [124], is no longer sound. For example if $E \equiv \delta.E'$, the right-hand term can (weakly) perform δ, whilst the left-hand one cannot because of the priority of τ actions over δ actions. The axiom above is crucial for the completeness of the axiomatization in that it allows to "escape" τ divergence in weakly guarded recursive terms, during the process of turning them into strongly guarded ones. The axiom reflects the possibility of observational congruence to equate τ divergent expressions to non-divergent ones. Due to the fact that observational congruence does not distinguish τ loops from τ transitions, this equivalence is sometimes said to be *fair* with respect to divergence, in the sense that it implicitly assumes that a τ loop will be eventually escaped from a process.

The effects of introducing such a notion of priority on the possibility of escaping divergence have been previously studied, but the problem was not really solved. For example it has previously been claimed [102] that introducing a notion of priority causes observational congruence to become unfair, i.e. it is not always possible to escape τ divergence. In [102] the problem of finding a complete axiomatization for recursive processes is tackled by modifying Milner's standard notion of observational congruence with the additional requirement that two bisimilar terms must have the same opportunity to silently become stable terms, i.e. terms that cannot perform τ actions. Adopting this additional constraint, which now makes observational congruence certainly sensitive to τ divergence, provides us with a simple necessary and sufficient condition for the possibility of escaping divergence. We can escape divergence in $recX.(\tau.X + E)$ if and only if E may perform a silent move.

In this chapter we show that with Milner's standard notion of observational congruence it is *always* possible to escape τ divergence even in the presence of priority. This is done by simply replacing E in the right-hand term of axiom $(*)$ with $pri(E)$, whose behavior is obtained from that of E by removing all transitions δ (and subsequent behaviors) performable in state E. Therefore the new axiom is:

$$recX.(\tau.X + E) = recX.\tau.pri(E)$$

The process algebra we consider is just the algebra of finite-state agents used by Milner in [124] for axiomatizing observational congruence in the presence of recursion, extended with δ prefixing, where δ actions have lower priority than τ actions, i.e. a generic version of the "interactive" calculi considered in Chapters 4 and 7 without static operators. As in [128] and in Chapters 4 and 7 we capture the priority of τ actions over δ actions by cutting transitions which cannot be performed directly in semantic models (and not by discarding them at the level of bisimulation definition as done in [102]) so that we can just apply the ordinary notion of observational congruence [122]. By employing our technique we provide a complete axiomatization for observational congruence over processes of the prioritized process algebra.

The chapter terminates with a discussion about the extension of our basic prioritized process algebra with static operators. The problem here is that, while the extension with the static operators considered in the timed process algebra of [98] can be harmlessly done, if we consider the static operators of the calculi of Chapters 4 and 6, it turns out that observational congruence is no longer a congruence when we consider weakly guarded terms (in particular it is not a congruence for the parallel operator). The approach that we take here is, instead of considering the coarsest congruence which would give rise to an equivalence sensible to τ divergence as that of [102], to consider new static operators similar to those of [98]. The introduction of such operators makes interactive timed calculi somehow more complex, but, we believe, more well-founded.

The chapter is structured as follows. In Sect. 9.2 we present the prioritized algebra and its operational semantics. In Sect. 9.3 we present a complete axiomatization for observational congruence over processes of the algebra. In Sect. 9.4 we report some concluding remarks about the extension of the prioritized algebra with static operators.

9.2 Prioritized Algebra

The prioritized algebra that we consider is just CCS without static operators [124] extended with δ prefixing, where δ actions have lower priority than silent τ actions. The set of names is $\mathcal{A}$. The set of prioritized actions is defined by $PAct = \mathcal{A} \cup \{\tau\}$, ranged over by $a, b, c, \ldots$. The set of all actions is defined by $Act = PAct \cup \{\delta\}$, ranged over by $\alpha, \beta, \ldots$. The set of term variables is Var, ranged over by $X, Y, \ldots$. The set $\mathcal{E}$ of behavior expressions, ranged over by $E, F, G, \ldots$ is defined by the following syntax.

$$E ::= \underline{0} \mid X \mid \alpha.E \mid E + E \mid recX.E$$

The meaning of the operators is the standard one of [122, 124], where $recX$ denotes recursion in the usual way.

As in [124] we take the liberty of identifying expressions which differ only by a change of bound variables. We will write $E\{F/X\}$ for the result of substituting F for each occurrence of X in E, renaming bound variables as necessary.

We adopt the standard definitions of [122, 124] for *free* variable, and *open* and *closed* term. The set of processes, i.e. closed terms, is denoted by $\mathcal{P}$, ranged over by $P, Q, R, \ldots$.

We adopt the following standard definitions concerning guardedness of variables.

Definition 9.1 *X is weakly guarded in E if each occurrence of X is within some subexpression of E of the form $\alpha.F$.*

X is (strongly) guarded in E if we additionally require that $\alpha \neq \tau$.

X is unguarded in E if X is not (strongly) guarded in E.

X is fully unguarded in E if X is neither strongly nor weakly guarded in E. ■

A recursion $recX.E$ is weakly guarded, (strongly) guarded, unguarded, fully unguarded, if X is weakly guarded, (strongly) guarded, unguarded, fully unguarded in E, respectively. As in [124] we say that an expression E is guarded (unguarded) if every (some) subexpression of E which is a recursion is guarded (unguarded).

The operational semantics of the algebra terms is given as a relation $\longrightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$. We write $E \xrightarrow{\alpha} F$ for $(E, \alpha, F) \in \longrightarrow$, $E \xrightarrow{\alpha}$ for $\exists F : (E, \alpha, F) \in \longrightarrow$ and $E \stackrel{\alpha}{\not\longrightarrow}$ for $\not\exists F : (E, \alpha, F) \in \longrightarrow$. $\longrightarrow$ is defined as the least relation satisfying the operational rules in Table 9.1.

$$\alpha.E \xrightarrow{\alpha} E$$

$$\frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} E'}$$

$$\frac{E \xrightarrow{\delta} E' \quad F \stackrel{\tau}{\not\longrightarrow}}{E + F \xrightarrow{\delta} E'} \qquad \frac{F \xrightarrow{\delta} F' \quad E \stackrel{\tau}{\not\longrightarrow}}{E + F \xrightarrow{\delta} F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{recX.E \xrightarrow{\alpha} E'\{recX.E/X\}}$$

Table 9.1: Operational Semantics

Note that the rule for recursion is not the standard one used by Milner in [122, 124], in that it defines the moves of $recX.E$ in a pure structural way, starting from those of E (where X occurs free), instead of $E\{recX.E/X\}$. This rule, which was proven to be equivalent to that of Milner in [12], gives the possibility of simplifying the proofs of some results. In particular such proofs can be made by simply inducing on the structure of terms instead of inducing on the depth of the inference of term transitions.

The equivalence notion we consider over the terms of our prioritized process algebra is the standard notion of observational congruence extended to open terms [122, 124]. In particular we consider here the following characterization of observational congruence given in [124].

Let $\stackrel{\alpha}{\Longrightarrow}$ denote $\xrightarrow{\tau}{}^* \xrightarrow{\alpha} \xrightarrow{\tau}{}^*$. Moreover we define $\stackrel{\hat{\alpha}}{\Longrightarrow} \; = \; \stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\stackrel{\hat{\tau}}{\Longrightarrow} \; = \; \xrightarrow{\tau}{}^*$. Let $E \rhd X$ denote that E contains a free unguarded occurrence of X.

Definition 9.2 *A relation $\beta \subseteq \mathcal{E} \times \mathcal{E}$ is a weak bisimulation if, whenever $(E, F) \in \beta$:*

- *If $E \xrightarrow{\alpha} E'$ then, for some F', $F \stackrel{\hat{\alpha}}{\Longrightarrow} F'$ and $(E', F') \in \beta$.*

- *If* $F \xrightarrow{\alpha} F'$ *then, for some* E', $E \stackrel{\hat{\alpha}}{\Longrightarrow} E'$ *and* $(E', F') \in \beta$.
- $E \rhd X$ *iff* $F \rhd X$.

Two expressions E, F *are weakly bisimilar, written* $E \approx F$, *iff* (E, F) *is included in some weak bisimulation.*

■

Definition 9.3 *Two expressions* E, F *are observational congruent, written* $E \simeq F$, *iff:*

- *If* $E \xrightarrow{\alpha} E'$ *then, for some* F', $F \stackrel{\alpha}{\Longrightarrow} F'$ *and* $E' \approx F'$.
- *If* $F \xrightarrow{\alpha} F'$ *then, for some* E', $E \stackrel{\alpha}{\Longrightarrow} E'$ *and* $E' \approx F'$.
- $E \rhd X$ *iff* $F \rhd X$. ■

Corollary 9.1 *If* $E \simeq F$ *then:*

$$E \xrightarrow{\tau} \quad \Leftrightarrow \quad F \xrightarrow{\tau}$$

■

The following theorem shows that the presence of priority preserves the congruence property of observational congruence w.r.t. the operators of the algebra.

Theorem 9.1 $\simeq$ *is a congruence w.r.t. prefix, choice and recursion operators.*

Proof As far as the prefix operator is concerned, from $E \simeq F$ we trivially derive $\alpha.E \simeq \alpha.F$

Concerning the choice operator, from $E \simeq F$ we derive $E+G \simeq F+G$ as follows. Supposed $E+G \xrightarrow{\alpha} H$ we have two cases.

- If $E \xrightarrow{\alpha} H$ then (from $E \simeq F$) $F \stackrel{\alpha}{\Longrightarrow} H'$ for some $H \approx H'$. Hence we have $F + G \stackrel{\alpha}{\Longrightarrow} H'$. In particular if $\alpha = \delta$ this derives from the fact that $E \xrightarrow{\alpha} H \;\Rightarrow\; G \stackrel{\tau}{\not\longrightarrow}$.
- If $G \xrightarrow{\alpha} H$ then $F + G \xrightarrow{\alpha} H$. In particular if $\alpha = \delta$ this derives from the following consideration. It must be that $E \stackrel{\tau}{\not\longrightarrow}$, hence (from Corollary 9.1) we have also that $F \stackrel{\tau}{\not\longrightarrow}$.

Concerning the recursion operator, from $E \simeq F$ we derive $recX.E \simeq recX.F$ as follows. We show that

$$\beta = \{(G\{recX.E/X\}, G\{recX.F/X\}) \mid G \text{ contains at most X free}\}$$

satisfies the condition:

> if $G\{recX.E/X\} \xrightarrow{\alpha} H$ then, for some H', H'',
>
> $$G\{recX.F/X\} \stackrel{\alpha}{\Longrightarrow} H'' \text{ with } H'' \approx H' \text{ such that } (H, H') \in \beta,$$
>
> and symmetrically for a move of $G\{recX.F/X\}$.

This implies that β is a weak bisimulation up to $\approx$. Moreover by taking $G \equiv X$ we may conclude that $recX.E \simeq recX.F$. In the following we prove that $(G\{recX.E/X\}, G\{recX.F/X\}) \in \beta$ satisfies the property above by structural induction on the syntax of G.

- If $G \equiv \underline{0}$ then the property above trivially holds.

- If $G \equiv X$ then $G\{recX.E/X\} \equiv recX.E$ and $G\{recX.F/X\} \equiv recX.F$.

  $recX.E \xrightarrow{\alpha} H$ implies $E \xrightarrow{\alpha} G'$ with $H \equiv G'\{recX.E/X\}$. Since $E \simeq F$ we have $F \stackrel{\alpha}{\Longrightarrow} G''$ with $G'' \approx G'$. Therefore $recX.F \stackrel{\alpha}{\Longrightarrow} G''\{recX.F/X\}$ with $G''\{recX.F/X\} \approx G'\{recX.F/X\}$.

  Symmetrically for a move of $recX.F$

- If $G \equiv \alpha.G'$ then $G\{recX.E/X\} \equiv \alpha.(G'\{recX.E/X\})$ and $G\{recX.F/X\} \equiv \alpha.(G'\{recX.F/X\})$. The result trivially follows.

- If $G \equiv G' + G''$ then $G\{recX.E/X\} \equiv G'\{recX.E/X\} + G''\{recX.E/X\}$ and $G\{recX.F/X\} \equiv G'\{recX.F/X\} + G''\{recX.F/X\}$.

  Supposed $G'\{recX.E/X\} + G''\{recX.E/X\} \xrightarrow{\alpha} H$, we have two cases.

  - If $G'\{recX.E/X\} \xrightarrow{\alpha} H$ then (by induction) $G'\{recX.F/X\} \stackrel{\alpha}{\Longrightarrow} H''$ with $H'' \approx H'$ and $(H, H') \in \beta$. Therefore $G'\{recX.F/X\}+G''\{recX.F/X\} \stackrel{\alpha}{\Longrightarrow} H''$ with $H'' \approx H'$ and $(H, H') \in \beta$. In particular if $\alpha = \delta$ this derives from the following consideration. It must be that $G''\{recX.E/X\} \stackrel{\tau}{\not\longrightarrow}$, hence (by induction) we have also that $G''\{recX.F/X\} \stackrel{\tau}{\not\longrightarrow}$.
  - If $G''\{recX.E/X\} \xrightarrow{\alpha} H$ then the result is derived in a similar way.

  Symmetrically for a move of $G'\{recX.F/X\} + G''\{recX.F/X\}$.

- If $G \equiv recY.G'$ then $G\{recX.E/X\} \equiv recY.G'\{recX.E/X\}$ and $G\{recX.F/X\} \equiv recY.G'\{recX.F/X\}$.

  $recY.G'\{recX.E/X\} \xrightarrow{\alpha} G''$ implies $G'\{recX.E/X\} \xrightarrow{\alpha} H$ with $G'' \equiv H\{recY.G'\{recX.E/X\}/Y\}$. By induction we have $G'\{recX.F/X\} \stackrel{\alpha}{\Longrightarrow} H''$ with $H'' \approx H'$ and $(H, H') \in \beta$. Therefore $recY.G'\{recX.F/X\} \stackrel{\alpha}{\Longrightarrow} H''\{recY.G'\{recX.F/X\}/Y\}$ with $H''\{recY.G'\{recX.F/X\}/Y\} \approx H'\{recY.G'\{recX.F/X\}/Y\}$ and $(H\{recY.G'\{recX.E/X\}/Y\}, H'\{recY.G'\{recX.F/X\}/Y\}) \in \beta$.

  Symmetrically for a move of $recY.G'\{recX.F/X\}$. ■

9.3 Axiomatization

We now present an axiomatization of $\simeq$ which is complete over processes $P \in \mathcal{P}$ of our algebra.

Because of priority of "τ" actions over "δ" actions, achieving completeness is not straightforward and we must resort to a technique similar to that presented in [102]. As we already explained in the introduction, the law of ordinary CCS which allows to escape τ divergence ($recX.(\tau.X + E) = recX.\tau.E$) is not sound in a calculus with this kind of priority. Consider for instance the divergent term $F \equiv recX.(\tau.X + \delta.\underline{0})$. Because of priority of "τ" actions over "δ" actions the operational semantics of F is isomorphic to that of $recX.\tau.X$. Hence F is an infinitely looping term which can never escape from recursion by executing the action "δ". If the cited law were sound, we would obtain $F = recX.\tau.\delta.\underline{0}$ and this is certainly not the case.

$(A1)$	$E + F = F + E$	$(A2)$	$(E + F) + G = E + (F + G)$
$(A3)$	$E + E = E$	$(A4)$	$E + \underline{0} = E$
$(Tau1)$	$\alpha.\tau.E = \alpha.E$	$(Tau2)$	$E + \tau.E = \tau.E$
$(Tau3)$	$\alpha.(E + \tau.F) + \alpha.F = \alpha.(E + \tau.F)$		
$(Pri1)$	$pri(\underline{0}) = \underline{0}$	$(Pri2)$	$pri(a.E) = a.E$
$(Pri3)$	$pri(\delta.E) = \underline{0}$	$(Pri4)$	$pri(E + F) = pri(E) + pri(F)$
$(Pri5)$	$pri(pri(E)) = pri(E)$	$(Pri6)$	$\tau.E + F = \tau.E + pri(F)$
$(Rec1)$	$recX.E = E\{recX.E/X\}$		
$(Rec2)$	$F = E\{F/X\} \Rightarrow F = recX.E$		provided that X is strongly guarded and serial in E
$(Ung1)$	$recX.(X + E) = recX.E$		
$(Ung2)$	$recX.(\tau.X + E) = recX.(\tau.pri(E))$		
$(Ung3)$	$recX.(\tau.(X + E) + F) = recX.(\tau.X + E + F)$		
$(Ung4)$	$recX.(\tau.(pri(X) + E) + F) = recX.(\tau.X + E + F)$		

Table 9.2: Axiom system $\mathcal{A}$

In order to overcome this problem in [102] the distinguished symbol "$\perp$" is introduced, which represents an ill-defined term that can be eliminated only if a silent computation is possible. In this way by considering the rule $recX.(\tau.X + E) = recX.\tau.(E + \perp)$ the resulting term which escapes divergence can be turned into a "normal" term only if E may execute a silent move.

This law is surely sound also in our language, but is not sufficient to achieve completeness. Since, differently from [102], we do not impose conditions about stability in our definition of observational congruence, we can escape divergence not only when E includes a silent move but *for all possible* terms E. For example in our calculus (but not in [102]) the term $recX.\tau.X$ is equivalent to $\tau.\underline{0}$ (as in standard CCS), so we can escape divergence in F even if $\tau.X$ has not a silent alternative inside $recX$. In general the behavior of E' such that $recX.\tau.E' = recX.(\tau.X + E)$ is obtained from that of E by removing all "δ" actions (and subsequent behaviors) performable in state E. We denote such E', representing the "prioritized" behavior of state E, with $pri(E)$. The operational semantics of operator pri is just:

$$\frac{E \xrightarrow{\alpha} E'}{pri(E) \xrightarrow{\alpha} E'} \qquad \alpha \neq \delta$$

Therefore in our case τ divergence can always be escaped by turning E into $pri(E)$ and the strongly guarded terms we obtain are always "well-defined" terms.

Note that the introduction of this operator is crucial for being able to axiomatize the priority of τ actions over δ actions in our case. Since we have to remove δ actions performable by a term E even if E does not include a silent move, we cannot do this by employing a special symbol like "$\perp$" instead of using an operator. This because $\perp$ must somehow be removed at the end of the deletion process (in [102] $\perp$ is eliminated by silent alternatives) in order to obtain a "normal" term.

The axiomatization of "$\simeq$" we propose is made over the set of terms $\mathcal{E}_{pri}$, generated by extending the syntax to include the new operator $pri(E)$.

We adopt the following notion of *serial* variable, which is used in the definition of the axiomatization.

Definition 9.4 *X is serial in $E \in \mathcal{E}_{pri}$ if every subexpression of E which contains X, apart from X itself, is of the form $\alpha.F$, $F' + F''$ or $recY.F$ for some variable Y.* ∎

The axiom system $\mathcal{A}$ is formed by the axioms presented in Table 9.2.

The axiom $(Pri6)$ expresses the priority of τ actions over δ actions. Note that from $(Pri6)$ we can derive $\tau.E + \delta.E = \tau.E$ by applying $(Pri3)$. The axioms $(Rec1)$, $(Rec2)$ handle strongly guarded recursion in the standard way [124]. The axioms $(Ung1)$ and $(Ung2)$ allow to turn unguarded terms into strongly guarded ones similarly as in [124]. The axiom $(Ung3)$ and the new axiom $(Ung4)$ are used to transform weakly guarded recursions into the form required by the axiom $(Ung2)$, so that they can be turned into strongly guarded ones. In particular the role of axiom $(Ung4)$ is to remove unnecessary occurrences of operator pri in weakly guarded recursions.

9.3.1 Soundness

Theorem 9.2 *If $\mathcal{A} \vdash E = F$ then $E \simeq F$.*

Proof We prove the soundness of the new laws $(Ung2)$ and $(Ung4)$. For the other laws $(Ung1)$ and $(Ung3)$ the proof is a similar adaptation of the standard one. The soundness of the laws $(Rec1)$ and $(Rec2)$ derives from the fact that systems of equations of the form $X_i = E_i$, where the variables X_i are guarded and serial in terms E_i and are not in the scope of any recursion, have a unique solution up to provable equality. The proof of this fact is a straightforward adaptation of the proof given in [122]. The soundness of the new equations $(Pri1) - (Pri6)$ trivially derives from the fact that the semantic models of left-hand and right-hand terms are isomorphic.

Soundness of $(Ung2)$ In order to prove the soundness of $(Ung2)$ we show that:

$$\beta = \{(G\{recX.(\tau.X + E)/X\}, G\{recX.(\tau.pri(E))/X\}) \mid G \text{ contains at most X free}\} \cup$$
$$\{(recX.(\tau.X + E), pri(E)\{recX.(\tau.pri(E))/X\})\}$$

is a weak bisimulation. From this result it straightforwardly follows that $recX.(\tau.X + E) \simeq recX.(\tau.pri(E))$.

We start the proof that β is a weak bisimulation by considering all the pairs $(G\{recX.(\tau.X + E)/X\}$, $G\{recX.(\tau.pri(E))/X\})$ such that G contains at most X free. In particular we prove by structural induction on the syntax of G the following stronger property:

if $G\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} H$ then, for some H',

$$G\{recX.(\tau.pri(E))/X\} \stackrel{\alpha}{\Longrightarrow} H' \text{ with } (H, H') \in \beta,$$

and symmetrically for a move of $G\{recX.(\tau.pri(E))/X\}$.

- If $G \equiv \underline{0}$, then the property above trivially holds.
- If $G \equiv X$, then $G\{recX.(\tau.X + E)/X\} \equiv recX.(\tau.X + E)$ and $G\{recX.(\tau.pri(E))/X\} \equiv recX.(\tau.pri(E))$.

  Supposed $recX.(\tau.X + E) \xrightarrow{\alpha} H$, first of all we note that $\alpha \neq \delta$ for the priority constraints. We have two cases:

  - If $recX.(\tau.X + E) \xrightarrow{\tau} recX.(\tau.X + E)$, then $recX.(\tau.pri(E)) \xrightarrow{\tau} pri(E)\{recX.(\tau.pri(E))/X)\}$.
  - If $recX.(\tau.X + E) \xrightarrow{a} E'\{recX.(\tau.X + E)/X\}$ with $E \xrightarrow{a} E'$, then $recX.(\tau.pri(E)) \xrightarrow{\tau} \xrightarrow{a} E'\{recX.(\tau.pri(E))/X\}$

  Vice-versa if $recX.(\tau.pri(E)) \xrightarrow{\alpha} H$, we have $\alpha = \tau$ and $H \equiv pri(E)\{recX.(\tau.pri(E))/X\}$. In this situation $recX.(\tau.X + E) \xrightarrow{\tau} recX.(\tau.X + E)$.
- If $G \equiv \alpha.G'$, then $G\{recX.(\tau.X + E)/X\} \equiv \alpha.(G'\{recX.(\tau.X + E)/X\})$ and $G\{recX.(\tau.pri(E))/X\} \equiv \alpha.(G'\{recX.(\tau.pri(E))/X\})$. The result trivially follows.
- If $G \equiv G' + G''$, then $G\{recX.(\tau.X + E)/X\} \equiv G'\{recX.(\tau.X + E)/X\} + G''\{recX.(\tau.X + E)/X\}$ and $G\{recX.(\tau.pri(E))/X\} \equiv G'\{recX.(\tau.pri(E))/X\} + G''\{recX.(\tau.pri(E))/X\}$.

  Supposed $G'\{recX.(\tau.X + E)/X\} + G''\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} H$, we have two cases.

  - If $G'\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} H$, then (by induction) for some H', $G'\{recX.(\tau.pri(E))/X\} \stackrel{\alpha}{\Longrightarrow} H'$ and $(H, H') \in \beta$. Therefore $G'\{recX.(\tau.pri(E))/X\} + G''\{recX.(\tau.pri(E))/X\} \stackrel{\alpha}{\Longrightarrow} H'$. In particular if $\alpha = \delta$ this derives from the following consideration. It must be that $G''\{recX.(\tau.X + E)/X\} \stackrel{\tau}{\not\longrightarrow}$, hence (by induction) we have also that $G''\{recX.(\tau.pri(E))/X\} \stackrel{\tau}{\not\longrightarrow}$.
  - If $G''\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} H$ then the result is derived in a symmetric way.

Similarly for a move of $G'\{recX.(\tau.pri(E))/X\} + G''\{recX.(\tau.pri(E))/X\}$.

- If $G \equiv recY.G'$ then $G\{recX.(\tau.X+E)/X\} \equiv recY.G'\{recX.(\tau.X+E)/X\}$ and $G\{recX.(\tau.pri(E))/X\} \equiv recY.G'\{recX.(\tau.pri(E))/X\}$.

  $recY.G'\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} G''$ implies $G'\{recX.(\tau.X + E)/X\} \xrightarrow{\alpha} H$ with $G'' \equiv H\{recY.G'\{recX.(\tau.X + E)/X\}/Y\}$. By induction we have $G'\{recX.(\tau.pri(E))/X\} \stackrel{\alpha}{\Longrightarrow} H'$ with $(H, H') \in \beta$. Therefore $recY.G'\{recX.(\tau.pri(E))/X\} \stackrel{\alpha}{\Longrightarrow} H'\{recY.G'\{recX.(\tau.pri(E))/X\}/Y\}$ with $(H\{recY.G'\{recX.(\tau.X + E)/X\}/Y\}, H'\{recY.G'\{recX.(\tau.pri(E))/X\}/Y\}) \in \beta$.

  Symmetrically for a move of $recY.G'\{recX.(\tau.pri(E))/X\}$.

We conclude the proof that β is a weak bisimulation by considering the pair: $(recX.(\tau.X + E), pri(E)\{recX.(\tau.pri(E))/X\})$.

Supposed $recX.(\tau.X + E) \xrightarrow{\alpha} H$, first of all we note that $\alpha \neq \delta$ for the priority constraints. We have the following two cases.

- If $recX.(\tau.X + E) \xrightarrow{\tau} recX.(\tau.X + E)$, then $pri(E)\{recX.(\tau.pri(E))/X\}$ just makes no move.
- If $recX.(\tau.X + E) \xrightarrow{a} E'\{recX.(\tau.X + E)/X\}$ with $E \xrightarrow{a} E'$, then $pri(E)\{recX.(\tau.pri(E))/X\} \xrightarrow{a} E'\{recX.(\tau.pri(E))/X\}$.

Vice-versa if $pri(E)\{recX.(\tau.pri(E))/X\} \xrightarrow{\alpha} H$ (it must be $\alpha \neq \delta$ by the definition of operator pri) we have the following two cases.

- If $pri(E)\{recX.(\tau.pri(E))/X\} \xrightarrow{a} E'\{recX.(\tau.pri(E))/X\}$ with $E \xrightarrow{a} E'$, then $recX.(\tau.X + E) \xrightarrow{a} E'\{recX.(\tau.X + E)/X\}$.
- If $pri(E)\{recX.(\tau.pri(E))/X\} \xrightarrow{\tau} pri(E)\{recX.(\tau.pri(E))/X\}$ (X occurs fully unguarded in E), then $recX.(\tau.X + E)$ just makes no move.

Soundness of $(Ung4)$ The proof that $(Ung4)$ is sound is similar. We show that:

$$\beta = \{(G\{recX.(\tau.(pri(X) + E) + F)/X\}, G\{recX.(\tau.X + E + F)/X\}) \mid G \text{ contains at most X free}\}$$
$$\cup \{(pri(recX.(\tau.(pri(X) + E) + F)) + E\{recX.(\tau.(pri(X) + E) + F)/X\}, recX.(\tau.X + E + F))\}$$

is a weak bisimulation. From this result it straightforwardly follows that $recX.(\tau.(pri(X) + E) + F) \simeq recX.(\tau.X + E + F)$.

Again we start the proof that β is a weak bisimulation by considering all the pairs $(G\{recX.(\tau.(pri(X)+E) + F)/X\}, G\{recX.(\tau.X + E + F)/X\})$ such that G contains at most X free. As we did for $(Ung2)$, we prove by structural induction on the syntax of G the following stronger property:

if $G\{recX.(\tau.(pri(X) + E) + F)/X\} \xrightarrow{\alpha} H$ then, for some H',

$$G\{recX.(\tau.X + E + F)/X\} \overset{\alpha}{\Longrightarrow} H' \text{ with } (H, H') \in \beta,$$

and symmetrically for a move of $G\{recX.(\tau.X + E + F)/X\}$.

In the cases $G \equiv \underline{0}$, $G \equiv \alpha.G'$, $G \equiv G' + G''$ and $G \equiv recY.G'$ the proof is the same as for $(Ung2)$.

In the case $G \equiv X$ we have that $G\{recX.(\tau.(pri(X) + E) + F)/X\} \equiv recX.(\tau.(pri(X) + E) + F)$ and $G\{recX.(\tau.X + E + F)/X\} \equiv recX.(\tau.X + E + F)$.

Supposed $recX.(\tau.(pri(X)+E)+F) \xrightarrow{\alpha} H$, first of all we note that $\alpha \neq \delta$ from the priority constraints. We have two cases:

- if $recX.(\tau.(pri(X)+E)+F) \xrightarrow{\tau} \{(pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\}$, then $recX.(\tau.X + E + F) \xrightarrow{\tau} recX.(\tau.X + E + F)$;
- if $recX.(\tau.(pri(X)+E)+F) \xrightarrow{a} F'\{recX.(\tau.(pri(X)+E)+F)/X\}$ with $F \xrightarrow{a} F'$, then $recX.(\tau.X + E + F) \xrightarrow{a} F'\{recX.(\tau.X + E + F)/X\}$.

Vice-versa if $recX.(\tau.X + E + F) \xrightarrow{\alpha} H$, we note that $\alpha \neq \delta$ from the priority constraints. We have three cases:

- if $recX.(\tau.X + E + F) \xrightarrow{\tau} recX.(\tau.X + E + F)$, then $recX.(\tau.(pri(X) + E) + F) \xrightarrow{\tau} \{(pri(recX.(\tau.(pri(X) + E) + F)) + E\{recX.(\tau.(pri(X) + E) + F)/X\}$;
- if $recX.(\tau.X + E + F) \xrightarrow{a} E'\{recX.(\tau.X + E + F)/X\}$, with $E \xrightarrow{a} E'$, then $recX.(\tau.(pri(X) + E) + F) \xrightarrow{\tau} \xrightarrow{a} E'\{recX.(\tau.(pri(X) + E) + F)/X\}$;
- if $recX.(\tau.X + E + F) \xrightarrow{a} F'\{recX.(\tau.X + E + F)/X\}$, with $F \xrightarrow{a} F'$, then $recX.(\tau.(pri(X) + E) + F) \xrightarrow{a} F'\{recX.(\tau.(pri(X) + E) + F)/X\}$.

We conclude the proof that β is a weak bisimulation by considering the pair: $(pri(recX.(\tau.(pri(X) + E) + F)) + E\{recX.(\tau.(pri(X) + E) + F)/X\}, recX.(\tau.X + E + F))$.

Supposed $pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\} \xrightarrow{\alpha} H$, first of all we note that $\alpha \neq \delta$ from the priority constraints. We have the following three cases.

- If $pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\} \xrightarrow{\tau} pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\}$, then $recX.(\tau.X+E+F)$ just makes no move. Note that this case may arise in two different situations if X occurs fully unguarded in E.
- If $pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\} \xrightarrow{a} F'\{recX.(\tau.(pri(X)+E)+F)/X\}$ with $F \xrightarrow{a} F'$, then $recX.(\tau.X+E+F) \xrightarrow{a} F'\{recX.(\tau.X+E+F)/X\}$. Note that this case may arise in two different situations if X occurs fully unguarded in E.
- If $pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X)+E)+F)/X\} \xrightarrow{a} E'\{recX.(\tau.(pri(X)+E)+F)/X\}$ with $E \xrightarrow{a} E'$, then $recX.(\tau.X + E + F) \xrightarrow{a} E'\{recX.(\tau.X + E + F)/X\}$.

Vice-versa if $recX.(\tau.X + E + F) \xrightarrow{\alpha} H$ (it must be $\alpha \neq \delta$ from the priority constraints) we have the following three cases.

- If $recX.(\tau.X+E+F) \xrightarrow{\tau} recX.(\tau.X+E+F)$, then $pri(recX.(\tau.(pri(X)+E)+F))+E\{recX.(\tau.(pri(X) + E) + F)/X\}$ just makes no move.
- If $recX.(\tau.X + E + F) \xrightarrow{a} E'\{recX.(\tau.X + E + F)/X\}$ with $E \xrightarrow{a} E'$, then $pri(recX.(\tau.(pri(X) + E) + F)) + E\{recX.(\tau.(pri(X) + E) + F)/X\} \xrightarrow{a} E'\{recX.(\tau.(pri(X) + E) + F)/X\}$.
- If $recX.(\tau.X + E + F) \xrightarrow{a} F'\{recX.(\tau.X + E + F)/X\}$ with $F \xrightarrow{a} F'$, then $pri(recX.(\tau.(pri(X) + E) + F)) + E\{recX.(\tau.(pri(X) + E) + F)/X\} \xrightarrow{a} F'\{recX.(\tau.(pri(X) + E) + F)/X\}$. ■

9.3.2 Completeness

Now we will show that the axiom system $\mathcal{A}$ is complete over processes of $\mathcal{P}$. In order to do this we smoothly follow the lines of Milner [124], so we deal with systems of recursive equations.

We start by introducing the machinery necessary for proving completeness over guarded expressions of $E \in \mathcal{E}$. Afterwards we will show how the axioms (Ung) can be used to turn an unguarded processes of $\mathcal{P}$ into a guarded process of $\mathcal{P}$.

Definition 9.5 *An equation set with formal variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ and free variables $\mathcal{W} = \{W_1, \ldots, W_m\}$, where $\mathcal{X}$ and $\mathcal{W}$ are disjoint, is a set $S = \{X_i = H_i \mid 1 \leq i \leq n\}$ of equations such that the expressions H_i ($1 \leq i \leq n$) have free variables in $\mathcal{X} \cup \mathcal{W}$.* ■

The equation sets that are actually dealt with in the proof of completeness of [124] belong to the subclass of *standard equation sets*. Here we have to slightly change the characterization of this subclass because of the presence of priority.

Definition 9.6 *An equation set $S = \{X_i = H_i \mid 1 \leq i \leq n\}$, with formal variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ and free variables $\mathcal{W} = \{W_1, \ldots, W_m\}$, is standard if each expression H_i ($1 \leq i \leq n$) is of the form:*

$$H_i \equiv \sum_j \alpha_{i,j}.X_{f(i,j)} + \sum_k X_{g(i,k)}$$

where:

$$\exists j : \alpha_{i,j} = \tau \ \Rightarrow \not\exists j : \alpha_{i,j} = \delta \ .$$

■

As in [124] we define for a standard equation set S the relations $\longrightarrow_S \subseteq \mathcal{X} \times Act \times \mathcal{X}$ and $\rhd_S \subseteq \mathcal{X} \times \mathcal{W}$ as follows:

$X_i \xrightarrow{\alpha}_S X$	iff	$\alpha.X$ occurs in H_i
$X_i \rhd_S W$	iff	W occurs in H_i

Definition 9.7 *A standard equation set S with formal variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ is guarded if there is no cycle $X_i \xrightarrow{\tau}_S {}^+ X_i$.* ■

The following theorem guarantees that from a guarded expression $E \in \mathcal{E}$ we can derive a standard guarded equation set which is provably satisfied by E.

Theorem 9.3 *Every guarded expression $E \in \mathcal{E}$ provably satisfies a standard guarded equation set S.*

Proof The proof is the same of [124], where, in addition, we modify the resulting equation set by eliminating the occurrences of $\delta.X$ (for some X) in the equations which include terms $\tau.Y$ (for some Y). This is done by using the laws $(Pri6)$ and $(Pri3)$. ■

Moreover we have the following two theorems, whose proof is the same as in [124].

Theorem 9.4 *Let $E \in \mathcal{E}$ provably satisfy S, and $F \in \mathcal{E}$ provably satisfy T, where both S and T are standard, guarded sets of equations, and let $E \simeq F$. Then there is a standard, guarded equation set U provably satisfied by both S and T.* ■

Theorem 9.5 *Let $E, F \in \mathcal{E}$ provably satisfy the same standard guarded equation set S, then $\mathcal{A} \vdash E = F$.* ■

Hence we have proved completeness over guarded expressions.

Theorem 9.6 *If E and F are guarded expressions of $\mathcal{E}$ and $E \simeq F$ then $\mathcal{A} \vdash E = F$.* ■

Now we show that each unguarded process can be turned into a guarded process of $\mathcal{P}$, so that we obtain completeness over unguarded processes. We start with a technical lemma which, in analogy to [124], will be used in the proof of this result. Its proof is the same as [124].

Lemma 9.1 *If X occurs free and unguarded in $E \in \mathcal{E}$, then $\mathcal{A} \vdash E = X + E$.* ■

Theorem 9.7 *For each process $P \in \mathcal{P}$ there exists a guarded $P' \in \mathcal{P}$ such that $\mathcal{A} \vdash P = P'$.*

Proof We show, by structural induction, that given an expression $E \in \mathcal{E}$, it is possible to find an expression $F \in \mathcal{E}_{pri}$ such that:

1. if $pri(G)$ is a subexpression of F then $G \equiv X$ for some free variable X;
2. for any variable X, $pri(X)$ is weakly guarded in F, i.e. each occurrence of $pri(X)$ is within some subexpression of F of the form $\alpha.G$;
3. a summation cannot have both $pri(X)$ and Y as arguments, for any (possibly coincident) variables X and Y;

4. for any variable X, each subterm $recX.G$ of F is (strongly) guarded in F, i.e. each occurrence of $recX.G$ is within some subexpression of F of the form $\alpha.H$, with $\alpha \neq \tau$;

5. F is guarded;

6. $\mathcal{A} \vdash E = F$.

Note that a consequence of property 4 is that each unguarded occurrence of any free variable X of F does not lie within the scope of a subexpression $recY.G$ of F;

Showing this result proves the theorem, in that if $E \in \mathcal{E}$ is a process of $\mathcal{P}$, i.e. a closed term, we have (by the properties of F above) that F is also a process of $\mathcal{P}$, it is guarded, and $\mathcal{A} \vdash E = F$.

The result above is derived by structural induction on the syntax of an expression $E \in \mathcal{E}$ as follows.

- If $E \equiv \underline{0}$, then $F \equiv \underline{0}$.

- If $E \equiv X$, for some variable X, then $F \equiv X$.

- If $E \equiv \alpha.E'$ then $F \equiv \alpha.F'$, where F' is the term obtained from E' via the induction hypothesis.

- If $E \equiv E' + E''$, then $F \equiv F' + F''$, where F' and F'' are the terms obtained from E' and E'', respectively, via the induction hypothesis.

- If $E \equiv recX.E'$, then F is evaluated as follows.

  First of all we derive from the term F', obtained from E' via the induction hypothesis, a term F'' such that:

  - F'' satisfies the properties $1-5$ above;
  - X is guarded in F'';
  - $\mathcal{A} \vdash recX.E' = recX.F''$;

  We start by eliminating all fully unguarded occurrences of X in F', via the axiom $(Ung1)$, so that we obtain a term G such that X is weakly guarded in G and $recX.E' = recX.G$. The axiom $(Ung1)$ is sufficient because F' satisfies the property 2 above, so we do not need to deal with priority.

  Then we need to remove weakly guarded occurrences of X in G. In order to do this we begin by evaluating a term G' such that:

  $$G' \equiv \tau.X + G''$$

  where X is guarded in G'' and $recX.G = recX.G'$.

  We do this by employing the following iterative procedure, where initially we let $H \equiv G$.

- We start each iteration with H such that X is weakly guarded in H and H satisfies the properties 1, 3, 4 and 5 above. If there still exists $H' \not\equiv X$ with X occurring unguarded in H' such that H has the form $\tau.H' + H''$ we continue with the procedure. Otherwise H can be turned into a term $G' \equiv \tau.X + G''$ with the property above, by simply using $\tau.X + \tau.X = \tau.X$ and we have finished.
- Since H satisfies property 3, we have the following cases for the structure of H' (after applying the idempotency law $(A3)$ to X or $pri(X)$ if needed).
  1. H' has the form $X + H'''$, where X is weakly guarded in H'''. In this case we consider the term $H'''' \equiv \tau.X + H''' + H''$ and we have $recX.H = recX.H''''$ by applying the axiom $(Ung3)$.
  2. H' has the form $pri(X) + H'''$, where X is weakly guarded in H'''. In this case we consider again the term $H'''' \equiv \tau.X + H''' + H''$ and we have $recX.H = recX.H''''$ by applying the axiom $(Ung4)$.
  3. X is weakly guarded in H' . We have two sub-cases.
     * If $pri(X)$ occurs in H', then we do the following.
       We consider a new variable Y' for each variable Y such that $pri(Y)$ appears in H'. Let H''' be the unique expression of $\mathcal{E}$ such that if we replace $pri(Y)$ for each new variable Y' inside H''', we obtain the term H'.
       Since X occurs unguarded in H', then X' occurs unguarded in H''' and from Lemma 9.1 we have that $H''' = X' + H'''$. Hence, since substitution preserves equality, we have $H' = pri(X) + H'$ and we continue as in the case 2.
     * Otherwise we do the following.
       We consider a new variable Y' for each variable Y such that $pri(Y)$ appears in H'. Let H''' be the unique expression of $\mathcal{E}$ such that if we replace $pri(Y)$ for each new variable Y' inside H''', we obtain the term H'.
       Since X occurs unguarded in H', then X occurs unguarded also in H''' and from Lemma 9.1 we have that $H''' = X + H'''$. Hence, since substitution preserves equality, we have $H' = X + H'$ and we continue as in the case 1.
- Now a new iteration is performed starting from the term H'''' that we have obtained. Since there is at least one unguarded occurrence of X such that the length of the τ path that leads to that occurrence is shorter in H'''' than in H, we are guaranteed that the iterative procedure will eventually terminate.

Now we are in a position to derive the term F'' with the properties we described above. From $G' \equiv \tau.X + G''$ we have $recX.G' = recX.\tau.pri(G'')$ by applying the axiom $(Ung2)$. X is guarded in the term $\tau.pri(G'')$ and such term satisfies properties $2 - 5$. Due to the property 4, the term G'' has the following structure:

$$G'' \equiv \sum_i a_i.H_i + \sum_j \delta.H'_j + \sum_k Y_k + \sum_h pri(Y'_h).$$

where the variables Y_k and Y'_h are free and do not coincide with X because X is guarded in G''. By applying the axioms $(Pri1)$, $(Pri2)$, $(Pri3)$, $(Pri4)$ and $(Pri5)$, we obtain a term $F'' = \tau.pri(G'')$, where:

$$F'' \equiv \tau.(\sum_i a_i.H_i + \sum_k pri(Y_k) + \sum_h pri(Y'_h)\,).$$

Now F'' satisfies also property 1.

Finally we have to transform the term $recX.F''$ so to obtain a term F that satisfies the six properties above.

$recX.F''$ already satisfies the properties 2 and 3 because F'' satisfies them. Moreover $recX.F''$ already satisfies property 5, i.e. it is guarded, because X is guarded in F'' and F'' already satisfy that property.

In order to satisfy property 1 we have to remove the occurrences of $pri(X)$ because now X is no longer a free variable.

We unfold $recX.F''$ via the axiom $(Rec1)$ and we obtain $F''' \equiv F''\{recX.F''/X\} = recX.F''$. Note that F''' may include subterms of the form $pri(recX.F'')$.

We deal with such subterms in the following way. We have $pri(recX.F'') = pri(F''\{recX.F''/X\})$, by applying law $(Rec1)$.

Since X is guarded in F'' and F'' satisfies the property 4, the term $F''\{recX.F''/X\}$ has the following structure:

$$F''\{recX.F''/X\} \equiv \sum_i a_i.H_i + \sum_j \delta.H'_j + \sum_k Y_k + \sum_h pri(Y'_h).$$

where the variables Y_k and Y'_h are free and obviously do not coincide with X. By applying the axioms $(Pri1)$, $(Pri2)$, $(Pri3)$, $(Pri4)$ and $(Pri5)$, we obtain a term $T = pri(F''\{recX.F''/X\})$, where:

$$T \equiv \sum_i a_i.H_i + \sum_k pri(Y_k) + \sum_h pri(Y'_h).$$

Now let us consider a new variable X'. Let T' be the unique term, not having $pri(recX.F'')$ as a subterm, such that $T'\{pri(recX.F'')/X'\} \equiv T$. Since X' is guarded in T' (because X is guarded in F'') and serial in T' (thanks to the transformation of $pri(F''\{recX.F''/X\})$ into T), from $pri(recX.F'') = T'\{pri(recX.F'')/X'\}$ we derive, by applying the law $(Rec2)$, $pri(recX.F'') = recX'.T'$.

Therefore now we can replace all occurrences of $pri(recX.F'')$ in F''' with $recX'.T'$, and we obtain a term $F'''' = F'''$ not having $pri(recX.F'')$ as a subterm. In order to satisfy the property 1 we still have to remove the occurrences of $recX.F''$ in F'''', so that we get rid of the occurrences of $pri(X)$ appearing inside F''.

In order to do this, we consider another new variable X''. Let T'' be the unique term, not having $recX.F''$ as a subterm, such that $T''\{recX.F''/X''\} \equiv F''''$. Since X'' is guarded in T'' (because each occurrence of the expression $recX.F''$ is guarded both in T' and in F''', since X is guarded in F'') and serial in T'' (thanks to the substitution of $pri(recX.F'')$ with $recX'.T'$), from $recX.F'' = F'''' \equiv T''\{recX.F''/X''\}$ we derive, by applying the law $(Rec2)$, $recX.F'' = recX''.T''$.

Now we have that $recX''.T''$ satisfies the property 1, because such term no longer uses the variable X, and does not include occurrences of $pri(X')$ or $pri(X'')$.

Moreover, since X'' is guarded in T'' and each occurrence of $recX'.T'''$, for any T''', is guarded in T'' (because each occurrence of $pri(X)$ is guarded in F''), in order to obtain property 4 it is sufficient to unfold $recX''.T''$, by applying the law $(Rec1)$. In this way we finally obtain $F \equiv T''\{recX''.T''/X''\} = recX''.T''$ that satisfies the six properties above. ■

From Theorem 9.6 and Theorem 9.7 we derive the completeness of $\mathcal{A}$ over processes of $\mathcal{P}$.

Theorem 9.8 *If P and Q are processes of $\mathcal{P}$ and $P \simeq Q$ then $\mathcal{A} \vdash P = Q$.* ■

Note that all the axioms of $\mathcal{A}$ are actually used in the proof of completeness. In particular in the proof of completeness over guarded expressions (Theorem 9.6) we employ the standard axioms $(A1)-(A4)$, $(Tau1)-(Tau3)$ and $(Rec1),(Rec2)$ as in [124], plus the new axioms $(Pri3)$ and $(Pri6)$. All these axioms are necessary even if we restrict to consider completeness over guarded processes only. Moreover, proving that a process of $\mathcal{P}$ can always be turned into a guarded process (Theorem 9.7) requires the use of the remaining axioms $(Pri1),(Pri2),(Pri4),(Pri5)$ and $(Ung1)-(Ung4)$. This supports the claim that our axiomatization is irredundant.

9.4 Conclusion

The basic prioritized algebra presented in this chapter can be easily extended to include static operators, such as a parallel composition operator, by following the approach of [98]. Unfortunately such extension cannot be done by directly considering static operators like those of the calculi of Chapters 4 and 6 because observational congruence would be no longer a congruence for the parallel operator. The problem with congruence is that, e.g., while $\tau.\underline{0} \simeq recX.\tau.X$, since the parallel operator considered in Chapters 4 and 6 is defined in such a way that the *presence* of a τ action within the actions executable by a process pre-empts the other process from executing a timed action δ (global pre-emption [62]), we have that $\tau.\underline{0} \,\|_\emptyset\, \delta.\underline{0} \not\simeq recX.\tau.X \,\|_\emptyset\, \delta.\underline{0}$. This because the semantics of $\tau.\underline{0} \,\|_\emptyset\, \delta.\underline{0}$ is that of $\tau.\delta.\underline{0}$, while the semantics of $recX.\tau.X \,\|_\emptyset\, \delta.\underline{0}$ is that of $recX.\tau.X$ (where no δ action can be executed). Conceptually, such a problem derives from the fact that the parallel operator considered in Chapters 4 and 6 deals with the terminated process $\underline{0}$ (and in general with processes which cannot execute neither τ actions nor δ actions) as if it let time pass. For example $\underline{0} \,\|_\emptyset\, \delta$ may execute δ

and become $\underline{0} \,\|_{\emptyset}\, \underline{0}$. This is obviously in contrast with the fact that $\underline{0}$ is weakly bisimilar to $recX.\tau.X$, which is clearly a process that does not let time pass (it represents a so-called time deadlock).

A very clean solution is, therefore, to consider as processes which can let time pass only processes which can actually execute δ actions. In this way $\underline{0}$ is interpreted not as a terminated process which may let time pass, but as a time deadlock and the definition of the parallel operator changes. In particular the parallel operator must be defined, similarly as in [98], in such a way that the *absence* of δ actions within the actions executable by a process (which means that the process cannot let time pass) pre-empts the other process from executing a timed action δ. Requiring the *absence* of δ actions differs from requiring the *presence* of τ actions exactly for the class of processes that were misinterpreted, i.e. processes which cannot execute neither τ actions nor δ actions. In particular for the calculus of IWMCs the rules for the parallel operator in Chapter 4 must be changed in such a way that, when we derive an exponentially timed move of $P \,\|_S\, Q$ from a corresponding move of P we require that also Q may perform an exponentially timed move, instead of requiring that Q must not perform a τ move (or a weighted move). [1] Note that, differently from [98], even if we require that Q may perform an exponentially timed move, we do not actually perform it because of the memoryless property of exponential delays. Similarly, for the calculus of IGSMPs the rule for the parallel operator in Chapter 6 must be changed in such a way that, when we derive a clock termination move of $P \,\|_S\, Q$ from a corresponding move of P we require that also Q may perform a clock termination move, instead of requiring that Q must not perform a τ move (or a clock start move). [2] Note that, differently from [98], even if we require that Q may perform a clock termination move, we do not actually perform it because of the assumption that every active clock in a state advances as time passes.

Although not strictly necessary, we believe that the modified versions of the calculi of IWMC and IGSMP (with the new parallel operator) should come along with other two new operators similar to those used in [98]:

- A new prefix operator $\tilde{\alpha}.P$ which is defined: as $\alpha.P$ if α is τ or δ, as $recX.(\alpha.P + \delta.X)$ otherwise. Such a prefix, which allow visible actions to be delayed as in the original definition of IWMCs and IGSMPs, should be used for specifying systems, while the "basic" prefix $\alpha.P$ constitutes the basis for building an axiomatization.

- A new choice operator $P \tilde{+} Q$ which, similarly as for the new parallel operator, allows one of P and Q to let time pass only if the other one may let time pass and is defined in such a way that time passage does not resolve the choice. Such a choice operator, which allow new prefixes $\tilde{a}$ (where a is a visible action) to be used without causing the delays δ preceding the execution of the a to solve the choice, should be used for specifying systems, while the "basic" choice $P + Q$ constitutes the basis for building an axiomatization.

[1] A similar change must be made when deriving weighted moves of $P \,\|_S\, Q$ from corresponding moves of P (we must require that also Q may perform a weighted move).

[2] A similar change must be made when deriving clock start moves of $P \,\|_S\, Q$ from corresponding moves of P (we must require that also Q may perform a clock start move).

Finally, we could also include a new symbol representing a terminated process defined as $recX.\delta.X$, which represents a terminated process which may let time pass as in the original definition of IWMCs and IGSMPs.

Even if we think that such "improved" versions of the calculi of IWMC and IGSMP, although more complex, are more "well-founded", we decided not to present them in Chapters 4 and 6 so as to keep presentation as simple as possible.

Part III

Generative-Reactive Processes

Chapter 10

Generative-Reactive Probabilistic Processes with Advancing Speeds

In this chapter we present an alternative approach for integrating fixed and probabilistic durations in a single modeling formalism which is not based on considering generally distributed time distributions, but simply relies on (finite) probabilistic choices. In this way we can avoid the intractability of mathematical performance analysis which is inherent of generalized semi-Markov processes.

Differently from the approach considered in the second part of the thesis, the technique we present here is based on discrete time, i.e. time is not considered to be continuous (as in continuous time Markov chains or generalized semi-Markov processes), but is represented by a sequence of discrete steps as in discrete time Markov chains.

In particular, we consider a discrete time process algebra capable of (i) modeling processes with different probabilistic advancing speeds (mean number of actions executed per time unit), and (ii) expressing probabilistic external/internal choices and multiway synchronization. More precisely, we introduce a variant of CSP that employs a probabilistic asynchronous parallel operator whose synchronization mechanism is based on a mixture of the classical generative and reactive models of probability. In particular, differently from existing discrete time process algebras where parallel processes are executed in synchronous locksteps, the parallel operator that we adopt allows processes with different mean action frequencies (actions executed per time unit) to be modeled. Moreover, our generative-reactive synchronization mechanism is a simple, but expressive, model of synchronization which makes it possible to always derive discrete time Markov chains in the case of fully specified systems.

We show that, when evaluating steady state based performance measures expressed by associating rewards with actions, such a probabilistic approach provides an exact solution even if advancing speeds of processes are considered not to be probabilistic, i.e. actions of different processes have an exact duration. In this way we can model actions with a different fixed duration without incurring the state space explosion problem which arises with an intuitive application of a standard synchronous approach (parallel processes proceeding in locksteps).

We show the expressive power of our calculus by presenting a non-trivial case study on multi-path routing, where both probabilistic and fixed durations are employed. Such a case study also shows that the language makes it particularly easy to produce scalable specifications.

As a further result we present a sound and complete axiomatization of probabilistic bisimulation over finite processes of our calculus, that is a smooth extension of the axiom system for a standard process algebra, thus solving the open problem of cleanly axiomatizing action restriction in the generative model.

10.1 Introduction

The modeling experience in the specification of complex concurrent systems (see e.g. [8, 9, 28]) has revealed the importance of using languages expressing time (the advancing speed of processes), probabilistic internal/external choices and multi-way synchronizations to be able to represent the behavior of such systems.

In this chapter we address the open problem of developing a calculus for describing and analyzing probabilistic distributed systems that combines, in a natural way, such mechanisms in a discrete time setting. More precisely, we make the assumption that time does not flow continuously, as e.g. in Continuous Time Markov Chains (CTMCs), but the passage of time is represented by a sequence of discrete steps, as in Discrete Time Markov Chains (DTMCs). In particular, we recall from Chapter 2 that DTMCs are labeled transition systems, where the transitions are labeled with their execution probabilities, that represent timed systems performing a transition every discrete time step of the same length 1.

10.1.1 Generative-Reactive Probabilistic Models for Discrete Time Systems

In the past 10 years several models for representing probabilistic systems (see e.g. [80, 145] and the references therein) and several probabilistic extensions of classical process algebras (see e.g. [112, 118, 150, 13, 80, 68]) based on such models have been proposed. In particular, we briefly recall two different models of probability, which have been presented in [80], by resorting to the terminology of Milner [122], where action type based synchronizations are described in terms of *button pushing experiments*. In this view the environment experiments on a process by pushing one of several buttons, where a button represents an action type. According to the *reactive model* of probability, a process reacts internally to a button push performed by its environment on the basis of a probability distribution that depends on the button that is pushed. According to the *generative model* of probability, instead, the process itself autonomously decides, on the basis of a probability distribution, which button will go down and how to behave after such an event.

When two processes, which behave in a reactive way, synchronize on an action of type a, each of them reacts internally to the synchronization according to the probability distribution associated with the actions of type a it can perform. Whenever the two processes can synchronize on more than one action type, each of them leaves the decision to the environment, hence the choice of the synchronizing action type turns out to be non-deterministic. This kind of synchronization is simple and natural [145], but does not allow a mechanism for the choice of the button to be pushed (external choice) to be expressed, thus leaving the system, in a sense, under-specified.

On the other hand, two processes that behave in a generative way, independently decide the action type over which they want to synchronize, hence there may be no agreement on the action type. Almost all the solutions proposed in the literature (see e.g. [80, 13]) are based on the fact that the synchronization on different action types is allowed. In the case the model of synchronization adopted is such that processes can synchronize only on the same action type then, as also stated in [145], there is no simple and intuitive solution to the problem. [1] As an example, let us consider a system composed of two sequential processes that must synchronize over actions of type a and b. Suppose that one process may execute an action a with probability p and an action b with probability $1-p$, while the other process may execute an action a with probability q and an action b with probability $1-q$. This scenario could be represented by a term like $a +^p b \,||_{\{a,b\}}\, a +^q b$. The problem is: How do we choose the synchronization to be performed ? The choice cannot be made independently by the two processes, because e.g. if the lefthand process locally chooses a and the righthand process locally chooses b, then no synchronization is possible between the two processes. In order to have a clear model of synchronization, it is important that, once chosen the set of processes that must cooperate, each of such processes can *independently* choose one of its local actions. The problem is that we do not know who is in control of the system since both processes want to decide the button to be pushed.

A simple and intuitive solution to the problem (as suggested also in [145]) is to adopt a mixed generative-reactive approach by considering an *asymmetric* form of synchronization, where a process that behaves generatively may synchronize only with processes that behave reactively. The intuition is that the process that behaves generatively decides which button will go down (and how to behave afterwards), while the process that behaves reactively just reacts internally to the button push of the other process. The integration of the generative and reactive approaches is naturally obtained (similarly as done in [155] for a probabilistic version of I/O automata [120]) by designating some actions, called *generative* actions, as predominant over the other ones, called *reactive* actions, and by imposing that generative actions can synchronize with reactive actions only. Syntactically, we distinguish reactive actions from generative actions by using the subscript $*$ stating the reactive nature of the action. As an example, let us consider a system composed of two sequential processes that must synchronize over actions of type a and b whose behavior could be described by a term like:

$$(a +^p b) \,||_{\{a,b\}} ((a_* +^q a_*) + (b_* +^r b_*))$$

According to the generative-reactive approach: first we probabilistically choose, locally to the lefthand process, between the two generative actions a and b according to probabilities p and $1-p$ (generative choice); then we probabilistically choose, locally to the righthand process, either between the two reactive actions a according to probabilities q and $1-q$ or between the two reactive actions b according to probabilities

[1] In spite of this, a rather artificial solution has been proposed in [68].

r and $1-r$, depending on whether generative action a or b wins in the first step (reactive choice).

The discrete time probabilistic transition systems that we consider, called *generative-reactive* transition systems (GRTSs), are a mixture of the generative and reactive transition systems of [80]. In a GRTS (a restricted version of the general model of [145]), each transition represents a discrete time step and is labeled with an action, which can be either a generative action a or a reactive action a_*, and a probability. Transitions leaving a state are grouped in several bundles. We have a single generative bundle composed of all the transitions labeled with a generative action and several reactive bundles, each one referring to a different action type a and composed of all the transitions labeled with a_*. A bundle of transitions expresses a probabilistic choice, hence the sum of the probabilities of the transitions composing a bundle must be 1. On the contrary, the choice among bundles is performed non-deterministically. In a GRTS we see the reactive actions as *incomplete* actions, which must synchronize with generative actions of another system component in order to form a complete system. A system is considered to be *fully specified* only when it gives rise to a probabilistic transition system that is purely generative, in the sense that it does not include reactive bundles. Fully specified systems are therefore fully probabilistic systems (systems not including non-deterministic choices [145]), from which a DTMC can be trivially derived by discarding actions from transition labels. As a consequence, they can be easily analyzed to get performance measures. In other words the limited form of non-determinism that we consider is sufficient to have a clear notion of control over action synchronizations, but is harmless from the performance analysis viewpoint, in the sense that for complete systems the underlying probabilistic transition system is fully probabilistic. Moreover, the modeling experience [23, 8, 9, 28, 19] showed that considering, as in the generative-reactive approach, two kinds of actions where one predominates over the other, leads to a master-slave discipline that is very suitable for describing real systems.

10.1.2 Probabilistic Asynchronous Parallel Composition

The parallel operator that we consider is similar to the CSP [109] operator $P \,\|_S\, Q$, where processes P and Q are required to synchronize over actions of the same type in the set S and locally execute all the other actions. Such an operator expresses multi-way synchronizations by assuming that the result of the synchronization of two actions a is again a visible action a. In addition, we impose that a synchronization between two actions of type a may occur only if either they are both reactive actions a_* (and the result is a reactive action a_*), or one of them is a generative action a and the other one is a reactive action a_* (and the result is a generative action a). As a consequence, a multi-way synchronization is composed of all reactive actions except at most one generative action: the choice of a generative action a performed by a process determines the button that is pushed (button a) and the other processes internally react by *independently* choosing one of their reactive actions a_*.

Differently from existing discrete time process algebras, where parallel processes are executed in synchronous locksteps (see e.g. [112, 150, 118]), the parallel composition operator that we adopt is asynchronous and allows processes with different ***probabilistic advancing speeds*** (mean number of actions executed per time unit) to be modeled.

Our approach is inspired by that of [13] where ACP [14] is extended with probability by means of a probabilistic parallel composition operator, even if in [13] time is not explicitly considered. In particular, as in [13], our parallel composition operator is parameterized with a probability p, so that in $P \,||_S^p\, Q$ the process performing the next move is determined according to probability p. As an example, let us consider a system composed of two sequential processes whose behavior is described by a term like: [2]

$$(a +^p b) \,||^q (c +^r d)$$

According to the approach inspired by [13], we first choose which of the two processes must make the next move according to probabilities q and $1 - q$. Then, if the lefthand process wins we locally choose between a and b according to probabilities p and $1 - p$, otherwise we locally choose between c and d according to probabilities r and $1 - r$.

As we will shortly show, in our discrete time setting $P \,||_S^p\, Q$ represents a system where the mean action frequency (mean number of actions executed per time unit) of process P is p, while the mean action frequency of process Q is $1-p$. Since P and Q may advance at different action frequencies, with respect to the classical synchronous approach, modeling a concurrent system does not necessarily imply adopting the same duration for the actions of P and Q. For instance, we could model a post office with a priority mail and an ordinary mail service (see Fig. 10.1) simply by the term $P \,||_\emptyset^{0.2}\, Q$, where: process P, representing the ordinary mail service, repeatedly executes actions a expressing the delivery of a letter via ordinary mail; and process Q, representing the priority mail service, repeatedly executes actions b expressing the delivery of a letter via priority mail. Supposed that we take minutes to be the time unit on which the post office specification is based, in $P \,||_\emptyset^{0.2}\, Q$ the mean frequency for the ordinary mail service is 0.2 letters per minute (288 per day) and the mean frequency for the priority mail service is 0.8 letters per minute (1152 per day). Therefore the actions of P take 5 minutes on average to be executed, while the actions of Q take 1 minute and 15 seconds to be executed.

To be more precise, the execution of a system $P \,||_S^p\, Q$ is determined by assuming a probabilistic scheduler that in each global state decides which process between P and Q will perform the next step. In particular, P and Q advance in discrete steps and the scheduler decides who is going to perform the next move by tossing an unfair coin that gives "head" with probability p and "tail" with probability $1-p$. If the coin gives "head" P moves, if the coin gives "tail" Q moves. After a certain number, let us say n, of coin tosses, i.e. after n time units, it turns out that the mean number of heads that have occurred (steps P has made) is $n \cdot p$ while the mean number of tails that have occurred (steps Q has made) is $n \cdot (1 - p)$. Formally, such mean values

[2] We use $||^q$ instead of $||_S^q$ when $S = \emptyset$.

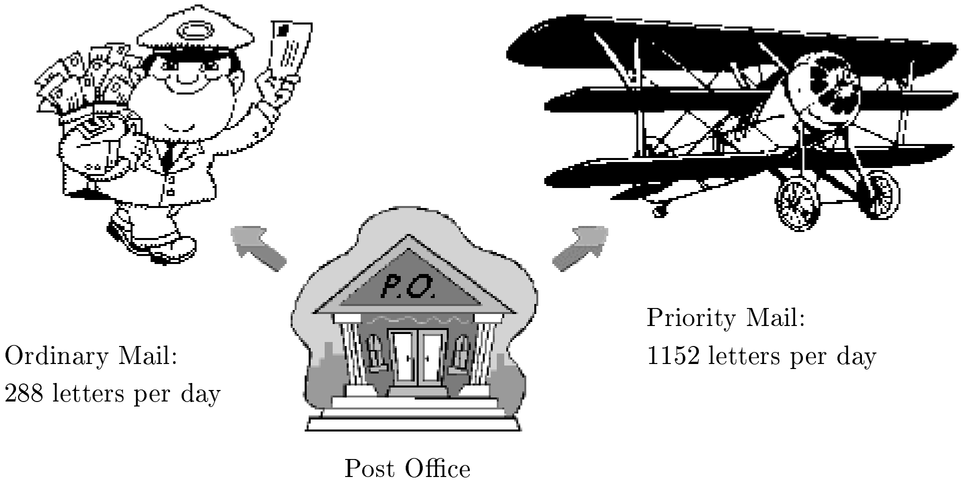


Figure 10.1: The Post Office Example

are derived in the following way: $n \cdot p$ is the mean value of a discrete random variable following a binomial distribution with parameters n (number of experiments) and p (probability of success for each experiment). This means that P performs a mean of p steps per time unit and Q performs a mean of $1-p$ steps per time unit. Hence p is P's probabilistic advancing speed while $1-p$ is Q's probabilistic advancing speed.

Note that in the representation of the behavior of a system $P \,||_S^p\, Q$ we do not express an actual concurrent execution of the actions of P and Q (so that when time passes for an action of P then it passes also for a concurrent action of Q). The behavior of a system $P \,||_S^p\, Q$ is, instead, described at a higher level of abstraction by a model with discrete time steps where there is no actual parallel execution, but only an interleaving of the steps of the two processes, where each step takes the same amount of time (a time unit). To make this more clear, we can interpret the behavior of $P \,||_S^p\, Q$ as originated by a single-processor machine executing both processes (P and Q) via a probabilistic scheduler. In this view, the choices in the global states of $P \,||_S^p\, Q$ do not represent "races" between concurrent time delays (as it is usual for continuous time process algebras) but only probabilistic choices that determine which is the process performing the next discrete step. Therefore we do not assume continuous memoryless distributed sojourn times as, e.g., in the time model of [155], but we simply execute a system transition every discrete time unit. In the execution of $P \,||_S^p\, Q$ sometimes we perform a discrete move of P and sometimes we perform a discrete move of Q and what matters, from the viewpoint of performance behavior, is the frequency with which the actions of a given process are executed. Therefore, even if we represent the system behavior at a certain level of abstraction where actions are not concurrently executed but just interleaved, we have that such a representation gives the correct execution frequencies for actions of processes. Representing the system behavior just taking care of execution frequencies (and not actual concurrency) is the level of abstraction necessary in order to evaluate performance properties in discrete time. This because, since the sojourn times in the states of a DTMC before taking a transition are all equal to 1, as opposed to the states of a CTMC, the evaluation of performance measures in a DTMC is entirely based on the execution frequency of transitions.

So far we have explained the behavior of parallel composition in the case of processes performing independent moves. Now we show what happens when we consider also generative-reactive synchronizations. Our approach integrates the generative-reactive approach inspired by [155] with the approach to probabilistic process choice inspired by [13], in that the selection of the action to be executed in a system state is conceptually carried out through two steps (a generative choice determining the action type followed by a reactive choice) each employing probabilistic choice among processes. As an example, let us consider a system composed of four sequential processes whose behavior is described by the term:

$$(a \,\|^{p} (b +^{p'} c)) \,\|_{\{b\}} (b_* \,\|^{q} (b_* +^{q'} b_*))$$

In the first step a generative action is chosen as follows. We first choose which of the two lefthand processes must make a move according to probabilities p and $1-p$. Then, if the process executing a wins then a is the winning generative action. Otherwise we choose between b and c according to probabilities p' and $1-p'$. If the generative action chosen in the first step must not synchronize (as for the actions a and c) then we are done, otherwise (as for the action b) a second step must be undertaken, where we choose a reactive action among those that can synchronize with the winning generative action. We first choose which of the two righthand processes must make a move according to probabilities q and $1-q$. Then, if the process executing a single b_* wins then this is the winning reactive action. Otherwise we choose between the two reactive actions b_* of the other process according to probabilities q' and $1-q'$. Note that extending a generative-reactive approach similar to that of [155] with a probabilistic parallel composition allows us to obtain fully probabilistic models, nevertheless remaining in a discrete time probabilistic setting. The approach of [155], instead, strictly relies on the fact that continuous time is considered. This because by endowing each state of a probabilistic I/O automata with an exponentially distributed sojourn time, the relative advancing speeds of the probabilistic I/O automata involved in a parallel composition are determined by a timed race condition. Moreover, expressing advancing speeds of processes via a parameterized parallel operator (instead of, e.g., via weights attached to the actions they can perform [150]) is also adequate from a modeling viewpoint. This because the modeler can first specify the behavior of processes in isolation and then establish, independently on how they are specified, their relative advancing speed when composing them in parallel. Finally, as we will show by means of a case study, this approach is also convenient because it leads to specifications that are easily scalable.

10.1.3 Modeling Exact Advancing Speeds

We also consider the problem of modeling timed systems where the different advancing speeds at which concurrent processes proceed are not probabilistic. According to this interpretation of the notion of advancing speed, if, e.g., the action frequency of a process P in a parallel composition is $1/n$, with n natural number, then each action of P takes exactly n time units to be executed, instead of n time units on average.

With a standard approach based on a synchronous parallel composition, where parallel processes are executed in synchronous locksteps (see, e.g., [112, 125, 118, 150, 98]), exact advancing speeds could be modeled as follows. Called n_P the duration of the actions of a process P (number of time units taken to execute each action of P), we compute the greater common divisor div of the set of durations n_P where P is a process composing the system. Then for each process P we split each action in n_P/div time units and we take div to be the duration of the time unit in the specification of the whole system. Such an approach has the problem that the state space of the system greatly increases due to the splitting of the actions of processes, especially if actions with largely different durations are employed: since the step reached by a timed action during its execution needs to be counted in the semantics, each action of P gives rise to n_P/div states. Other approaches in the literature that avoid action splitting (see, e.g., [53, 5, 64]) do not give rise to models that can be turned into analyzable stochastic processes. The reason is that they either (as in [53]) produce dense-time models with infinite states, or (as in [5] and [64]) produce models in which the time progress is separately represented within each different process.

Our approach, which is based on probabilistic advancing speeds, constitutes an approximated solution to the problem of modeling timed systems with different exact advancing speeds, in that action frequencies of processes are probabilistic instead of being exact, but it has the advantage that actions are not split. Nevertheless, an important property of our approach is that, in the case of non-blocking processes (i.e. processes enabling at least a generative action in each state), while such an approximation may affect the performance behavior of the system during an initial transient evolution, it gives exact values for the performance measures of interest when the system reaches a stationary behavior. Note that the restriction to non-blocking processes is not a limiting requirement from the modeling viewpoint, because the waiting condition of a process can be expressed by executing generative *idle* actions representing the fact that the process is idle. Therefore, as far as the evaluation of steady state based performance measures of systems is concerned (at least if they are expressible by associating rewards with actions [19]), our approach avoids action splitting, hence the state space explosion problem, while preserving the possibility of exactly analyzing concurrent processes with exact advancing speeds.

10.1.4 Overview and Main Results

We start by formally presenting the probabilistic model for discrete time, based on a mixture of the generative and reactive models of probability, we explained above. In particular we show that, from the models of fully specified systems, we can easily derive DTMCs (Sect. 10.2).

We then introduce a calculus based on the generative-reactive model and the probabilistic asynchronous parallel composition operator explained above. In particular we present its syntax, its formal semantics and we show that a simple extension of the probabilistic bisimulation equivalence [117] is a congruence with respect to all the operators of the calculus and recursive definitions (Sect. 10.3).

Afterwards, we address the problem of finding a sound and complete axiomatization for the probabilistic bisimulation equivalence over the finite processes of our calculus. In order to do this we introduce a new auxiliary operator $\langle P\rangle_K^J$, defined in such a way that if $\langle P\rangle_K^J$ is derived from P by our axiom system, then K denotes which bundles are executable by P and J denotes the set of types of the generative actions executable by P. Thanks to this operator we obtain an axiomatization that is a smooth probabilistic extension of the classical axiomatization for the parallel composition based on the left and synchronization merge operators [13]. In particular, one of the most important consequences of adopting such an operator is that it leads to a clean axiomatization of action restriction in the generative model, without resorting to axioms with implications as in [13]. Moreover, the axioms for the reactive actions are exactly like the axioms for the standard actions except for the calculations related to probability (Sect. 10.4).

We subsequently show that when evaluating performance measures of a system that are based on its behavior at a steady state, our approach provides a correct solution even when the advancing speeds are considered to be exact. In particular, with respect to the standard approach based on a synchronous parallel composition, we avoid the state space explosion problem mentioned above (Sect. 10.5).

Finally, we present a case study that shows all the main features of our calculus: the algebraic model of a router implementing a probabilistic multi-path routing mechanism (Sect. 10.6). In this case study we show that: (*i*) our approach makes it possible to analyze systems whose components are specified through actions with largely different exact durations (in our model we have actions lasting from 0.5 microseconds to 20 milliseconds); (*ii*) expressing advancing speeds of processes via a parameterized parallel operator (instead of, e.g., via weights attached to the actions they can perform) is convenient from a modeling viewpoint because the modeler can first specify the behavior of processes in isolation and then establish, independently of how they are specified, their advancing speeds when composing them in parallel; (*iii*) thanks to the use of our probabilistic parallel operator and to the generative-reactive mechanism, it is possible to define a specification of the router that is easily scalable to an arbitrary size of the routing table.

To terminate the chapter, we report some concluding remarks (Sect. 10.7).

10.2 The Generative-Reactive Model

GRTSs are a mixture of the generative and reactive transition systems of [80]. In a GRTS (a restricted version of the general model of [145]), each transition is labeled with an action, which can be either a generative action a or a reactive action a_*, and a probability. Formally, we denote the set of action types by $AType$, ranged over by $a, b, \ldots$. As usual $AType$ includes the special type τ denoting internal actions. We denote the set of reactive actions by $RAct = \{a_* \mid a \in AType\}$ and the set of generative actions by $GAct = AType$. The set of actions is denoted by $Act = RAct \cup GAct$, ranged over by θ, $\theta', \ldots$.

Transitions leaving a state are grouped in several bundles. We have a single generative bundle composed of all the transitions labeled with a generative action and several reactive bundles, each one referring to a different action type a and composed of all the transitions labeled with a_*. A bundle of transitions expresses a probabilistic choice. On the contrary the choice among bundles is performed non-deterministically.

Definition 10.1 *A probabilistic transition system PTS is a quadruple* $(S, AType, T, s_1)$ *with*

- S *a set of states,*
- $AType$ *a set of action types,*
- $T \in \mathcal{M}(S \times Act \times]0,1] \times S)$ *a multiset*[3] *of probabilistic transitions,*
- *and* $s_1 \in S$ *the initial state.* ■

Definition 10.2 *A Generative-Reactive Transition System (GRTS) is a PTS* $(S, AType, T, s_1)$ *such that*

1. $\forall s \in S, \forall a_* \in RAct,\ \sum\{\!|\, p \mid \exists t \in S : (s, a_*, p, t) \in T \,|\!\} \in \{0, 1\}$
2. $\forall s \in S,\ \sum\{\!|\, p \mid \exists a \in GAct, t \in S : (s, a, p, t) \in T \,|\!\} \in \{0, 1\}$ ■

The first requirement defines the structure of the reactive bundles leaving a state (one for each action type), and the second requirement says that each state must have a unique generative bundle. Both requirements say that for each state the probabilities of the transitions composing a bundle, if there are any, sum up to 1 (otherwise the summation over empty multisets is defined equal to 0).

As an example, we show in Fig. 10.2 a *GRTS* composed of a generative bundle which enables three transitions (a, b, and c), a reactive bundle of type b which enables two transitions b_*, and a reactive bundle of type d. As in [145] we denote transitions of the same bundle by grouping them by an arc, and we omit the probability from a transition label when it is equal to 1.

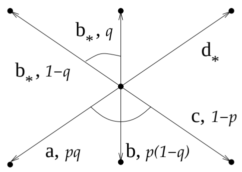


Figure 10.2: Example of a Generative-Reactive Transition System

[3]We use "$\{\!|$" and "$|\!\}$" as brackets for multisets and $\mathcal{M}(S)$ to denote the collection of multisets over set S.

We recall that reactive actions are seen as *incomplete* actions which must synchronize with generative actions of another system component in order to form a complete system. Therefore a fully specified system is *performance closed*, in the sense that it gives rise to a fully probabilistic transition system which does not include reactive bundles. From such a transition system a DTMC can be trivially derived by discarding actions from transition labels. Such a DTMC can, then, be easily analyzed to get performance measures of systems. Formally, a (homogeneous) DTMC on a set of states S (see Chapter 2) is defined by a transition matrix $\mathbf{P} = [p_{i,j}]_{i,j \in S}$, where $p_{i,j}$ represents the probability of going from the state i to the state j, and a vector of initial probabilities which associates with each state $i \in S$ the probability of being the starting state. From a *GRTS* $(\{s_1, \ldots, s_n\}, AType, T, s_1)$ not including reactive bundles we derive the corresponding DTMC $\mathbf{P} = [p_{i,j}]_{i,j \in \{1,\ldots,n\}}$ (whose vector of initial probabilities assigns probability 1 to state s_1 and probability 0 to all the other states) as follows:

$$p_{i,j} = \sum \{\!| \, p \mid \exists \, a \in GAct : (s_i, a, p, s_j) \in T \, |\!\}$$

If the system under analysis reaches a steady behavior at a limiting execution time, we can evaluate its performance by studying the (time averaged) stationary state probabilities of the underlying DTMC $\mathbf{P}$ (see Chapter 2). This is done by computing the vector of state probabilities $\boldsymbol{\pi} = [\pi_i]_{i \in \{1,\ldots,n\}}$ (such that $\sum_{i \in \{1,\ldots,n\}} \pi_i = 1$) that solves the global balance equation:

$$\boldsymbol{\pi} \cdot \mathbf{P} = \boldsymbol{\pi}$$

The i-th element π_i of the steady state probability vector $\boldsymbol{\pi}$ represents the probability that $\mathbf{P}$ is in state i when observed at a steady behavior.

Analyzing the system for a particular performance measure can be done by associating bonus *rewards* (see Chapter 2) with the generative actions occurring in the system algebraic specification. In particular we express the performance measure that we want to evaluate in terms of the reward r_a earned by the execution of an action of type a according to such a performance measure. In order to compute the value m of the performance measure we have to evaluate the reward structure $\mathbf{B} = [b_{i,j}]_{i,j \in \{1,\ldots,n\}}$ associated to the DTMC $\mathbf{P}$ underlying the system. This is done as follows:

$$b_{i,j} = \sum \{\!| \, r_a \cdot \frac{p}{p_{i,j}} \mid \exists \, a \in GAct : (s_i, a, p, s_j) \in T \, |\!\}$$

where $p/p_{i,j}$ is the probability that the transition with reward r_a is performed given that we perform a transition from state i to state j. We use such probabilities to aggregate the rewards of several transitions going from state i to state j into a single reward.

Finally, we recall from Chapter 2 that the performance measure m is computed as follows:

$$m = \sum_{i \in \{1,\ldots,n\}} \; \sum_{j \in \{1,\ldots,n\}} \pi_i \cdot b_{i,j} \cdot p_{i,j}$$

10.3 The Calculus: Syntax and Semantics

In this section we introduce the generative-reactive calculus. We begin by presenting the syntax of the calculus and a detailed explanation of the probabilistic parallel composition. We then introduce the calculus operational semantics which generates GRTSs from process terms. Finally, we present a notion of strong probabilistic bisimulation, which is shown to be a congruence for the calculus.

10.3.1 Syntax

The syntax of the generative-reactive process algebra is defined as follows.

Let *Const* be a set of constants, ranged over by $A, B, \ldots$.

Definition 10.3 *The set $\mathcal{L}$ of process terms is generated by the syntax:*

$$P ::= \underline{0} \mid \theta.P \mid P +^p P \mid P \parallel_S^p P \mid P[a \to b]^p \mid A$$

where $a \in AType - \{\tau\}, b \in AType, S \subseteq AType - \{\tau\}$, $p \in]0,1[$. The set $\mathcal{L}$ is ranged over by $P, Q, \ldots$. We denote by $\mathcal{G}$ the set of guarded and closed terms of $\mathcal{L}$. ■

$\underline{0}$ represents a terminated or deadlocked term having no transitions, while the prefix operator $\theta.P$ performs the action θ with probability 1 and then behaves like P.

The alternative composition operator $P +^p Q$ represents a probabilistic choice between the generative actions of P and Q and between the reactive actions of P and Q of the same type. As far as generative actions are concerned, $P +^p Q$ executes a generative action of P with probability p and a generative action of Q with probability $1-p$. In the case one process P or Q cannot execute generative actions, $P +^p Q$ chooses a generative action of the other process with probability 1 (similarly as in [13]). As far as reactive actions of a given type a are concerned, $P +^p Q$ chooses between the reactive actions a_* of P and Q according to probability p, by following the same mechanism.

Example 10.1 As an example, [4] $a +^p b$ and $b_* +^p b_*$ represent purely probabilistic choices made according to parameter p (see the corresponding *GRTS*s in Fig. 10.3(a)). On the other hand, $a +^p b_*$ and $a_* +^p b_*$ represent purely non-deterministic choices, where the parameter p is not considered (see Fig. 10.3(b)). Finally, $(a +^{p'} b_*) +^p (b +^{p''} b_*)$ represents a mixed probabilistic and non-deterministic choice where parameters p' and p'' are not considered and therefore we have: one generative bundle made up of two transitions a and b, with probabilities p and $1-p$ respectively, and one reactive bundle of type b made up of two transitions b_*, with probabilities p and $1-p$, respectively (see Fig. 10.3(c)). ■

The parallel composition operator $P \parallel_S^p Q$ is based on a CSP like synchronization policy, where processes P and Q are required to synchronize over actions of type in the set S, and locally execute all the other actions.

[4] We abbreviate terms $\theta.\underline{0}$ by omitting the final $\underline{0}$.

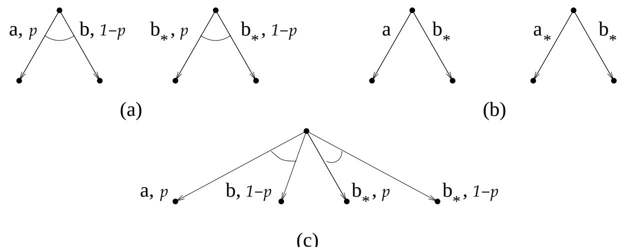


Figure 10.3: Some Examples of GRTSs Derived from Alternative Composition

Moreover, as already explained in the introduction, we impose that a synchronization between two actions of type a may occur only if either they are both reactive actions a_* (and the result is a reactive action a_*), or one of them is a generative action a and the other one is a reactive action a_* (and the result is a generative action a). As far as generative actions are concerned, the generative actions of P (Q) executable by $P \,\|_S^p\, Q$ are such that either their type a is not in S, or a is in S and Q (P) can perform some reactive action a_*. In particular, as standard when restricting actions in the generative model [80], the probabilities of executing such actions are proportionally redistributed so that their overall probability sums up to 1. The choice among the generative actions of P and Q executable by $P \,\|_S^p\, Q$ is made according to probability p, by following the same probabilistic mechanism seen for alternative composition. In the case of synchronizing generative actions a of P (Q), their probability is further redistributed among the reactive actions a_* executable by Q (P), according to the probability they are chosen in Q (P). As far as reactive actions of a given type $a \notin S$ are concerned, $P \,\|_S^p\, Q$ may perform all the reactive actions a_* executable by P or Q and the choice among them is made according to probability p, by following the same probabilistic mechanism seen for alternative composition. As far as reactive actions of a given type $a \in S$ are concerned, if both P and Q may execute some reactive action a_*, the choice of the two actions a_* of P and Q forming the actions a_* executable by $P \,\|_S^p\, Q$ is made according to the probability they are *independently* chosen by P and Q.

Example 10.2 As a first example, consider the term $P_1 \equiv ((a +^q b) +^{q'} c) \,\|_{\{b\}}^p\, \underline{0}$. The term P_1 may execute the generative actions a and c only, because action b must synchronize but does not have a reactive counterpart in the righthand process. Since term $((a +^q b) +^{q'} c)$ in isolation executes the generative action a with probability $q' \cdot q$, the generative action b with probability $q' \cdot (1 - q)$ and the generative action c with probability $1 - q'$, we redistribute the probabilities of a and c by dividing them by the sum of their probabilities $q' \cdot q + 1 - q'$. Therefore term P_1 executes action a with probability $(q' \cdot q)/(q' \cdot q + 1 - q')$ and action c with probability $(1 - q')/(q' \cdot q + 1 - q')$ (see the *GRTS* of Fig. 10.4(a)).

As a second example, consider the term $P_2 \equiv ((a +^q b) +^{q'} c_*) \,\|_{\{c\}}^p\, c$ which may execute the two non-synchronizing generative actions a and b of the lefthand process and the synchronizing generative action c of the righthand process (note that the execution of c is allowed by the reactive action c_* of the lefthand term). As in the case of alternative composition, since both processes may execute some generative actions, we perform a generative action of the lefthand process with probability p and a generative action of the righthand process with probability $1 - p$. Since the generative actions executable by $(a +^q b) +^{q'} c_*$ are a with probability q and b with probability $1 - q$ (according to the probabilistic mechanism of alternative composition q' is not considered), term P_2 executes action a with probability $p \cdot q$, action b with probability $p \cdot (1 - q)$ and action c with probability $1 - p$ (see the *GRTS* of Fig. 10.4(b)).

As a third example, consider the term $P_3 \equiv b \,\|^p_{\{b\}} (b_*.c +^q b_*.d)$ where only the lefthand process may execute some generative action. Therefore, as in the case of alternative composition, we execute the unique generative action b with probability 1 and parameter p is not considered. Such an action synchronizes with one of the two reactive actions b_* of the righthand process chosen according to probability q. Therefore we execute the action b leading to $\underline{0} \,\|^p_{\{b\}}\, c$ with probability q and the action b leading to $\underline{0} \,\|^p_{\{b\}}\, d$ with probability $1 - q$ (see the *GRTS* of Fig. 10.4(c)).

As a final example, consider the term $P_4 \equiv (a +^{q'} b_*) \,\|^p_{\{b\}} (b_*.c +^q b_*.d)$. As far as the generative actions executable by P_4 are concerned, only the lefthand process may execute some generative action, hence, as in the case of alternative composition, we execute the unique generative action a with probability 1 and parameter p is not considered. As far as the reactive actions are concerned, P_4 may execute reactive actions of type b because b belongs to the synchronization set and both lefthand and righthand processes may execute some action b_*. The probability associated with an action b_* obtained by synchronizing an action b_* of the lefthand process with an action b_* of the righthand process is given by the probability that such actions are independently chosen by the two processes among their reactive actions of type b. Therefore, since in our example the lefthand process may perform only one reactive action b_*, P_4 executes the action b_* leading to $\underline{0} \,\|^p_{\{b\}}\, c$ with probability q and the action b_* leading to $\underline{0} \,\|^p_{\{b\}}\, d$ with probability $1 - q$ (see the *GRTS* of Fig. 10.4(d)). ■

The relabeling operator $P[a \to b]^p$ turns actions of type a into actions of type b. The parameter p expresses the probability that reactive actions b_* obtained by relabeling actions a_* of P are executed with respect to the actions b_* previously performable by term P. As an example, consider the second *GRTS* of Fig. 10.3(b), corresponding to the process $P \stackrel{\Delta}{=} a_* +^q b_*$, where the choice is purely nondeterministic. If we apply the relabeling operator $P[a \to b]^p$ we obtain the process represented by the second *GRTS* of Fig. 10.3(a), where the semantics of $P[a \to b]^p$ is a probabilistic choice between the action b_* obtained by relabeling the action a_* and the other action b_*, performed according to probabilities p and $1-p$, respectively. In this way the probabilistic information p provided in the operator $P[a \to b]^p$ guarantees that the relabeling operator does not introduce non-determinism between reactive actions of the same type. Parameter p is, instead, not used when relabeling generative actions because the choice between generative actions of type a and b in P is already probabilistic. Note that, since b may be the internal action τ, $P[a \to b]^p$ may behave also as an hiding operator.

Finally, constants A are used to specify recursive systems. In general, when defining an algebraic specification, we assume a set of constant defining equations of the form $A \stackrel{\Delta}{=} P$ to be given.

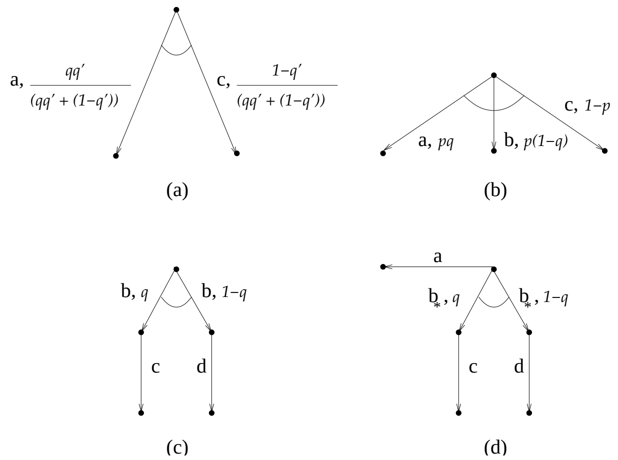


Figure 10.4: Some Examples of GRTSs Derived from Parallel Composition

In order to avoid ambiguities, we introduce the following operator precedence relation: prefix > relabeling > parallel composition > alternative composition. Moreover we use the following abbreviations to denote the hiding and relabeling of several action types. Let "P/L", where L is a finite sequence $\langle a_1^{p_1}, \ldots, a_n^{p_n} \rangle$ of actions $a_i \neq \tau$ with an associated probability p_i, stand for the expression $P[a_1 \to \tau]^{p_1} \ldots [a_n \to \tau]^{p_n}$, hiding the actions with types $a_1, \ldots, a_n$. In a similar way, let "$P[\varphi]$", where φ is a finite sequence $\langle (a_1, b_1)^{p_1}, \ldots, (a_n, b_n)^{p_n} \rangle$ of pairs of actions (a_i, b_i) such that $\tau \notin \{a_i, b_i\}$, with an associated probability p_i, stand for the expression $P[a_1 \to b_1]^{p_1} \ldots [a_n \to b_n]^{p_n}$, relabeling the actions of type $a_1, \ldots, a_n$ with the visible actions $b_1, \ldots, b_n$, respectively. For the sake of simplicity, we assume the parameter p to be equal to $\frac{1}{2}$ whenever it is omitted from any operator of our calculus.

Note that we employ a unique alternative composition operator, instead of a purely probabilistic one and a purely non-deterministic one for two main reasons. First, it permits to circumscribe all the non-determinism to bundles of different kinds, so that fully specified systems are performance closed. Second, a unique probabilistic choice operator is the natural counterpart of the unique parallel composition operator $P \parallel_S^p Q$ (having two parallel composition operators is not convenient because it makes it impossible to express mixed non-deterministic and probabilistic choices between the same two processes). An important consequence is that, as we will see in Sect. 5, we can simply characterize parallel composition by means of alternative composition.

Now, in order to make the reader more familiar with the probabilistic operators of the language, we present a small example that employs the main features of the mixed generative-reactive approach on which our calculus is based.

Example 10.3 Let us consider a system composed of a producer and a buffer. The overall system can be described as the interaction of two processes:

$$Producer \parallel_{\{produce\}}^{p} Buffer$$

The probabilistic parallel composition operator "$\_ \|_S^p \_$" is used to express a set of concurrent system components, their communication interface, and their probabilistic advancing speed in the way we will describe.

The communication interface $\{produce\}$ says that the two processes can interact by synchronously executing actions of type *produce*. Each other local action is asynchronously executed by the two processes. Probability p is the parameter of a probabilistic scheduler which, at each discrete time step (whose duration is the *time unit* on which the system specification is based), decides which of the two processes must be scheduled: in each system state the process between *Producer* and *Buffer* that will execute the next move is probabilistically chosen according to probability p and $1-p$, respectively. As already explained in Sect. 10.1.2 this means that process *Producer* executes a mean of p actions per time unit, while process *Buffer* executes a mean of $1-p$ actions per time unit.

Now let us detail each component. Process *Producer* repeatedly produces new items:

$$Producer \stackrel{\Delta}{=} produce.Producer +^q \tau.Producer$$

The probabilistic alternative choice operator "$\_ +^q \_$" says that the producer can either produce a message (action *produce*) with probability q, or stay idle (action τ) with probability $1-q$, and afterwards behaving as the same process *Producer*. The actions *produce* and τ are *generative*, hence according to the *generative model* of probability [80], the process itself autonomously decides, on the basis of a probability distribution, which action will be executed and how to behave after such an event.

Process *Buffer*, instead, is ready to accept new incoming items or it stays idle:

$$Buffer \stackrel{\Delta}{=} (produce_*.discard.Buffer +^r produce_*.store.Buffer) +^{r'} \tau.Buffer$$

The two actions $produce_*$ allow a synchronous interaction to be activated with an external process through a synchronization with an action of type *produce*. The potential interaction is guided by the external process (the producer) and, whenever this is the case, the buffer reacts by choosing either the first action $produce_*$ with probability r and then discarding the message (generative action *discard*), or the second action $produce_*$ with probability $1-r$ and then storing the message (generative action *store*). The two actions $produce_*$ are *reactive*, hence according to the *reactive model* of probability [80], the process reacts internally to an external action of type *produce* (performed by its environment) on the basis of a probability distribution associated with the reactive actions of type *produce* it can perform. In practice, the buffer reacts to stimuli presented by its environment in the form of synchronizing actions of type *produce*. This reflects a *master-slave* discipline, where a master (the producer) decides the action to execute and the slave (the buffer) reacts to its decision (reactive actions are incomplete actions which must synchronize with generative actions of another system component in order to be executed). In our example, process *Buffer* stays idle by executing internal actions τ while it is not getting items from the *Producer*. The choice between the reactive actions $produce_*$ and such an internal event is just nondeterministic (parameter r' is not considered), because the execution of a reactive action $produce_*$ by means of process *Buffer* is entirely guided by the process *Producer*. Whenever the process

Producer performs its generative action *produce* the process *Buffer* will execute one of its $produce_*$ actions. For instance, in the initial state $Producer \,\|^{p}_{\{produce\}}\, Buffer$ of our example, the system executes a move of process *Producer* with probability p: it executes either the internal move τ with probability $p \cdot (1-q)$, or the move *produce* with probability $p \cdot q$ (with probability $p \cdot q \cdot r$ it executes a *produce* action synchronized with the first reactive action of *Buffer* and with probability $p \cdot q \cdot (1-r)$ a *produce* action synchronized with the second reactive action of *Buffer*). On the other hand, the system may schedule with probability $1-p$ the process *Buffer* by executing its internal action τ (which gets the *entire* probability $1-p$ associated to process *Buffer*). ■

10.3.2 Parallel Composition of Processes with Different Action Durations

As already explained in Sect. 10.1.2 our probabilistic parallel composition operator $P \,\|^{p}_{S}\, Q$ can be used to express the concurrent execution of processes P and Q specified with respect to different action durations. Before showing how this can be done in the general case we recall some basic concepts concerning the semantics of our parallel operator.

In the semantic model of $P \,\|^{p}_{\emptyset}\, Q$ the parallel execution of processes P and Q is represented as being originated by a single-processor machine executing both processes via a probabilistic scheduler. In each global state the scheduler probabilistically decides if P or Q is going to perform the next move according to probabilities p and $1-p$, respectively. In this way, as we explained in Sect. 10.1.2, P performs a mean of p actions per time unit (P is executed with action frequency p) and Q performs a mean of $1-p$ actions per time unit (Q is executed with action frequency $1-p$).

If we strictly follow this single-processor interpretation of the semantics of $P \,\|^{p}_{S}\, Q$, we assume that the specifications of processes P and Q are based on the same time unit u representing action duration, and consequently that actions of $P \,\|^{p}_{S}\, Q$ also take time u to be executed, i.e. u is also the time unit of $P \,\|^{p}_{S}\, Q$. Since P and Q must share a single resource (the processor), the effect of putting P in parallel with Q is that both P and Q get slowed down. In particular, when P (Q) is considered in isolation it executes one action per time unit u; when, instead, it is assumed to be in parallel with Q (P) by means of $P \,\|^{p}_{\emptyset}\, Q$, it executes p ($1-p$) actions per time unit u.

On the other hand such an action interleaving based representation of $P \,\|^{p}_{S}\, Q$ can be interpreted as being an abstract description of the actual concurrent execution of two processes P and Q specified with respect to (possibly) different action durations, as in the case of the post office example of Sect. 10.1.2. In general, if f_P is the mean action frequency assumed in the specification of P (an action of P is assumed to take time $1/f_P$ on average to be executed) and f_Q is the mean action frequency assumed for the specification of Q, it is easy to derive a time unit u and a probability p such that, if we assume actions to take time u to be executed, $P \,\|^{p}_{\emptyset}\, Q$ represents the actual concurrent execution of processes P and Q with mean action frequencies f_P and f_Q, respectively. Since the mean action frequency of process P is f_P and the mean action frequency of process Q is f_Q, the mean action frequency of the parallel composition of P and Q must be

$f = f_P + f_Q$. Therefore the time unit representing the duration of the actions of $P \|_{\emptyset}^{p} Q$ that we have to consider is $u = 1/f = 1/(f_P + f_Q)$, and the action frequency p of P with respect to the new time unit u is given by $f_P = p/u = p \cdot f$, hence $p = f_P/f = f_P/(f_P + f_Q)$. Similarly the action frequency $1-p$ of Q with respect to u turns out to be $1 - p = f_Q/f = f_Q/(f_P + f_Q)$. It is worth noting that by adopting a suitable time unit in this way, the speed at which P and Q are executed when they are considered in isolation is not reduced when they are executed in parallel.

Example 10.4 Let us consider a system composed of a server managing a video-conference and a mail server. The first server must broadcast the flow of video images, hence requires a high speed channel to deliver data, while the second server mainly deals with textual messages, hence has smaller requirements in terms of channel bandwidth.

Let us first consider the scenario where the two servers share a single high speed channel which can transmit 500 data packets per second. Since both servers must send data over the same channel we assume a probabilistic scheduling policy for accessing the channel. In particular, since the video-conference server requires much larger bandwidth than the mail server we assume that every 2 milliseconds (the time it takes to send a packet through the channel) we schedule: a data packet coming from the video-conference server with probability 0.8, a data packet coming from the mail server with probability 0.2. In the system algebraic specification we represent the transmission over the channel of a packet coming from the video-conference server with actions a and the transmission over the channel of a packet coming from the mail server with actions b. The system is then specified by the term $A \|_{\emptyset}^{0.8} B$, where $A \triangleq a.A$ (which repeatedly executes actions a) describes the behavior of the video-conference server and $B \triangleq b.B$ describes the behavior of the mail server. The time unit that we adopt is 2 milliseconds. If we consider the specification A in isolation, the video-conference server has at its disposal the full bandwidth of the channel, hence it transmits a packet every time unit, i.e. 500 data packets per second. The same happens if we consider the specification B of the mail server in isolation. When instead we consider the system $A \|_{\emptyset}^{0.8} B$, since the two servers must share a unique channel, the effect that we obtain is that both servers get slowed down. In particular the video-conference server transmits 0.8 packets per time unit, i.e. 400 packets per second, while the mail server transmits 0.2 packets per time unit, i.e. 100 packets per second.

We can give a different interpretation to the same specification $A \|_{\emptyset}^{0.8} B$ by considering it as the description of the two servers in a different scenario where each of them has a dedicated channel: the video-conference server has an outgoing channel which can transmit a mean of 400 packets per second, while the mail server has an outgoing channel which can transmit a mean of 100 packets per second. According to this interpretation the actions a executed by process A (considered in isolation) represent the transmission of a packet of the video-conference server over its dedicated channel and take a mean of 2.5 milliseconds to be executed, while the actions b executed by process B (considered in isolation) represent the transmission of a packet of the mail server over its dedicated channel and take a mean of 10 milliseconds to be executed. In order to specify the actual concurrent execution of processes A and B we have to consider the global action frequency of the two processes, i.e. the total number $400 + 100$ of packets per second they transmit, and take as time unit the mean time to send a packet in the overall system, i.e. $1/(400 + 100)$ seconds or equally 2 milliseconds. Then we have to compute the action frequency p of A with respect to such a time unit. 400 packets per second correspond to $0.8 = 400/500$ packets per 2 milliseconds. Similarly the action frequency $1 - p$ of B with respect to the adopted time unit turns out to be $0.2 = 100/500$ packets per 2 milliseconds. Therefore the specification $A \,\|_{\emptyset}^{0.8}\, B$ with respect to the time unit 2 milliseconds represents the actual parallel execution of the two servers. In particular, differently from the case of the shared channel their advancing speed is not reduced when they are executed in parallel. ■

10.3.3 Semantics

The formal semantics of our calculus maps terms onto GRTSs, where each transition label is composed of an action and a probability.

We assume the following abbreviations that will make the definition of the semantic rules easier. We use $P \xrightarrow{\theta}$ to stand for $\exists\, p, P' : P \xrightarrow{\theta, p} P'$, meaning that P can execute action θ and $P \xrightarrow{G}$ to stand for $\exists\, a \in G : P \xrightarrow{a}$, $G \subseteq AType$, meaning that P can execute a generative action of type belonging to set G.

The GRTS deriving from a term $\mathcal{G}$ is defined by the operational rules in Tables 10.1 and 10.2, where in addition to the rules $(\mathrm{r}2_l) - (\mathrm{r}5_l)$ and $(\mathrm{g}2_l) - (\mathrm{g}7_l)$ referring to a local move of the lefthand process P, we consider also the symmetrical rules $(\mathrm{r}2_r) - (\mathrm{r}5_r)$ and $(\mathrm{g}2_r) - (\mathrm{g}7_r)$ taking into account the local moves of the righthand process Q, obtained by exchanging the roles of terms P and Q in the premises and, for rules $(\mathrm{r}2_r)$, $(\mathrm{r}4_r)$, $(\mathrm{g}2_r)$, $(\mathrm{g}4_r)$ and $(\mathrm{g}6_r)$, by replacing p with $1-p$ in the label of the derived transitions. Similarly as in [107], we consider the operational rules as generating a multiset of transitions (consistently with the definition of a *GRTS*), where a transition has arity n if and only if it can be derived in n possible ways from the operational rules. Note that even if the operational rules in Tables 10.1 and 10.2 include negative premises, this does not cause inconsistencies because when applying such rules for deriving the moves of a term P, the negative premises always refer to the moves of a subterm of P (and not of P itself), hence the operational semantics is stratifiable [89].

Rule (r1) states that a term $a_*.P$ may execute a single reactive bundle of type a composed of a single transition leading to state P.

Rules (r2) state that, for any type a, the reactive bundle a executable by $P +^p Q$ is obtained by redistributing the probabilities of the transitions composing the reactive bundle a of P and Q, according to p and $1 - p$, respectively. In the case the reactive bundle a of P or Q is empty, according to (r3), $P +^p Q$ simply inherits the reactive bundle a of the other process, without considering the parameter p.

Rules (r4) and (r5) state that, for any type $a \notin S$, the reactive bundle a executable by $P \,\|_S^p\, Q$ is obtained as previously explained for the alternative composition. Rule (r6) states that, for any type $a \in S$, if both the reactive bundles a of P and Q are non-empty, then the reactive bundle a of $P \,\|_S^p\, Q$ is non-empty, and is obtained by merging the two reactive bundles a of P and Q. In particular, the probability assigned

$$(\mathbf{r1})\quad a_*.P \xrightarrow{a_*,1} P$$

$$(\mathbf{r2_l})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,p\cdot q} P'} \qquad (\mathbf{r3_l})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,q} P'}$$

$$(\mathbf{r4_l})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P \,\|_S^p\, Q \xrightarrow{a_*,p\cdot q} P' \,\|_S^p\, Q}\ a \notin S \qquad (\mathbf{r5_l})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P \,\|_S^p\, Q \xrightarrow{a_*,q} P' \,\|_S^p\, Q}\ a \notin S$$

$$(\mathbf{r6})\ \frac{P \xrightarrow{a_*,q} P' \ Q \xrightarrow{a_*,q'} Q'}{P \,\|_S^p\, Q \xrightarrow{a_*,q\cdot q'} P' \,\|_S^p\, Q'}\ a \in S$$

$$(\mathbf{r7})\ \frac{P \xrightarrow{a_*,q} P' \quad P \xrightarrow{b_*}}{P[a \to b]^p \xrightarrow{b_*,p\cdot q} P'[a \to b]^p} \qquad (\mathbf{r8})\ \frac{P \xrightarrow{a_*,q} P' \quad P \not\xrightarrow{b_*}}{P[a \to b]^p \xrightarrow{b_*,q} P'[a \to b]^p}$$

$$(\mathbf{r9})\ \frac{P \xrightarrow{b_*,q} P' \quad P \xrightarrow{a_*}}{P[a \to b]^p \xrightarrow{b_*,(1-p)\cdot q} P'[a \to b]^p} \qquad (\mathbf{r10})\ \frac{P \xrightarrow{b_*,q} P' \quad P \not\xrightarrow{a_*}}{P[a \to b]^p \xrightarrow{b_*,q} P'[a \to b]^p}$$

$$(\mathbf{r11})\ \frac{P \xrightarrow{c_*,q} P'}{P[a \to b]^p \xrightarrow{c_*,q} P'[a \to b]^p}\ c \notin \{a,b\}$$

$$(\mathbf{r12})\ \frac{P \xrightarrow{a_*,q} P'}{A \xrightarrow{a_*,q} P'}\ \text{if } A \triangleq P$$

Table 10.1: Semantic Rules for Reactive Transitions

$$(\mathbf{g1})\quad a.P \xrightarrow{a,1} P$$

$$(\mathbf{g2_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{GAct}}{P +^p Q \xrightarrow{a,p\cdot q} P'} \qquad (\mathbf{g3_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \stackrel{GAct}{\not\longrightarrow}}{P +^p Q \xrightarrow{a,q} P'}$$

$$(\mathbf{g4_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{G_{S,P}}}{P \,\|_S^p\, Q \xrightarrow{a,p\cdot q/\nu_P(G_{S,Q})} P' \,\|_S^p\, Q}\; a \notin S$$

$$(\mathbf{g5_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \,\|_S^p\, Q \xrightarrow{a,q/\nu_P(G_{S,Q})} P' \,\|_S^p\, Q}\; a \notin S$$

$$(\mathbf{g6_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \xrightarrow{G_{S,P}}}{P \,\|_S^p\, Q \xrightarrow{a,p\cdot q'\cdot q/\nu_P(G_{S,Q})} P' \,\|_S^p\, Q'}\; a \in S$$

$$(\mathbf{g7_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \,\|_S^p\, Q \xrightarrow{a,q'\cdot q/\nu_P(G_{S,Q})} P' \,\|_S^p\, Q'}\; a \in S$$

$$(\mathbf{g8})\quad \frac{P \xrightarrow{a,q} P'}{P[a \to b]^p \xrightarrow{b,q} P'[a \to b]^p} \qquad (\mathbf{g9})\quad \frac{P \xrightarrow{c,q} P'}{P[a \to b]^p \xrightarrow{c,q} P'[a \to b]^p}\; a \neq c$$

$$(\mathbf{g10})\quad \frac{P \xrightarrow{a,q} P'}{A \xrightarrow{a,q} P'}\ \text{if } A \stackrel{\Delta}{=} P$$

Table 10.2: Semantic Rules for Generative Transitions

to each transition a_* composing the reactive bundle a of $P \,\|_S^p\, Q$, is equal to the probability that the two synchronizing transitions a_* of P and Q are independently chosen by P and Q.

Rules (r7) and (r9) state that, the reactive bundle b executable by $P[a \to b]^p$ is obtained by redistributing the probabilities of the transitions composing the reactive bundles a and b of P, according to p and $1-p$, respectively. As for the alternative composition, if the reactive bundle a of P is empty, according to (r10), $P[a \to b]^p$ simply inherits the reactive bundle b of P, without considering the parameter p. Similarly, if the reactive bundle b of P is empty, according to (r8), the reactive bundle b of $P[a \to b]^p$ is obtained from the reactive bundle a of P by simply relabeling actions in transitions, without considering the parameter p. Rule (r11) states that, for any type $c \notin \{a, b\}$, $P[a \to b]^p$ simply inherits the reactive bundle c of P.

Rule (r12) states that the reactive bundles executable by A such that $A \stackrel{\Delta}{=} P$ are the same reactive bundles executable by P.

Rules (g1), (g2), and (g3) are similar to the corresponding reactive rules. We only point out that the set $GAct$ is used in the premises of rules (g2) and (g3) because all generative actions are grouped in a single bundle.

According to rules (g4) $-$ (g7), the generative transitions of P (Q) executable by $P \,\|_S^p\, Q$ are such that either their type a is not in S, or a is in S and Q (P) can perform some reactive action a_*. In rules (g4) $-$ (g7) we assume the set $G_{S,P} \subseteq AType$, with $S \in AType - \{\tau\}$ and $P \in \mathcal{G}$, to be defined as follows:

$$G_{S,P} = \{a \in AType \mid a \notin S \vee (a \in S \wedge P \xrightarrow{a_*})\}$$

In this way $G_{S,Q}$ ($G_{S,P}$) is the set of types of the generative transitions of P (Q) executable by $P \,\|_S^p\, Q$. Since we consider a restricted set of executable actions, as explained in Sect. 3.1 we redistribute the probabilities of the generative transitions of P (Q) executable by $P \,\|_S^p\, Q$ so that their overall probability sums up to 1 [80]. To this aim in semantic rules we employ the function $\nu_P(G) : \mathcal{P}(AType) \longrightarrow]0,1]$, with $P \in \mathcal{G}$, defined as follows:

$$\nu_P(G) = \sum \{\!| p \mid \exists P', a \in G : P \xrightarrow{a,p} P' |\!\}$$

which computes the sum of the probabilities of the generative transitions executable by P whose type belongs to the set G. In this way $\nu_P(G_{S,Q})$ ($\nu_Q(G_{S,P})$) computes the overall probability of the generative transitions of P (Q) executable by $P \,\|_S^p\, Q$ and can be used to normalize the probabilities of the generative transitions of P (Q). Finally, the generative bundle of $P \,\|_S^p\, Q$ is obtained by redistributing the probabilities of the normalized generative transitions of P and Q executable by $P \,\|_S^p\, Q$ according to p and $1-p$, as for the alternative composition. The probability of the generative transitions a of P (Q), such that $a \in S$, must be further distributed among the reactive transitions belonging to the reactive bundle a of Q (P).

Rules (g8) and (g9) state that the generative bundle of $P[a \to b]^p$ is obtained from the generative bundle of P by simply relabeling actions in transitions, without considering the parameter p. Finally, rule (g10) is the exact counterpart of (r12).

Definition 10.4 *The operational semantics of $P \in \mathcal{G}$ is the PTS $[\![P]\!]$ composed of the terms reachable from P according to the operational rules of Tables 10.1 and 10.2. We say that $P \in \mathcal{G}$ is performance closed if and only if $[\![P]\!]$ does not include reactive transitions.* ■

Lemma 10.1 *Let P and Q be two processes of $\mathcal{G}$. If $[\![P]\!]$ and $[\![Q]\!]$ are GRTSs, then $[\![\theta.P]\!]$, $[\![P[a \to b]^p]\!]$, $[\![P +^p Q]\!]$ and $[\![P \parallel^p_S Q]\!]$ are GRTSs. Moreover, recursive definitions preserve the property of being a GRTS.*

Proof *The result simply derives from the fact that, as explained above for each operational rule, the application of an operator preserves the bundle structure of GRTSs.* ■

Theorem 10.1 *If P is a process of $\mathcal{G}$, then $[\![P]\!]$ is a GRTS.*

Proof *The result derives by structural induction by using lemma 10.1.* ■

10.3.4 Equivalence

We now equip the algebra with a probabilistic bisimulation equivalence (along the lines of [117]), which relates systems having the same functional and probabilistic behavior. We first introduce a function capturing the total probability with which a term reaches a given class of terms by executing a given action of *Act*.

Definition 10.5 *We define function $Prob : (\mathcal{G} \times Act \times \mathcal{P}(\mathcal{G})) \longrightarrow [0,1]$ by*

$$Prob(P, \theta, C) = \sum \{\!|\, p \mid P \xrightarrow{\theta,p} P' \wedge P' \in C \,|\!\}$$

■

Definition 10.6 *An equivalence relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a strong probabilistic bisimulation if and only if, whenever $(P, Q) \in \mathcal{B}$, then for all $\theta \in Act$ and equivalence classes $C \in \mathcal{G}/\mathcal{B}$*

$$Prob(P, \theta, C) = Prob(Q, \theta, C)$$

The strong probabilistic bisimulation equivalence, denoted by $\sim_{PB}$, is the union of all the strong probabilistic bisimulations. ■

Theorem 10.2 *$\sim_{PB}$ is a congruence w.r.t. all the algebraic operators and recursive definitions.*

Proof *The most relevant cases are those of parallel composition operator and recursive definitions. In the case of the parallel operator, it suffices to show that $\{(P_1 \parallel^p_S Q, P_2 \parallel^p_S Q) \mid P_1, P_2, Q \in \mathcal{G}. P_1 \sim_{PB} P_2\} \cup Id_{\mathcal{G}}$, where $Id_{\mathcal{G}}$ is the identity relation over $\mathcal{G}$, is a strong probabilistic bisimulation. In the case of recursive definitions, it suffices to apply the technique introduced in [38] (here presented in Chapter 11).* ■

10.4 Axiomatization

In this section we develop an equational characterization of $\sim_{PB}$ for the set $\mathcal{G}_{fin}$ of nonrecursive terms of $\mathcal{G}$, i.e. guarded and closed terms of our generative-reactive calculus not including constants A.

In order to produce a finite axiom system which is sound and complete over $\mathcal{G}_{fin}$ processes we adopt (as e.g. in [13]) the standard technique of expressing the parallel composition operator by means of the left merge $\lfloor^p_S$ and the synchronization merge $|^p_S$ operators. Moreover, in order to achieve completeness we need to introduce an auxiliary operator $\langle P\rangle^J_K$, so that when a term $\langle P\rangle^J_K$ is derived from P by the axiom system, K denotes the bundles executable by P and J denotes the type of the executable generative actions in the case the generative bundle belongs to K. Formally, we define bundle *kinds* as follows. Given a term P, "$\bullet$" is the kind of the generative bundle executable by P, while a is the kind of the reactive bundle of type a executable by P. We denote with $BKind = \{\bullet\} \cup AType$ the set of bundle kinds, ranged over by $\kappa, \kappa', \ldots$

We denote by $\mathcal{G}_{ext}$ the set of terms obtained by extending the syntax of $\mathcal{G}_{fin}$ terms with the auxiliary operators $\lfloor^p_S$ and $|^p_S$, and the new operator $\langle P\rangle^J_K$, where K is a finite set of bundle kinds, i.e. a subset of $BKind$, and $J \subseteq GAct$ is such that $J \neq \emptyset \iff \bullet \in K$.

Intuitively we have to introduce the new operator $\langle P\rangle^J_K$ in order to understand the role of parameter p in a choice $P +^p Q$, which can be probabilistic or non-deterministic depending on which bundles are executable by P and Q. In particular, we have that the choice $\langle P\rangle^J_K +^p \langle Q\rangle^{J'}_{K'}$ is purely probabilistic when there exists κ such that $K = K' = \{\kappa\}$, while it is purely non-deterministic whenever K and K' are disjoint.

In order to make it possible to produce an axiom system which, besides being complete, is also sound, we define an operational semantics for the new operator $\langle P\rangle^J_K$. As for the other operators of our calculus we will show that such semantics produces GRTSs and that $\langle P\rangle^J_K$ satisfies the congruence property.

The role of the operator $\langle P\rangle^J_K$ in our axiom system is just to denote that, when $\langle P\rangle^J_K$ is derived from P, the bundles executable by P are exactly the bundles stated by K and the types of the generative transitions of P are included in J. Therefore the operator $\langle P\rangle^J_K$ must have no effect on the behavior of processes P satisfying such a requirement. However, even if our axiom system cannot produce a term $\langle P\rangle^J_K$ from a process P not satisfying the requirement above, we have to define the semantics of $\langle P\rangle^J_K$ over arbitrary terms P. The idea here is to define $\langle P\rangle^J_K$ in such a way that, independently on which are the bundles executable by P, the bundles executable by $\langle P\rangle^J_K$ satisfy the requirement above. As we will see, defining $\langle P\rangle^J_K$ in this way will lead to the soundness of our axiom system.

The operational semantics of $\langle P\rangle^J_K$ is defined by the semantic rules defined in the first part of Table 10.3. In this table we employ the function $\kappa : Act \longrightarrow BKind$ which determines the kind of bundle a transition belongs to. Formally $\kappa(\theta)$ is defined as follows:

$$\begin{array}{lrcl} \forall a \in GAct. & \kappa(a) & = & \bullet \\ \forall a_* \in RAct. & \kappa(a_*) & = & a \end{array}$$

Rule (e1) eliminates all the transitions of P not executable according to K and J. Rule (e2) creates a

$$(\mathbf{e1})\ \frac{P \xrightarrow{\theta,q} P'}{\langle P\rangle^J_K \xrightarrow{\theta,q} P'} \quad \text{if} \ \begin{array}{l}\kappa(\theta) \in K \wedge \\ (\kappa(\theta) = \bullet \ \Rightarrow\ \theta \in J)\end{array}$$

$$(\mathbf{e2})\ \frac{P \stackrel{a_*}{\not\longrightarrow}}{\langle P\rangle^J_K \xrightarrow{a_*,1} \underline{0}} \quad \text{if } a \in K \qquad (\mathbf{e3})\ \frac{\nu_P(J) < 1}{\langle P\rangle^J_K \xrightarrow{a,1-\nu_P(J)} \underline{0}} \quad \text{if} \ \begin{array}{l}\bullet \in K \wedge \\ a \in J\end{array}$$

$$(\mathbf{lm1})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P \lfloor^p_S Q \xrightarrow{a_*,p\cdot q} P' \|^p_S Q}\ a \notin S \qquad (\mathbf{lm2})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \stackrel{a_*}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a_*,q} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm3})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{G_{S,P}}}{P \lfloor^p_S Q \xrightarrow{a,p\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm4})\ \frac{P \xrightarrow{a,q} P' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a,q/\nu_P(G_{S,Q})} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm5})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \xrightarrow{G_{S,P}}}{P \lfloor^p_S Q \xrightarrow{a,p\cdot q'\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q'}\ a \in S$$

$$(\mathbf{lm6})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a,q'\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q'}\ a \in S$$

$$(\mathbf{sm1})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*,q'} Q'}{P |^p_S Q \xrightarrow{a_*,q\cdot q'} P' \|^p_S Q'}\ a \in S$$

Table 10.3: Semantic Rules for $\langle P\rangle^J_K$, $P \lfloor^p_S Q$, and $P |^p_S Q$.

reactive bundle of type a in the case P cannot execute such a bundle and $a \in K$. Rule (e3) creates a complete generative bundle in the case $\bullet \in K$ and the generative transitions of P whose type is in J do not form a complete generative bundle. The action a used in rule (e3) is any *fixed* arbitrarily chosen generative action $a \in J$. Such an action can be considered as a fixed parameter of the operator $\langle P \rangle_K^J$.

In the second and third part of Table 10.3 we present the obvious operational rules for $\lfloor_S^p$ and $|_S^p$ that derive from those we presented for the parallel operator.

Lemma 10.2 *Let P and Q be two processes of $\mathcal{G}_{ext}$. If $[\![P]\!]$ and $[\![Q]\!]$ are GRTSs then $[\![\langle P \rangle_K^J]\!]$, $[\![P \lfloor_S^p Q]\!]$, and $[\![P \,|_S^p\, Q]\!]$ are GRTSs.*

Proof *The result simply derives from the fact that, as it can be easily verified, the application of the operators $[\![\langle P \rangle_K^J]\!]$, $[\![P \lfloor_S^p Q]\!]$, and $[\![P \,|_S^p\, Q]\!]$ preserve the bundle structure of GRTSs. In particular for the left merge and synchronization merge operators this is a trivial consequence of the fact that the bundle structure is preserved by the parallel operator.* ■

Theorem 10.3 *If P is a process of $\mathcal{G}_{ext}$, then $[\![P]\!]$ is a GRTS.*

Proof *The result derives by structural induction by using lemma 10.1 and lemma 10.2.* ■

Theorem 10.4 *$\sim_{PB}$ is a congruence w.r.t. the auxiliary operators $\langle P \rangle_K^J$, $P \lfloor_S^p Q$ and $P \,|_S^p\, Q$.*

Proof *As far as the operator $\langle P \rangle_K^J$ is concerned the result simply derives from the fact that, if $P_1 \sim_{PB} P_2$ then for any set of action types J we have that $\nu_{P_1}(J) = \nu_{P_2}(J)$, hence $\langle P_1 \rangle_K^J \sim_{PB} \langle P_2 \rangle_K^J$. Moreover, since the equivalence $\sim_{PB}$ does not abstract from τ transitions the congruence w.r.t. the auxiliary operators $P \lfloor_S^p Q$ and $P \,|_S^p\, Q$ is just a trivial consequence of the congruence with respect to the parallel operator (see [1]).* ■

The equational characterization of $\sim_{PB}$ is composed of the set $\mathcal{A}$ of axioms shown in Table 10.4. The axioms $\mathcal{A}_1 - \mathcal{A}_5$ express the basic properties of the choice operator. The axioms $\mathcal{BK}_1 - \mathcal{BK}_3$ generate the brackets $\langle P \rangle_K^J$ denoting the bundles executable by P. The axioms $\mathcal{R}_1 - \mathcal{R}_7$ and $\mathcal{P}$, $\mathcal{LM}_1 - \mathcal{LM}_9$, $\mathcal{SM}_1 - \mathcal{SM}_5$ refer to the relabeling and parallel composition operators, respectively.

Theorem 10.5 *The axiom system $\mathcal{A}$ is sound for $\sim_{PB}$ over processes of $\mathcal{G}_{ext}$.*

Proof *Just a trivial verification of the consistency of the axioms with respect to the operational semantics of the operators. In particular note that the internal angular parentheses in the righthand term of axiom $\mathcal{BK}_3$ are necessary to achieve soundness. If we had considered external angular parentheses only, then the axiom system would have been complete but not sound, as can be seen by taking, e.g. $P \equiv \underline{0}$. Moreover, note that axiom $\mathcal{LM}_6$ is sound because we apply it when the lefthand process is guarded by a synchronizing generative action a waiting for the reactive counterpart a_* in the righthand process.* ■

$(\mathcal{A}_1)$	$P +^p \underline{0}$	$=$	P	
$(\mathcal{A}_2)$	$P +^p P$	$=$	P	
$(\mathcal{A}_3)$	$P +^p Q$	$=$	$Q +^{1-p} P$	
$(\mathcal{A}_4)$	$(\langle P\rangle^{J}_{\{\kappa\}} +^p \langle Q\rangle^{J'}_{\{\kappa\}}) +^q \langle R\rangle^{J''}_{\{\kappa\}}$	$=$	$\langle P\rangle^{J}_{\{\kappa\}} +^{pq} (\langle Q\rangle^{J'}_{\{\kappa\}} +^{(1-p)\cdot q/(1-p\cdot q)} \langle R\rangle^{J''}_{\{\kappa\}})$	
$(\mathcal{A}_5)$	$(\langle P\rangle^{J}_{K} +^p \langle Q\rangle^{J'}_{K'}) +^q R$	$=$	$\langle P\rangle^{J}_{K} +^q (\langle Q\rangle^{J'}_{K'} +^q R)$	$K \cap K' = \emptyset$
$(\mathcal{BK}_1)$	$a.P$	$=$	$\langle a.P\rangle^{\{a\}}_{\{\bullet\}}$	
$(\mathcal{BK}_2)$	$a_*.P$	$=$	$\langle a_*.P\rangle^{\emptyset}_{\{a\}}$	
$(\mathcal{BK}_3)$	$\langle P\rangle^{J}_{K} +^p \langle Q\rangle^{J'}_{K'}$	$=$	$\langle\langle P\rangle^{J}_{K} +^p \langle Q\rangle^{J'}_{K'}\rangle^{J\cup J'}_{K\cup K'}$	
$(\mathcal{R}_1)$	$\underline{0}[a \to b]^p$	$=$	$\underline{0}$	
$(\mathcal{R}_2)$	$\theta.P[a \to b]^p$	$=$	$\theta.(P[a \to b]^p)$	$\theta \notin \{a,\, a_*\}$
$(\mathcal{R}_3)$	$a.P[a \to b]^p$	$=$	$b.(P[a \to b]^p)$	
$(\mathcal{R}_4)$	$a_*.P[a \to b]^p$	$=$	$b_*.(P[a \to b]^p)$	
$(\mathcal{R}_5)$	$(\langle P\rangle^{J}_{K} +^{ND} \langle Q\rangle^{J'}_{K'})[a \to b]^p$	$=$	$\langle P\rangle^{J}_{K}[a \to b]^p +^{ND} \langle Q\rangle^{J'}_{K'}[a \to b]^p$	$a, b \notin K \wedge K \cap K' = \emptyset$
$(\mathcal{R}_6)$	$(\langle P\rangle^{\emptyset}_{\{a\}} +^{ND} \langle Q\rangle^{\emptyset}_{\{b\}})[a \to b]^p$	$=$	$\langle P\rangle^{\emptyset}_{\{a\}}[a \to b]^p +^p \langle Q\rangle^{\emptyset}_{\{b\}}[a \to b]^p$	
$(\mathcal{R}_7)$	$(\langle P\rangle^{J}_{\{\kappa\}} +^q \langle Q\rangle^{J'}_{\{\kappa\}})[a \to b]^p$	$=$	$\langle P\rangle^{J}_{\{\kappa\}}[a \to b]^p +^q \langle Q\rangle^{J'}_{\{\kappa\}}[a \to b]^p$	
$(\mathcal{P})$	$P \parallel^p_S Q$	$=$	$(P \lfloor^p_S Q +^p Q \lfloor^{1-p}_S P) +^{ND} P \vert^p_S Q$	
$(\mathcal{LM}_1)$	$\underline{0} \lfloor^p_S Q$	$=$	$\underline{0}$	
$(\mathcal{LM}_2)$	$a_*.P \lfloor^p_S Q$	$=$	$\underline{0}$	$a \in S$
$(\mathcal{LM}_3)$	$a.P \lfloor^p_S \underline{0}$	$=$	$\underline{0}$	$a \in S$
$(\mathcal{LM}_4)$	$a.P \lfloor^p_S \theta.Q$	$=$	$\underline{0}$	$a \in S \wedge \theta \neq a_*$
$(\mathcal{LM}_5)$	$a.P \lfloor^p_S a_*.Q$	$=$	$a.(P \parallel^p_S Q)$	$a \in S$
$(\mathcal{LM}_6)$	$a.P \lfloor^p_S (Q +^q R)$	$=$	$a.P \lfloor^p_S Q +^q a.P \lfloor^p_S R$	$a \in S$
$(\mathcal{LM}_7)$	$\theta.P \lfloor^p_S Q$	$=$	$\theta.(P \parallel^p_S Q)$	$\theta \notin S$
$(\mathcal{LM}_8)$	$(P +^q \langle Q\rangle^{J}_{\{\bullet\}}) \lfloor^p_S \langle R\rangle^{J'}_{K}$	$=$	$P \lfloor^p_S \langle R\rangle^{J'}_{K}$	$J \subseteq S \wedge J \cap K = \emptyset$
$(\mathcal{LM}_9)$	$(\langle P\rangle^{J}_{K} +^q \langle Q\rangle^{J'}_{K'}) \lfloor^p_S \langle R\rangle^{J''}_{K''}$	$=$	$\langle P\rangle^{J}_{K} \lfloor^p_S \langle R\rangle^{J''}_{K''} +^q \langle Q\rangle^{J'}_{K'} \lfloor^p_S \langle R\rangle^{J''}_{K''}$	$(J \cup J') \cap S \subseteq K''$
$(\mathcal{SM}_1)$	$P \vert^p_S Q$	$=$	$Q \vert^{1-p}_S P$	
$(\mathcal{SM}_2)$	$\underline{0} \vert^p_S Q$	$=$	$\underline{0}$	
$(\mathcal{SM}_3)$	$\theta.P \vert^p_S \theta'.Q$	$=$	$\underline{0}$	$\theta \notin \{a_* \mid a \in S\} \vee \theta \neq \theta'$
$(\mathcal{SM}_4)$	$a_*.P \vert^p_S a_*.Q$	$=$	$a_*.(P \parallel^p_S Q)$	$a \in S$
$(\mathcal{SM}_5)$	$(P +^q Q) \vert^p_S R$	$=$	$P \vert^p_S R +^q Q \vert^p_S R$	

Table 10.4: Axiomatization

The axiom $\mathcal{A}_2$ expresses idempotency and $\mathcal{A}_3$ a form of commutativity which complements the parameter p (as in [13]).

The axiom $\mathcal{A}_4$ expresses associativity for pure probabilistic choices, i.e. choices among terms enabling a single common bundle. The axiom defines how probabilistic parameters must be adjusted so that the probability of executing a given member of the sum is preserved (as in [13]).

Thanks to the properties of commutativity and associativity of pure probabilistic choices we can just use the notation [5]

$$\sum_{i\in\{1,\dots,n\}}[p_i]\langle P_i\rangle^{J_i}_{\{\kappa\}}$$

as a shorthand notation for a probabilistic choice among $\langle P_1\rangle^{J_1}_{\{\kappa\}}\dots\langle P_n\rangle^{J_n}_{\{\kappa\}}$, where p_i is the probability of executing $\langle P_i\rangle^{J_i}_{\{\kappa\}}$.

The axiom $\mathcal{A}_5$ expresses associativity for choices where two of the terms involved enable disjoint sets of bundle kinds. Two particular cases of $\mathcal{A}_5$ lead to two important properties of the choice operator.

By taking $R \equiv \underline{0}$ in $\mathcal{A}_5$ and applying axiom $\mathcal{A}_1$ we obtain:

$$\langle P\rangle^J_K +^p \langle Q\rangle^{J'}_{K'} \;=\; \langle P\rangle^J_K +^q \langle Q\rangle^{J'}_{K'} \quad \text{if } K\cap K' = \emptyset$$

This equation expresses the fact that whenever we have a choice between two terms enabling different bundles, i.e. a pure non-deterministic choice, the probabilistic parameter of the choice is not important. Thanks to this property, supposed $K\cap K'=\emptyset$, we can just use the notation $\langle P\rangle^J_K +^{ND} \langle Q\rangle^{J'}_{K'}$, to stand for any term $\langle P\rangle^J_K +^p \langle Q\rangle^{J'}_{K'}$ obtained for a particular choice of p. [6]

By taking $R\equiv\langle R'\rangle^{J''}_{K''}$, such that $K''\cap K=\emptyset$ and $K''\cap K'=\emptyset$, in $\mathcal{A}_5$ and by taking into account the previous equation we obtain:

$$(\langle P\rangle^J_K +^{ND} \langle Q\rangle^{J'}_{K'}) +^{ND} \langle R\rangle^{J''}_{K''} \;=\; \langle P\rangle^J_K +^{ND} (\langle Q\rangle^{J'}_{K'} +^{ND} \langle R\rangle^{J''}_{K''})$$
$$\text{if } K, K', K'' \text{ are pairwise disjoint}$$

This equation expresses associativity for pure non-deterministic choices, i.e. choices among terms enabling disjoint sets of bundles.

Thanks to the properties of commutativity and associativity of pure non-deterministic choices we can just use the notation

$$\sum_{i\in\{1,\dots,n\}}[ND]\langle P_i\rangle^{J_i}_{K_i}$$

[5] We assume $\sum_{i\in I}[p_i]\langle P_i\rangle^{J_i}_{\{\kappa\}} \equiv \underline{0}$ whenever $I=\emptyset$.

[6] This notation is employed just to improve the readability of axioms: whenever $P +^{ND} Q$ occurs in some equation, it could just be replaced by any choice of a particular parameter p.

as a shorthand notation for a non-deterministic choice among $\langle P_1\rangle^{J_1}_{K_1} \dots \langle P_n\rangle^{J_n}_{K_n}$, where K_i with $i \in \{1,\dots,n\}$ are pairwise disjoint.

Besides the two properties above that the axiom $\mathcal{A}_5$ expresses, this axiom is important in that it allows us to turn any sequential term (a term made up of $\underline{0}$, prefix and choice) into *sum normal form*.

Definition 10.7 *$P \in \mathcal{G}_{ext}$ is in sum normal form (snf) if and only if*

$$P \equiv \sum_{i\in I}[p_i]a_i.P_i \; +^{ND} \sum_{a\in AType} [ND] \sum_{i\in I^a}[p_i]a_*.P_i$$

where every P_i is itself in snf. [7] *We assume the sets of indexes I and I^a for each $a \in AType$ to be pairwise disjoint, i.e. $\forall a \in AType.\, I_a \cap I = \emptyset$ and $\forall a, b \in AType, a \neq b.\, I_a \cap I_b = \emptyset$.* ■

The axioms $\mathcal{R}_1 - \mathcal{R}_7$ characterize the behavior of the relabeling operator and allow us to turn a term $P[a \to b]^p$, where P is in snf, into snf. More precisely, we resort to the axioms $\mathcal{R}_5 - \mathcal{R}_7$ for distributing the relabeling operator among the alternative behaviors of P, and then we use axioms $\mathcal{R}_1 - \mathcal{R}_4$ for applying the relabeling operator to each $\underline{0}$ or prefix term. In particular, axiom $\mathcal{R}_6$ specifies how the nondeterministic choice among reactive actions a_* and b_* in P becomes a probabilistic choice (with parameter p) among reactive actions b_* in $P[a \to b]^p$. Note that, thanks to the fact that relabeling employs the same choice mechanism between reactive actions a_* and b_* as that of alternative composition, it is possible to express relabeling with parameter p in terms of choice with parameter p.

The axiom $\mathcal{P}$ characterizes the parallel composition operator in terms of the left merge $\lfloor^p_S$ and the synchronization merge $|^p_S$ operators (see e.g. [13]) and allows us to turn a term $P \,\|^p_S\, Q$, where P and Q are in snf, into snf. In particular, we resort to the left merge in order to express the local reactive and generative moves of the lefthand process, including the generative moves which synchronize with a reactive response of the righthand process (note that such moves are those depending on the parameter p of the parallel composition, see axioms $\mathcal{LM}_1 - \mathcal{LM}_9$), and we resort to the synchronization merge in order to express the synchronizing reactive moves (see axioms $\mathcal{SM}_1 - \mathcal{SM}_5$). Note that, thanks to fact that the parallel composition operator employs the same choice mechanism between the generative and reactive actions locally executed by P and Q as for the alternative composition, it is possible to express parallel composition with parameter p in terms of choice with parameter p. As far as generative actions are concerned, the operator $\langle P\rangle^J_K$ used in axioms $\mathcal{LM}_8$ and $\mathcal{LM}_9$ allows us to deal with action restriction (which in our calculus arises from parallel composition), without resorting to axioms with implications as in [13]. Finally, it is worth noting that when considering only reactive actions we obtain a smooth probabilistic extension of the classical axiomatization for the parallel composition by means of the left and synchronization merge.

Lemma 10.3 *For any $P \in \mathcal{G}_{fin}$ there exists $P' \in \mathcal{G}_{fin}$ in snf such that $\mathcal{A} \vdash P = P'$.*

[7] The use of $\sum$ and ND is correct since we have $a_i.P_i = \langle a_i.P_i\rangle^J_{\{\bullet\}}$ (axiom $\mathcal{A}_4$) and $a_*.P_i = \langle a_*.P_i\rangle^{\emptyset}_{\{a\}}$ (axiom $\mathcal{A}_5$).

Proof *The proof proceeds by structural induction over terms $P \in \mathcal{G}_{fin}$. The base step of the induction is just when P is a prefix operator or $\underline{0}$, hence it is already in snf.*

For the inductive step we have the following cases:

- *Supposed that terms P_1 and P_2 are in snf we show that $P \equiv P_1 +^p P_2$ can be turned into snf by using axiom $\mathcal{A}_5$.*

  Since P_1 is in snf we can, by using axioms $\mathcal{BK}_1 - \mathcal{BK}_3$, write it as: $\langle P_1' \rangle_K^J +^{ND} \langle P_1'' \rangle_\kappa^{J'}$ with $\kappa \notin K$.
  From $P = (\langle P_1' \rangle_K^J +^{ND} \langle P_1'' \rangle_\kappa^{J'}) +^p P_2$ we derive, by using $\mathcal{A}_5$: $P = \langle P_1' \rangle_K^J +^p (\langle P_1'' \rangle_\kappa^{J'} +^p P_2)$.
  Now, supposed that P_2 enables the bundle κ (the opposite case is trivial), since P_2 is in snf we can, by using axioms $\mathcal{BK}_1 - \mathcal{BK}_3$, write it as: $\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}$ with $\kappa \notin K'$.
  Therefore we have: $P = \langle P_1' \rangle_K^J +^p (\langle P_1'' \rangle_\kappa^{J'} +^p (\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}))$
  By applying axiom $\mathcal{A}_3$ we have: $P = \langle P_1' \rangle_K^J +^p (((\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}) +^{1-p} \langle P_1'' \rangle_\kappa^{J'})$
  By applying the axiom $\mathcal{A}_5$ we have: $P = \langle P_1' \rangle_K^J +^p (\langle P_2' \rangle_{K'}^{J''} +^{ND} (\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}))$
  By applying two times the axiom $\mathcal{A}_3$ we have: $P = (((\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}) +^{ND} \langle P_2' \rangle_{K'}^{J''}) +^{1-p} \langle P_1' \rangle_K^J$
  By applying the axiom $\mathcal{A}_5$ we have: $P = (\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}) +^{ND} (\langle P_2' \rangle_{K'}^{J''} +^{1-p} \langle P_1' \rangle_K^J)$
  By applying two times the axiom $\mathcal{A}_3$ we have: $P = (\langle P_1'' \rangle_\kappa^{J'} +^p \langle P_2'' \rangle_\kappa^{J'''}) +^{ND} (\langle P_1' \rangle_K^J +^p \langle P_2' \rangle_{K'}^{J''})$
  In this expression we have isolated the bundle κ from the other bundles. Now the same procedure is applied to $(\langle P_1' \rangle_K^J +^p \langle P_2' \rangle_{K'}^{J''})$ which no longer includes the bundle κ. When all the bundles have been isolated in this way we have obtained an expression in snf for term P.

- *Supposed that term P_1 is in snf it is easy to see that $P \equiv P_1[a \to b]^p$ can be turned into snf by using axioms $\mathcal{R}_1 - \mathcal{R}_7$. We first employ the axioms $\mathcal{R}_5 - \mathcal{R}_7$ in order to properly apply the relabeling operator to each alternative behavior of P. We apply the relabeling operator to each component of P representing a single bundle by using axiom $\mathcal{R}_5$. It is worth noting that axiom $\mathcal{R}_5$ cannot be used to distribute the relabeling operator among the two reactive bundles a and b. In order to do this, we apply axiom $\mathcal{R}_6$, as it recomputes the probability distribution of the reactive bundles a and b (by using the parameter p) according to the fact that the reactive actions a_* will be turned into the reactive actions b_*. Finally, for each component representing a single bundle, we use axiom $\mathcal{R}_7$ in order to distribute the relabeling operator among the possible behaviors of the bundle. Then we apply the relabeling operator to each $\underline{0}$ or prefix term by means of axioms $\mathcal{R}_1 - \mathcal{R}_4$.*

- *Supposed that terms P_1 and P_2 are in snf it is easy to see that $P \equiv P_1 \,\|_S^p\, P_2$ can be turned into snf by using axioms $\mathcal{P}$, $\mathcal{LM}_1 - \mathcal{LM}_9$ and $\mathcal{SM}_1 - \mathcal{SM}_5$.*

  In particular, we resort to the synchronization merge in order to manage the actions of the synchronizing reactive bundles belonging to the set S, and the left merge for the local reactive and generative actions, including the generative actions which synchronize with a reactive action. In the case of the synchronization merge, by applying the axioms $\mathcal{SM}_1$ and $\mathcal{SM}_5$ we obtain a term in sum form on which we can apply the axioms $\mathcal{SM}_2$, $\mathcal{SM}_3$ and $\mathcal{SM}_4$. Note that the axiom $\mathcal{SM}_1$ is the exact counterpart *of the axiom $\mathcal{A}_3$, and all the axioms we use are exactly as in the classical axiomatization except for the presence of the parameter p in the parallel operator. In the case of the left merge, by applying the axioms $\mathcal{LM}_8$ and $\mathcal{LM}_9$ we obtain a term in sum form on which we can apply the axioms $\mathcal{LM}_1 - \mathcal{LM}_7$. In particular, the axiom $\mathcal{LM}_8$ allows us to eliminate the transitions representing the subset of the generative bundle which is blocked because of the synchronization constraint expressed by the set S of the parallel operator. After this, we can apply the axiom $\mathcal{LM}_9$ in order to distribute the left merge operator among the components in sum form of the left side of the parallel composition. Axioms $\mathcal{LM}_1 - \mathcal{LM}_4$ allows us to eliminate the blocked components according to the synchronization set S of the parallel operator, and the axioms $\mathcal{LM}_5$ and $\mathcal{LM}_7$ determine all the synchronizing generative moves and the local moves of the lefthand process, respectively. Finally, axiom $\mathcal{LM}_6$ allows us to distribute the left merge operator among the components of the righthand process, in order to apply the axioms $\mathcal{LM}_4$ and $\mathcal{LM}_5$.* ■

Example 10.5 Let us consider the term

$$((((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0}) +^{ND} (b_*.\underline{0} +^s b_*.\underline{0})) +^{ND} a.\underline{0})[a \to b]^p$$

which is the relabeling of a process in snf.

We first apply axiom $\mathcal{R}_5$ that allows us to isolate the components representing the reactive bundles a and b, thus obtaining:

$$(((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0}) +^{ND} (b_*.\underline{0} +^s b_*\underline{0}))[a \to b]^p +^{ND} a.\underline{0}[a \to b]^p$$

Then, we distribute the relabeling operator between these two components by applying axiom $\mathcal{R}_6$, hence obtaining:

$$(((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0})[a \to b]^p +^p (b_*.\underline{0} +^s b_*.\underline{0})[a \to b]^p) +^{ND} a.\underline{0}[a \to b]^p$$

In this way the relabeling of the two reactive bundles of type a and b is expressed in terms of a probabilistic choice among two terms executing reactive actions of type b. Then, by repeatedly applying axiom $\mathcal{R}_7$ we have:

$(((a_*.\underline{0}[a \to b]^p +^q a_*.\underline{0}[a \to b]^p) +^r a_*.\underline{0}[a \to b]^p) +^p$
$(b_*.\underline{0}[a \to b]^p +^s b_*.\underline{0}[a \to b]^p)) +^{ND} a.\underline{0}[a \to b]^p$

Finally, by applying axioms $\mathcal{R}_1 - \mathcal{R}_4$, we change the actions of type a into actions of type b as follows:

$$(((b_*.\underline{0} +^q b_*.\underline{0}) +^r b_*.\underline{0}) +^p (b_*.\underline{0} +^s b_*.\underline{0})) +^{ND} b.\underline{0}$$

Such a term correctly represents the behavior of the relabeled term, where the choice among the actions b_* obtained by relabeling actions a_* and the preexisting actions b_* is guided by parameter p. ■

Theorem 10.6 *The axiom system $\mathcal{A}$ is complete for $\sim_{PB}$ over processes of $\mathcal{G}_{fin}$.*

Proof *The result is a trivial consequence of lemma 10.3. It is just sufficient to show that if two terms in snf are equivalent then they can be proved to be equal. Similarly as done in [19] this is proved in a rather standard way by inducing on the structure of the two terms in snf, in such a way that equivalent subterms are turned into equal subterms. In this procedure an important role is played by the idempotency axiom $\mathcal{A}_2$.*

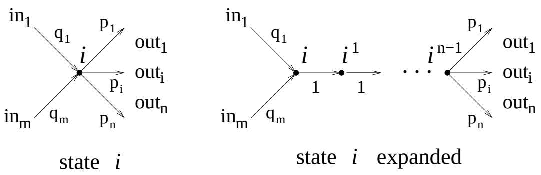
■

10.5 Processes with Exact Advancing Speeds

In this section we show that, when evaluating steady state based performance measures, we are able to deal with processes proceeding with different advancing speeds which are not probabilistic. In particular, while during an initial transient evolution considering the action frequency of processes as being exact instead of probabilistic may lead to different results when evaluating performance, we will show that in the case of non-blocking processes this does not happen when the system reaches a limiting steady behavior.

10.5.1 Exact Time Unit Scaling via Action Splitting

Let us assume that process P is a fully specified system such that the underlying DTMC $\mathbf{P} = [p_{i,j}]_{i,j \in S}$, where $S = \{1, \ldots, d\}$, possesses a (time averaged) steady state probability distribution (see Chapter 2). Let us suppose that we aim at executing P in parallel with another process which proceeds with a different action frequency. As already explained in Sect. 10.1.3, this could be done through a standard approach based on a synchronous parallel composition by adequately scaling the time unit on which the specification of the process is based, i.e. by splitting each action of P in a certain number n of subactions. Let us consider the effect of such an action split on the DTMC $\mathbf{P}$ underlying the process specification. Considered a state i of the DTMC $\mathbf{P}$, the split is applied to all the transitions outgoing from i. It is easy to see that, since in a DTMC each action takes one time unit to be executed, scaling the time unit by a factor $1/n$ corresponds to expand i in n states representing the passage of time while the system sojourns in i. More precisely, for each state i we add $n-1$ states $i^1, \ldots, i^{n-1}$ and we modify state transitions as follows. The outgoing transitions of i^{n-1} are all the transitions previously leaving i, the incoming transitions of i are preserved, and, defined $i^0 = i$, for each $k \in \{1, \ldots, n-1\}$ we have a transition going from i^{k-1} to i^k with probability 1.



For the sake of simplicity, let us consider the case $n = 2$, i.e. we split each action into two subactions. The new DTMC $\mathbf{P}' = [p'_{i,j}]_{i,j \in S'}$ where $S' = \{1, \ldots, 2 \cdot d\}$ can be represented as follows. Let us suppose that each state i^1 added when expanding a state i of $\mathbf{P}'$ is numbered in $\mathbf{P}'$ by $d+i$, hence in $\mathbf{P}'$ the first d states are those of $\mathbf{P}$ and the states from $d+1$ to $2\cdot d$ are those added in the expansion procedure. According to such a procedure $\mathbf{P}'$ turns out to be as shown below:

$$\begin{array}{c c}
\begin{array}{cc} 1 \;\ldots\; d & d{+}1 \ldots\; 2{\cdot}d \end{array} & \\
\left[\begin{array}{c|c} 0 & \mathrm{I}_d \\ \hline \mathrm{P} & 0 \end{array}\right] & \begin{array}{l} 1 \\ \vdots \\ d \\ d+1 \\ \vdots \\ 2\cdot d \end{array}
\end{array}$$

where I_d is the identity matrix of size d.

Now, let us evaluate the (time averaged) steady state behavior of the DTMC $\mathbf{P}'$. Called $\boldsymbol{\pi} = [\pi_i]_{i\in S}$ the steady state probability vector of $\mathbf{P}$, it turns out that in $\mathbf{P}'$ the stationary probability of a state $i \in S$ is equally partitioned between the states i and $d+i$ obtained by expanding i. Formally, the steady state probability vector $\boldsymbol{\pi}' = [\pi'_i]_{i\in S'}$ of $\mathbf{P}'$ is given by:

$$\pi'_i = \begin{cases} \pi_i/2 & \text{if } 1 \leq i \leq d \\ \pi_{i-d}/2 & \text{if } d+1 \leq i \leq 2\cdot d \end{cases}$$

or more intuitively by:

$$\boldsymbol{\pi}' = [\ \boldsymbol{\pi}/2 \ \mid \ \boldsymbol{\pi}/2\ ]$$

It is immediate to verify that the multiplication of such vector $\boldsymbol{\pi}'$ by the matrix $\mathbf{P}'$ yields the vector $\boldsymbol{\pi}'$ again.

In general for an arbitrary time unit scaling factor $1/n$ we represent the DTMC $\mathbf{P}' = [p'_{i,j}]_{i,j\in S'}$ where $S' = \{1,\ldots,n\cdot d\}$ as follows. Each state i^k added when expanding a state i of $\mathbf{P}'$ is numbered in $\mathbf{P}'$ by $k\cdot d+i$, hence in $\mathbf{P}'$ the first d states are those of $\mathbf{P}$, the states from $d+1$ to $2\cdot d$ represent the second step of the expansion of the states of $\mathbf{P}$ and so on. Therefore in general $\mathbf{P}'$ is given by:

$$\begin{array}{c c}
\begin{array}{cc} 1 \ldots\; d & d{+}1 \ldots\; n\cdot d \end{array} & \\
\left[\begin{array}{c|c} 0 & \mathrm{I}_{(n-1)\cdot d} \\ \hline \mathrm{P} & 0 \end{array}\right] & \begin{array}{l} 1 \\ \vdots \\ (n-1)\cdot d \\ (n-1)\cdot d+1 \\ \vdots \\ n\cdot d \end{array}
\end{array}$$

In $\mathbf{P}'$ the stationary probability of a state $i \in S$ is equally partitioned between the states $i, d+i, 2\cdot d + i, \ldots, (n-1)\cdot d + i$ obtained by expanding i. This because it is immediate to verify that the steady state probability vector $\boldsymbol{\pi}' = [\pi'_i]_{i \in S'}$ of $\mathbf{P}'$ turns out to be:

$$\boldsymbol{\pi}' = [\ \underbrace{\boldsymbol{\pi}'/n \ \mid \ \ldots \ \mid \ \boldsymbol{\pi}'/n}_{n \text{ times}}\ ]$$

Now, given an arbitrary performance measure m that we want to evaluate for the process P, m can be expressed by considering a reward structure $\mathbf{B} = [b_{i,j}]_{i,j \in S}$ corresponding to m (see Sect.2). If $\mathbf{B}' = [b'_{i,j}]_{i,j \in S}$ is the reward structure obtained from $\mathbf{B}$ by scaling the time unit by a factor $1/n$, then $\mathbf{B}'$ associates rewards to transitions between expanded states of S, i.e. $b'_{i,j}$ is the reward associated to the transition from the expanded state i to the expanded state j. More precisely the reward $b'_{i,j}$ is associated to the transition representing the execution of the final step of the transition from i to j, i.e. to the transition from state $i^{n-1} = (n-1)\cdot d + i$ to state j, while, for each $k \in 1, \ldots, n-1$ the transitions going from states i^{k-1} to states i^k are endowed with reward zero. Therefore we actually consider the reward structure $\mathbf{B}'' = [b''_{i,j}]_{i,j \in S'}$ depicted below:

$$\begin{array}{cl} \begin{array}{cc} 1 \ldots d & d{+}1 \ldots n\cdot d \end{array} & \\ \left[\begin{array}{c|c} 0 & 0 \\ \hline \mathrm{B}' & 0 \end{array}\right] & \begin{array}{l} 1 \\ \vdots \\ (n-1)\cdot d \\ (n-1)\cdot d+1 \\ \vdots \\ n\cdot d \end{array} \end{array}$$

The performance measure m is then given by:

$$m = \sum_{i \in S'} \sum_{j \in S'} \pi'_i \cdot b''_{i,j} \cdot p'_{i,j}$$

Since $b''_{i,j}$ is not null only if $i \in \{(n-1)\cdot d + 1, \ldots, n\cdot d\}$ and $j \in S$; and for all $i \in \{(n-1)\cdot d+1, \ldots, n\cdot d\}$ and $j \in S$ we have that $b''_{i,j} = b'_{i-(n-1)\cdot d, j}$, $p'_{i,j} = p_{i-(n-1)\cdot d, j}$ and $\pi'_i = \pi_{i-(n-1)\cdot d}/n$, it turns out:

$$m = \sum_{i \in S} \sum_{j \in S} \frac{\pi_i}{n} \cdot b'_{i,j} \cdot p_{i,j} = \frac{1}{n} \cdot \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \qquad (*)$$

Example 10.6 Let us consider a communication system composed of a processing unit that receives messages from an incoming channel and after some internal computation it sends out them to an outgoing channel. Suppose that P is a specification of such a system where the time unit is considered to be a second, i.e. an action takes one second to be executed, and $\mathbf{P} = [p_{i,j}]_{i,j \in S}$ is the underlying DTMC.

Suppose that we want to evaluate the throughput of the system in terms of the number of messages sent out per time unit. In order to do this we consider a reward structure $\mathbf{B_t} = [b_{i,j}]_{i,j \in S}$ determined as follows. We associate a reward equal to 1 to each action representing the sending of a message and a reward equal to 0 to each other action of the system specification. As explained in Sect. 2, called $\boldsymbol{\pi} = [\pi_i]_{i \in S}$ the vector of the steady state probabilities of $\mathbf{P}$, the value m_t of the throughput of the system is given by $\sum_{i \in S} \sum_{j \in S} \pi_i \cdot b_{i,j} \cdot p_{i,j}$. Now, let us suppose that we want to express the behavior of the system with respect to a different time unit, e.g. tenth of seconds instead of seconds. We may need to do this because we want to execute it in parallel with another process whose specification is made in tenth of seconds. As already explained, scaling the time unit by a factor $1/10$ can be made by splitting each action a of P in 10 subactions (9 *idle* actions followed by action a), thus obtaining a scaled DTMC $\mathbf{P'}$. The reward structure $\mathbf{B'_t}$ that we consider for the time scaled system is unchanged with respect to the reward structure $\mathbf{B_t}$ considered for the original system (we associate reward 0 to *idle* actions). This because the reward gained by actions is not related with their duration but it is just used to count the occurrences of the actions, i.e. rewards 0 and 1 associated to actions are not durations expressed in seconds, but just numbers. By calculating the throughput of the time scaled system with the formula $(*)$ we obtain the value $m_t/10$, i.e. one tenth of the throughput of the original system. This is an expected result because the throughput is a frequency which is expressed in number of actions executed per time unit and we changed the time unit from seconds to tenth of seconds.

Now let us suppose that we want to evaluate the utilization of the processing unit in terms of the percentage of time occupied by the system in performing internal computations for message processing. We consider a reward structure $\mathbf{B_u} = [b_{i,j}]_{i,j \in S}$ determined by associating a reward 1 to each action representing an internal computation of the processing unit and a reward equal to 0 to each other action of the system specification. Let us call m_u the value of the utilization of the processing unit derived from $\mathbf{B_u}$. Now, similarly as in the previous case, let us suppose that we want to scale the time unit by a factor $1/10$. Considered the DTMC $\mathbf{P'}$ obtained by expanding the states of $\mathbf{P}$, we have to evaluate the reward structure $\mathbf{B'_u}$ for the time scaled system. Differently from the case of the message throughput, in the definition of $\mathbf{B_u}$ the reward gained by an action is related to the duration of the activity it represents, i.e. rewards 0 and 1 are durations expressed in seconds. Hence, when we consider as the time unit tenth of seconds instead of seconds each reward must be multiplied by 10: it is like as all the subactions obtained by splitting an action with reward 1 of the original system would contribute to the evaluation of the utilization of the processing unit by all inheriting reward 1. Therefore we have $\mathbf{B'_u} = \mathbf{B_u} \cdot 10$. By calculating the utilization of the time scaled system with the formula $(*)$ we obtain the value m_u, i.e. the same utilization of the original system. This is an expected result because the percentage of utilization of the processing unit does not change if we scale the time unit for each activity of the system. ■

10.5.2 Exact Time Unit Scaling via Probabilistic Advancing Speed

Now we will show that the above analysis of steady state based performance measures gives the same results when the time scaling of P is probabilistic instead of being exact. By employing our probabilistic parallel composition operator we can approximate the scaling of a factor $1/n$ of the time unit used in a specification P by executing P with a probabilistic action frequency $1/n$. This is obtained by considering, e.g., the term $P \,\|^p_\emptyset\, \mathit{Idle}$, where $\mathit{Idle} \stackrel{\Delta}{=} \mathit{idle}.\mathit{Idle}$ is a process which repeatedly executes the action idle and $p = 1/n$.

Let us see what is the effect of executing process P with a probabilistic action frequency p on the DTMC $\mathbf{P}$ underlying the process specification. The set of states is unchanged, in particular the states of the GRTS underlying $P \,\|^p_\emptyset\, \mathit{Idle}$ are of the form $P' \,\|^p_\emptyset\, \mathit{Idle}$, where P' is a state of the semantics of P. The transitions leaving a state i of the DTMC $\mathbf{P}$ change as follows. A new transition corresponding to the execution of the action idle is added, which is executed with probability $1 - p$ and goes to i itself (a self-loop). The probability of every old transition leaving i is multiplied by p. Therefore the new DTMC $\mathbf{P'} = [p'_{i,j}]_{i,j \in S}$ where $S = \{1, \ldots, d\}$ turns out to be as follows:

$$p'_{i,j} = \begin{cases} p_{i,j} \cdot p + (1-p) & \text{if } i = j \\ p_{i,j} \cdot p & \text{if } i \neq j \end{cases}$$

or equivalently:

$$\mathbf{P'} = \mathbf{P} \cdot p + I_d \cdot (1-p)$$

where I_d is the identity matrix of size d.

Now, let us evaluate the (time averaged) steady state behavior of the DTMC $\mathbf{P'}$. It turns out that the stationary probability vector $\boldsymbol{\pi}$ of $\mathbf{P}$ represents the state probabilities at a steady behavior also for the new matrix $\mathbf{P'}$. This can be easily seen as follows:

$$\begin{aligned} \boldsymbol{\pi} \cdot \mathbf{P'} &= \boldsymbol{\pi} \cdot (\mathbf{P} \cdot p + I \cdot (1-p)) \\ &= (\boldsymbol{\pi} \cdot \mathbf{P}) \cdot p + (\boldsymbol{\pi} \cdot I) \cdot (1-p) \\ &= \boldsymbol{\pi} \cdot p + \boldsymbol{\pi} \cdot (1-p) \\ &= \boldsymbol{\pi} \end{aligned}$$

Now, suppose that we scale the time unit of process P with our probabilistic parallel operator and we want to evaluate the performance measure m expressed by the reward structure $\mathbf{B'} = [b'_{i,j}]_{i,j \in S}$ obtained from $\mathbf{B} = [b_{i,j}]_{i,j \in S}$ by scaling the time unit of a factor $1/n$. In the DTMC $\mathbf{P'}$ obtained from $P \,\|^p_\emptyset\, \mathit{Idle}$ the reward $b'_{i,j}$ is associated to the transitions representing moves of P that go from state i to state j, while we associate reward zero to the transitions derived from the moves idle of Idle. Since in $\mathbf{P'}$ the value $p'_{i,j}$ represents the total probability of going from state i to state j, in the case $i = j$ it accounts also for transitions going from i to itself due to the execution of the idle action. Therefore, according to the formula presented in Sect. 2, the total reward that we must associate with such an event is given by $b'_{i,i}$ multiplied by the probability that the passage from state i to itself is due to a self-loop representing a move of P (and not to an *idle* action). Such a probability is given by $p_{i,i} \cdot p$, i.e. the probability associated to the execution of a self-loop representing a move of P, divided by the total probability $p'_{i,i}$. Therefore for the DTMC $\mathbf{P}'$ we consider the reward structure $\mathbf{B}'' = [b''_{i,j}]_{i,j \in S'}$ defined as follows:

$$b''_{i,j} = \begin{cases} b'_{i,j} \cdot \frac{p_{i,j} \cdot p}{p'_{i,j}} & \text{if } i = j \\ b'_{i,j} & \text{if } i \neq j \end{cases}$$

The performance measure m is then given by:

$$m = \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b''_{i,j} \cdot p'_{i,j}$$

Since it holds that $b''_{i,j} \cdot p'_{i,j} = b'_{i,j} \cdot p_{i,j} \cdot p$ both for $i = j$ and for $i \neq j$, it turns out:

$$m = \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \cdot p = p \cdot \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \qquad (**)$$

which, recalling that we assumed $p = 1/n$, is the same formula that we obtained with the standard synchronous approach.

Example 10.7 Let us consider the communication system P of the previous example 10.6. We recall that in P the time unit is considered to be a second and $\mathbf{P} = [p_{i,j}]_{i,j \in S}$ is the DTMC underlying P. Now we show how to evaluate the two performance measures of example 10.6 by employing our approach based on probabilistic parallel composition. We scale the time unit of P to tenth of seconds by executing P with a probabilistic action frequency $1/10$. In particular we consider the term $P \,||^{1/10}_{\emptyset}\, \mathit{Idle}$ and the DTMC $\mathbf{P}'$ obtained from such a term. Let us consider the throughput of the system as defined by the reward structure $\mathbf{B_t}$ of example 10.6. As explained in example 10.6, since $\mathbf{B_t}$ is not based on action durations, the reward structure $\mathbf{B'_t}$ that we must apply to the time scaled system is the same as $\mathbf{B_t}$. If we instead consider the utilization of the processing unit as defined by the reward structure $\mathbf{B_u}$ of example 10.6 we must proceed as follows. As explained in example 10.6, since rewards are related to durations of actions, the reward structure $\mathbf{B'_u}$ that we must apply to the time scaled system is $\mathbf{B'_u} = \mathbf{B_u} \cdot 10$. Both in the case of the system throughput and in the case of the processing unit utilization, by using the same reward structures for the time scaled system $\mathbf{B'_t} = \mathbf{B_t}$ and $\mathbf{B'_u} = \mathbf{B_u} \cdot 10$ as those considered in Example 10.6, since we showed that formula $(**)$ coincides with formula $(*)$, we obtain the same performance measures as with the approach based on action splitting. ■

Finally it is worth noting that, with respect to the approach of Sect 10.5.1 based on action splitting, scaling the time unit via our probabilistic parallel operator gives the new possibility of using scaling factors p which are not of the form $p = 1/n$ with n natural number. In general $P \,||^{p}_{\emptyset}\, \mathit{Idle}$, when considered at the steady state, scales the time unit u_P used in the specification of P of a factor p, i.e. $u = u_P \cdot p$ is the time unit representing action duration in the behavior of $P \,||^{p}_{\emptyset}\, \mathit{Idle}$. This because $P \,||^{p}_{\emptyset}\, \mathit{Idle}$ executes a mean of p actions of P per time unit u, hence the mean action frequency f_P of P is given by $f_P = p/u$. Since an action of P takes a mean of $u_P = 1/f_P$ time to be executed we have $u = u_P \cdot p$. It is easy to see that also in the general case the performance measures obtained from $P \,||^p_\emptyset\, Idle$ at the steady state are the same obtained with an exact time unit scaling of factor p. In the case of performance measures representing a frequency (as e.g. the throughput of the system considered in examples 10.6 and 10.7), since the reward structure $\mathbf{B}'$ considered for the scaled system is the same as that ($\mathbf{B}$) of the original system, the formula ($**$) yields $m \cdot p$, where m is the value of the measure in the original system. The value $m \cdot p$ is correct for the scaled system because when we take time units into account, we have $m/u_P = (m \cdot p)/u$. In the case of performance measures representing percentage of time (as e.g. the utilization of the processing unit in examples 10.6 and 10.7), since the reward structure $\mathbf{B}'$ considered for the scaled system is determined from that ($\mathbf{B}$) of the original system by letting $\mathbf{B}' = \mathbf{B}/p$, according to formula ($**$) the value of the measure in the scaled system is the same as in the original system. This is correct because a percentage value does not refer to the time unit considered.

Summing up, we have the following theorem.

Theorem 10.7 *Let $P \in \mathcal{G}$ be a fully specified system such that the underlying DTMC possesses a time averaged steady state probability distribution. The steady state based performance measures of $P \,||^p_\emptyset\, Idle$ expressible by attaching rewards to the generative actions of P are exactly as those derived by executing the generative actions of P with an exact frequency p.* [8] ■

10.5.3 Exact Time Unit Scaling of System Components

Now we extend the result above by showing that even in the case of non-blocking processes which are part of a larger system (hence not performance closed processes), our probabilistic parallel operator determines, when the system is considered at steady state, an exact scaling of the time unit on which the specification of the processes in isolation are based. As a consequence we can express the parallel composition of processes specified with respect to different time units (hence proceeding at different exact action frequencies) through a common time unit by considering a different scaling factor for the time unit of each process. In Sect. 10.3.2 we showed that given two probabilistic action frequencies f_P and f_Q there exists a time unit u and a probability p such that in $P \,||^p_S\, Q$ the processes P and Q advance with mean action frequencies f_P and f_Q respectively. What we show now is that, if we suppose that both processes P and Q never block during the execution of $P \,||^p_S\, Q$ (in all the states of $P \,||^p_S\, Q$ at least one generative action of P and one generative action of Q are executable), at the steady state the representation of system behavior given by $P \,||^p_S\, Q$ is correct even when the action frequencies f_P and f_Q are considered as being exact instead of probabilistic, i.e. when each action of P (Q) is assumed to take exactly $u_P = 1/f_P$ ($u_Q = 1/f_Q$) time to be executed. In other words, if we assume that the specifications of processes P and Q in isolation are based on two different time units u_P and u_Q representing action duration, then $P \,||^p_S\, Q$ just expresses, by performing an exact scaling

[8] In this theorem and in the following Theorem 10.8 we consider performance measures of periodic DTMCs to be evaluated from their time averaged steady state probabilities (see Chapter 2).

of the time units u_P and u_Q to the new common time unit u, the concurrent execution of the two processes without affecting their behavior (apart from the consequences of action synchronization). In particular, as explained in Sect 10.5.2, u_P turns out to be scaled of a factor p ($u = u_P \cdot p$) and similarly u_Q turns out to be scaled of a factor $1 - p$.

In the following we show how the result of Theorem 10.7 can be smoothly extended to non-blocking system components.

In Sect 10.5.2 we showed that, supposed P a fully specified system such that the underlying DTMC possesses a (time averaged) steady state distribution, the term $P \,\|^p_\emptyset\, \mathit{Idle}$, where $\mathit{Idle} \triangleq \mathit{idle}.\mathit{Idle}$, at the steady state correctly represents the system behavior even when we execute exactly p actions of P per time unit u.

We generalize this result by first considering the case in which a process P with all the properties above is in parallel with an arbitrary process Q with the same properties (instead of the particular process Idle) but we have no synchronization between P and Q. It is trivial to see that, since $P \,\|^p_\emptyset\, \mathit{Idle}$ at steady state may be interpreted as executing exactly p actions of P per time unit u, then the same holds also for $P \,\|^p_\emptyset\, Q$. This because, since both processes P and Q enable at least one generative action in each system state (otherwise they could not be fully specified processes), given a state P' of process P, each system state $P' \,\|^p_\emptyset\, Q'$ (for any state Q' of Q) executes with probability p a transition of P' and with probability $1 - p$ a transition of Q'. Therefore the particular behavior of process Q in the states Q' reached through state changes while process P sojourns in state P' does not affect the behavior of P. Hence as far as the behavior of P is concerned considering the process $\mathit{Idle} \triangleq \mathit{idle}.\mathit{Idle}$ which has a single state is just like considering any arbitrary process Q with the properties above. More formally it is easy to see that in $P \,\|^p_\emptyset\, Q$ when we replace Q by the Idle process the sum of the steady state probabilities of being in states $P' \,\|^p_\emptyset\, Q'$ for any state Q' of Q becomes the steady state probability of being in the state $P' \,\|^p_\emptyset\, \mathit{Idle}$. Since in each state $P' \,\|^p_\emptyset\, Q'$ the process P behaves in the same way (as in $P' \,\|^p_\emptyset\, \mathit{Idle}$), at the steady state the behavior of process P in $P \,\|^p_\emptyset\, Q$ is the same as in $P \,\|^p_\emptyset\, \mathit{Idle}$. Note that by a symmetric argument we have also that $P \,\|^p_\emptyset\, Q$ at steady state executes exactly $1 - p$ actions of Q per time unit u.

Now we consider a more general case in which the process P may also synchronize with process Q. More precisely we consider a system $P \,\|^p_S\, Q$ such that both P and Q (which are no longer required to be fully specified processes) never get blocked during execution, i.e. in all the states of $P \,\|^p_S\, Q$ at least one generative action of P and one generative action of Q are executable. Since, whenever a process is a non interacting system component executed with mean action frequency p, at steady state it may be interpreted as executing exactly p generative actions per time unit u, it is easy to see that the same holds when the interacting process P is executed by $P \,\|^p_S\, Q$. This because, with respect to the execution of P with no interactions, the synchronization requirement with the reactive actions of process Q may change the set of generative actions of P executable in a system state, but, since P and Q never get blocked, not the frequency p at which generative actions of P are executed. More precisely, since in every system state both processes

P and Q may execute at least one generative action, the probability for P to be the process performing the next generative action is always p. Therefore independently on which are the generative actions of P actually enabled in system states, we have that the frequency of firings of P generative transitions is still exactly p per time unit u. By a symmetric argument we have also that the frequency of firings of Q generative transitions is exactly $1 - p$ per time unit u.

Summing up, we have the following theorem.

Theorem 10.8 *Supposed that both $P \in \mathcal{G}$ and $Q \in \mathcal{G}$ never block during the execution of $P \parallel_S^p Q$ (in all the states of $P \parallel_S^p Q$ at least one generative action of P and one generative action of Q are executable), the steady state based performance measures of $P \parallel_S^p Q$ expressible by attaching rewards to the generative actions of P and Q are exactly as those derived by executing the generative actions of P and Q with an exact frequency p and $1 - p$, respectively.* ■

10.6 A Case Study: Multi-Path Routing

In this section we present a case study showing how our approach provides a modular, compositional, and intuitive method for specifying concurrent systems in a scalable way. In particular, we consider a multi-path routing mechanism of the OSI network layer [148], and we model and analyze an internetworking node, whose arriving packets have several possible destinations with several possible ways to reach a destination. In the following, we refer to the node as the Interface Message Processor (IMP).

Before presenting the model, let us briefly recall some assumptions used in routing protocols. The routing algorithm decides, at the network layer, on which output link an arriving packet should be sent, depending on the destination of that packet. We abstract from the particular algorithm used to determine an optimal routing path between two nodes of a network, and we assume that the modeled IMP has a routing table including the route information with several possible choices for each destination. A weight is associated to each possible path and these weights are used as probabilities to decide where to send the present packet. Supporting multiple paths to the same destination, unlike single-path algorithms, permits traffic multiplexing over multiple lines. The advantages of multi-path are obvious: they can provide substantially better throughput and reliability.

10.6.1 Algebraic Specification of the Multi-Path Router

The overall model of our IMP (term *Multipath*) is shown in Table 10.5 in the particular case of two possible destinations (a and b) and two possible paths for each destination (a_1, a_2 for a, and b_1, b_2 for b). [9]

The algebraic specification is composed of several processes which are actually concurrent and are specified with respect to different time units. In particular, system *Multipath* consists of three concurrent components:

[9] In Table 10.5 we omit the parameters of the probabilistic operators if they are not meaningful for the system specification.

$$
\begin{array}{rcl}
\mathit{Multipath} & \triangleq & \mathit{Arrivals}\,\|^{p_1}_{S}\,(\mathit{Router}\,\|^{p_2}_{C}\,\mathit{Channels}) \\
S & = & \{\mathit{receivea},\ \mathit{receiveb}\} \\
C & = & \{\mathit{avail\_cha}_1,\ \mathit{avail\_cha}_2,\ \mathit{avail\_chb}_1,\ \mathit{avail\_chb}_2, \\
 & & \ \mathit{transma}_1,\ \mathit{transma}_2,\ \mathit{transmb}_1,\ \mathit{transmb}_2\} \\
\mathit{Arrivals} & \triangleq & \mathit{Arrivala}\,\|^{\frac{1}{2}}\,\mathit{Arrivalb} \\
\mathit{Channels} & \triangleq & (\mathit{Channel}[\varphi'_1]\ \|^{pa}\ \mathit{Channel}[\varphi'_2])\ \|^{v}\ (\mathit{Channel}[\varphi''_1]\ \|^{pb}\ \mathit{Channel}[\varphi''_2]) \\
\mathit{Router} & \triangleq & (\mathit{Queues}\,\|_{S_1 \cup I}\,\mathit{Switch})\,\|_{I}\,\mathit{Idle} \\
S_1 & = & \{\mathit{accepta},\ \mathit{acceptb}\} \qquad I\ =\ \{\mathit{idle}\} \\
\mathit{Queues} & \triangleq & \mathit{Queue}[\varphi']\,\|_{I}\,\mathit{Queue}[\varphi''] \\
\mathit{Switch} & \triangleq & (\mathit{Manager}[\varphi']\,\|_{S'_2}(\mathit{Routing}[\varphi'][\varphi'_1]\ \|^{qa}_{B'}\mathit{Routing}[\varphi'][\varphi'_2])\,)\quad \|^{p}_{I} \\
 & & (\mathit{Manager}[\varphi'']\,\|_{S''_2}(\mathit{Routing}[\varphi''][\varphi''_1]\ \|^{qb}_{B''}\mathit{Routing}[\varphi''][\varphi''_2])\,) \\
S'_2 & = & \{\mathit{senda},\ \mathit{busya}\} \qquad S''_2\ =\ \{\mathit{sendb},\ \mathit{busyb}\} \\
B' & = & \{\mathit{busya}\} \qquad\qquad\quad B''\ =\ \{\mathit{busyb}\} \\
\varphi' & = & \langle(\mathit{receive},\ \mathit{receivea}),\ (\mathit{accept},\ \mathit{accepta}),\ (\mathit{send},\ \mathit{senda}),\ (\mathit{busy},\ \mathit{busya})\rangle \\
\varphi'' & = & \langle(\mathit{receive},\ \mathit{receiveb}),\ (\mathit{accept},\ \mathit{acceptb}),\ (\mathit{send},\ \mathit{sendb}),\ (\mathit{busy},\ \mathit{busyb})\rangle \\
\varphi'_i & = & \langle(\mathit{transm},\ \mathit{transma}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_cha}_i)\rangle\ \ i \in \{1,2\} \\
\varphi''_i & = & \langle(\mathit{transm},\ \mathit{transmb}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_chb}_i)\rangle\ \ i \in \{1,2\} \\
\mathit{Arrivala} & \triangleq & \mathit{receivea}.\mathit{Arrivala} +^{ra} \mathit{wait}.\mathit{Arrivala} \\
\mathit{Arrivalb} & \triangleq & \mathit{receiveb}.\mathit{Arrivalb} +^{rb} \mathit{wait}.\mathit{Arrivalb} \\
\mathit{Queue} & \triangleq & \mathit{receive}_*.\mathit{Queue}' + \mathit{idle}_*.\mathit{Queue} \\
\mathit{Queue}' & \triangleq & \mathit{receive}_*.\mathit{accept}_*.\mathit{Queue}' + \mathit{accept}_*.\mathit{Queue} \\
\mathit{Manager} & \triangleq & \mathit{accept}.\mathit{Manager}' + \mathit{idle}_*.\mathit{Manager} \\
\mathit{Manager}' & \triangleq & \mathit{send}.\mathit{Manager} + \mathit{busy}.\mathit{Manager}' \\
\mathit{Routing} & \triangleq & \mathit{send}_*.\mathit{Routing}' + \mathit{avail\_ch}_*.\mathit{Routing} \\
\mathit{Routing}' & \triangleq & \mathit{transm}_*.\mathit{Routing} + \mathit{busy}_*.\mathit{Routing}' \\
\mathit{Idle} & \triangleq & \mathit{idle}.\mathit{Idle} \\
\mathit{Channel} & \triangleq & \mathit{avail\_ch}.\mathit{Channel} + \mathit{transm}.\mathit{Channel}
\end{array}
$$

Table 10.5: Multi-Path Routing Model

a term *Arrivals* modeling the incoming traffic, a term *Router* modeling the core of the IMP, and a term *Channels* modeling the outgoing channels. The structure of the three components is as follows.

- The term *Arrivals* is composed of two concurrent processes *Arrivala* and *Arrivalb* which model the incoming traffic directed to destinations a and b, respectively. The time unit representing action duration that we consider for both processes is one millisecond. The adoption of such a time unit makes it easy to represent a realistic workload for the IMP. In particular we assume that for each destination at most one packet per millisecond can arrive to the IMP, i.e. the maximum frequency of the incoming traffic is 2000 packets per second.

- The term *Router* represents a process whose time unit is half a microsecond, meaning that it can execute 2000000 actions per second. As we will see, the *Router* term, which is the core of the IMP, is a single processor machine managing the packets directed to the two destinations a and b via a probabilistic scheduler.

- Finally, the term *Channels* is composed of four concurrent processes modeling the four possible outgoing channels a_1, a_2, b_1, and b_2. Since we take packet transmission to be represented by the execution of a corresponding action, their time units are defined on the basis of their bandwidth. The time unit for the channel a_1 directed to a is 10 milliseconds, i.e. it can send out 100 packets per second, while the time unit for the channel a_2 directed to a is 2.5 milliseconds, i.e. it can send out 400 packets per second. The time unit for the channel b_1 directed to b is 5 milliseconds, i.e. it can send out 200 packets per second while the time unit for the channel b_2 directed to b is 4 milliseconds, i.e. it can send out 250 packets per second.

Note that it is possible to compose in parallel the above processes, which are specified with respect to different time units, because, as it can be easily verified, each of them never blocks during system execution (see Sect. 10.5.3). According to what we have explained in Sect. 10.5.3, in order to express the actual concurrent execution of such processes all the time units used in their specification are scaled to a common global time unit u. In particular u is evaluated by computing the inverse of the global action frequency of the composed system. Hence in our case study u is the inverse of $1000 + 1000 + 2000000 + 100 + 400 + 200 + 250$ actions per second, i.e. $u = 1/2002950$ seconds. The global time unit u adopted determines the action frequencies to be considered as parameters of the parallel operators used to describe the concurrent execution of system components in Table 10.5. In particular, parameter p_1 representing the advancing speed of term *Arrivals* in $\mathit{Arrivals} \,\|_S^{p_1} (\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels})$ is given by the ratio of the action frequency of term *Arrivals* over the global action frequency of system *Multipath*, i.e. if we express action frequency in seconds $p_1 = 2000/2002950 \approx 0.000999$ (see Sect 10.3.2). Similarly, parameter p_2 representing the advancing speed of term *Router* in $\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels}$ is given by the ratio of the action frequency of term *Router* over the global action frequency of term $\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels}$, i.e. $p_2 = 2000000/(2000000 + 950) \approx 0.999525$. As far as the specification of the *Arrivals* component is concerned, the parallel composition of the two concurrent processes *Arrivala* and *Arrivalb* has parameter $\frac{1}{2}$ because their action frequency is the same (1000 actions per second). As far as the specification of the *Channels* component is concerned, in $(Channel[\varphi_1'] \parallel^{pa} Channel[\varphi_2']) \parallel^{v} (Channel[\varphi_1''] \parallel^{pb} Channel[\varphi_2''])$ we take: pa to be $100/(100+400)$, i.e. $pa = 0.2$; pb to be $200/(200+250)$, i.e. $pb \approx 0.444444$; and v to be $500/(500+450)$, i.e. $v \approx 0.526316$. It is easy to see that the parameters adopted for the parallel operators give rise to the correct action frequencies. For instance process *Arrivala* executes $p_1 \cdot \frac{1}{2} \approx 0.000499$ actions per time unit u, i.e. 1000 actions per second, and process *Router* executes $(1-p_1) \cdot p_2 \approx 0.998527$ actions per time unit u, i.e. 2000000 actions per second.

Now let us describe in detail the behavior of each process of the system *Multipath*.

The process *Arrivala* (*Arrivalb*) models the incoming traffic through a Bernoulli distribution with parameter ra (rb). In particular, an arriving packet is represented by the action *receivea* (*receiveb*) which synchronizes with the corresponding reactive action in the queue for packets a (b) of term *Router*. In the case such a queue is full the action *receivea* (*receiveb*) is not enabled and the arriving packets are lost (the generative action *wait* is executed with probability 1).

The process *Router* is the core of the IMP and is composed of a term *Queues* collecting the arriving packets, a term *Switch* which delivers the packets to the outgoing channels and a term *Idle* modeling the phases of router inactivity. They are defined as follows.

- Term *Queues* consists of two *Queue* processes, one for each kind of packet, which behave reactively. In particular, they receive packets destined to a (b) through reactive actions of type *receivea* (*receiveb*) and pass them to the *Switch* term through reactive actions of type *accepta* (*acceptb*). For the sake of simplicity we assume both queues to be of size 2.

- Term *Switch* is a single-processor machine executing two different terms, each one managing packets with a certain destination (a or b), via a probabilistic scheduler with parameter p. In this way, by varying p, we can model an IMP that delivers packets with a particular destination more efficiently than packets with another destination, e.g. for commercial reasons. The term delivering packets to destination a (b) is composed of a *Manager* and two *Routing* terms, each one delivering packets to a particular channel a_1 or a_2 (b_1 or b_2). Term *Manager* accepts packets destined to a (b) from the dedicated queue through action *accepta* (*acceptb*) and afterwards either immediately passes them to one of the two *Routing* terms through action *senda* (*sendb*), or waits until at least one channel is available for transmission by performing action *busya* (*busyb*). This behavior is realized through a generative-reactive mechanism as follows. The two *Routing* terms behave reactively and each of them accepts packets through a reactive action of type *senda* (*sendb*) and transmits them through the corresponding channel via a reactive action of type *transma* (*transmb*). Whenever the generative action *senda* (*sendb*) is enabled by the *Manager* term, the *Routing* term accepting the packet is chosen according to the probability qa (qb) parameterizing the parallel composition of the two *Routing* terms. Note that a *Routing* term may be not available for accepting a packet because it is currently transmitting through action *transma* (*transmb*) a packet previously received. Therefore, whenever only one *Routing* process is available for accepting a packet coming from the *Manager* term, the packet is transmitted through the corresponding channel with probability 1. Whenever both *Routing* processes are busy the transmission of packets destined to a (b) is not possible and this is signalled to the *Manager* term through a multiway synchronization by enabling the reactive action of type *busya* (*busyb*) whose execution requires the synchronization of the two *Routing* processes.

- Term *Idle* executes an action *idle* (representing the fact that the IMP is idle) whenever term *Router* has nothing else to do. More precisely, action *idle* is executed through a multiway synchronization with all the other *Router* components if and only if the input queues (term *Queues*) are empty and the core of the IMP (term *Switch*) is not waiting for delivering a packet to the channel. In particular, term *Idle* prevents the term *Router* from blocking, thus allowing the advancing speed of terms *Arrivals*, *Router*, and *Channels* to be preserved and satisfying the condition needed for composing processes with different time units (see Sect. 10.5.3).

The four *Channel* processes model the two outgoing channels a_1 and a_2 directed to destination a and the two outgoing channels b_1 and b_2 directed to destination b. Each process *Channel* can either be transmitting a packet when the generative action *transm* is synchronized with the corresponding reactive action of term *Routing* managing that channel, or be available for transmission when the generative action *avail_ch* is synchronized with the corresponding reactive action of term *Routing*. For instance in the case of channel a_1 the generative actions $transma_1$ and $avail\_cha_1$ must synchronize with the reactive actions $transma_{1_*}$ and $avail\_cha_{1_*}$ of the *Routing* term managing channel a_1, respectively. In this way the generative actions of a *Channel* process are executed in mutual exclusion in the sense that in every system state one and only one of them is enabled. As a consequence term *Channels* never blocks.

Thanks to our approach which allows processes specified with respect to different time units to be modeled without splitting actions (see Sect. 4), we have that the transition system underlying the algebraic specification of Table 10.5 is composed of only 576 states and 4768 transitions. This is a crucial result, because if we want to deal with the same system by resorting to a classical and intuitive approach which scales the time unit by splitting each action, we have to cope with the serious problem of a greatly increased size of the state space. For instance, since the basic time unit for the router is half a microsecond whereas the basic time unit for the input channels is a millisecond, in order to compose in parallel terms *Arrivals* and *Router* we have to split the actions of term *Arrivals* in thousands of subactions thus causing a state space explosion.

Moreover, we point out that the generative-reactive behavior of the *Switch* process represents the core of this case study. In particular, process *Switch* generatively decides, according to probability p, which of the two *Manager* terms performs a send action (*senda* for the manager delivering packets to destination a or *sendb* for the other one), while it reactively decides, according to probability qa or qb (depending on the send action performed) which of the term *Routing* synchronizes with such an action. A calculus capable of

$$
\begin{array}{rcl}
\mathit{Multipath} & \triangleq & \mathit{Arrivals} \parallel_S^{p_1} (\mathit{Router} \parallel_C^{p_2} \mathit{Channels}) \\
C & = & \{\mathit{avail\_cha}_1,\ \mathit{avail\_cha}_2,\ \mathit{avail\_cha}_3,\ \mathit{avail\_cha}_4, \\
 & & \ \mathit{avail\_chb}_1,\ \mathit{avail\_chb}_2,\ \mathit{avail\_chb}_3,\ \mathit{avail\_chb}_4, \\
 & & \ \mathit{transma}_1,\ \mathit{transma}_2,\ \mathit{transma}_3,\ \mathit{transma}_4 \\
 & & \ \mathit{transmb}_1,\ \mathit{transmb}_2,\ \mathit{transmb}_3,\ \mathit{transmb}_4\} \\
 & & \\
\mathit{Channels} & \triangleq & ((\mathit{Channel}[\varphi'_1] \parallel^{pa'} \mathit{Channel}[\varphi'_2]) \parallel^{pa''} \\
 & & \ (\mathit{Channel}[\varphi'_3] \parallel^{pa'''} \mathit{Channel}[\varphi'_4])) \\
 & & \parallel^{v} \\
 & & ((\mathit{Channel}[\varphi''_1] \parallel^{pb'} \mathit{Channel}[\varphi''_2]) \parallel^{pb''} \\
 & & \ (\mathit{Channel}[\varphi''_3] \parallel^{pb'''} \mathit{Channel}[\varphi''_4])) \\
 & & \\
\mathit{Switch} & \triangleq & (\mathit{Manager}[\varphi'] \parallel_{S'_2} ((\mathit{Routing}[\varphi'][\varphi'_1] \parallel_{B'}^{qa'} \mathit{Routing}[\varphi'][\varphi'_2]) \parallel_{B'}^{qa''} \\
 & & \qquad\qquad\qquad (\mathit{Routing}[\varphi'][\varphi'_3] \parallel_{B'}^{qa'''} \mathit{Routing}[\varphi'][\varphi'_4])\,) \\
 & & \parallel_I^{p} \\
 & & (\mathit{Manager}[\varphi''] \parallel_{S''_2} ((\mathit{Routing}[\varphi''][\varphi''_1] \parallel_{B''}^{qb'} \mathit{Routing}[\varphi''][\varphi''_2]) \parallel_{B''}^{qb''} \\
 & & \qquad\qquad\qquad (\mathit{Routing}[\varphi''][\varphi''_3] \parallel_{B''}^{qb'''} \mathit{Routing}[\varphi''][\varphi''_4])\,) \\
\varphi'_i & = & \langle(\mathit{transm},\ \mathit{transma}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_cha}_i)\rangle \quad i \in \{1,2,3,4\} \\
\varphi''_i & = & \langle(\mathit{transm},\ \mathit{transmb}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_chb}_i)\rangle \quad i \in \{1,2,3,4\}
\end{array}
$$

Table 10.6: Multi-Path Routing Model with 4 Possible Channels per Destination

expressing generative-reactive choices is, therefore, very suitable (if not necessary) to model systems with such a behavior.

Finally it is worth noting that, thanks to the choice of putting probabilities in the operators (instead of, e.g., attaching them to actions) and to the expressive power of our generative-reactive approach, it was possible to specify the IMP in such a way that all the probabilistic mechanisms on which its behavior is based (and which are not related with the internal behavior of a process) depend on the parameters of parallel composition operators only. As a consequence scaling the system specification to a higher number of components does not make it necessary to change the internal behavior of processes. For instance in the router specification we can scale the system to a higher number of destinations or different channels for each destination by simply adding as many instances of *Switch* and *Channel* processes as we want and adjust appropriately the parameters of parallel composition operators. Here we consider the case of a router with 4 possible channels for each destination a and b instead of 2. In Table 10.6 we show the algebraic specification of the terms which are modified by such a scaling. Assumed that the bandwidth of the outgoing channels a_1, a_2, a_3, and a_4 is 50, 100, 150, and 200 packets per second, respectively, whereas the bandwidth of the outgoing channels b_1, b_2, b_3, and b_4 is 60, 120, 180, and 240 packets per second, respectively, the parameters of the parallel operators of Table 10.6 turn out to be as follows: $p1 \approx 0.000998$, $p2 \approx 0.998452$, $v \approx 0.454545$, $pa' = pb' \approx 0.333333$, $pa'' = pb'' = 0.3$, $pa''' = pb''' \approx 0.428571$. The transition system derived from such a specification is composed of 9216 states and 120640 transitions.

10.6.2 Performance Analysis

In order to derive performance measures from the multi-path router specification, we resorted to the software tool TwoTowers [19], that has been recently extended to support our generative-reactive approach (see Sect. 12.5). Such a tool also implements the algebraic reward based method described in Sect. 10.5 to specify and derive performance measures. The results of our performance analysis are shown in Fig. 10.5-10.8. In particular, we concentrated on two main metrics.

On the one hand we evaluate the throughput of the system at steady state, represented by occurrences of actions of type $transma_1$, $transma_2$, $transmb_1$, and $transmb_2$, by attaching a reward equal to 1 to the above actions and a reward equal to 0 to each other action. Since the throughput is a frequency expressed in terms of number of actions executed per time unit and the time unit is 1/2002950 seconds, we have to multiply the throughput resulting from the Markov chain analysis by 2002950 in order to obtain the results (expressed in seconds) shown in our tables.

On the other hand we evaluate the router idleness at steady state in terms of the percentage of time the IMP is inactive. The router is considered to be idle when no packet is currently inside the IMP, i.e. when it executes actions of type *idle*. Therefore we attach a reward equal to 1 to such actions and a reward equal to 0 to each other action. Since the time unit of the *Router* process (half a microsecond) is scaled by a factor $(1 - p_1) \cdot p_2 \approx 0.998527$ and the reward gained by actions is related to the duration of the corresponding activity expressed in half microseconds (see Sect. 4), due to the time unit change we must multiply each reward by 1/0.998527 before analyzing the Markov chain.

For each conducted analysis, we assumed that the incoming traffic for each kind of destination a and b follows the same Bernoulli distribution of parameter $r = ra = rb$. The pictures are built by showing how the performance measure under analysis changes when we vary r from 0.1 (sometimes 0.01) to 0.9. In this way we can observe the system behavior under various levels of workload ranging from 10% (or 1%) to 90%.

We start by evaluating the system throughput under different circumstances. We first consider the situation in which $p = \frac{1}{2}$, i.e. the packets destined to a and b are managed at the same speed by the *Switch* process, and parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. In particular, since channel a_1 can deliver 100 packets per second and channel a_2 can deliver 400 packets per second, we take $qa = 100/(100 + 400) = 1/5$ (the ratio of the bandwidth of channel a_1 over the overall bandwidth of the channels directed to destination a), so that packets are probabilistically

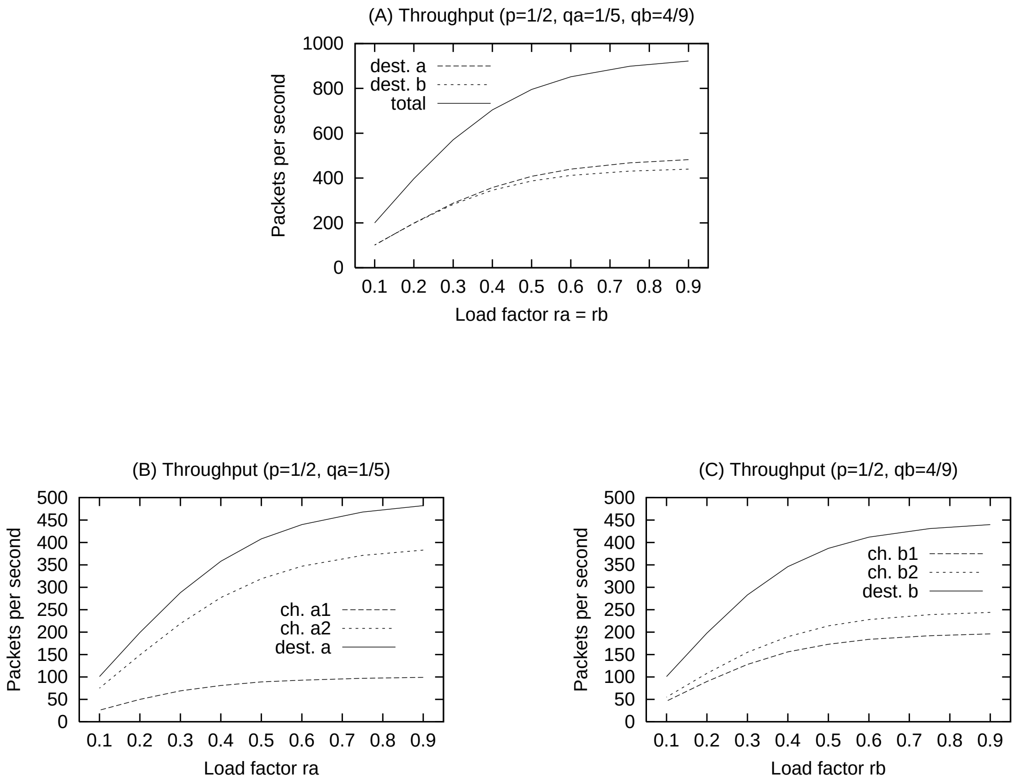


Figure 10.5: Throughput of the Multi-Path Router

distributed between channels a_1 and a_2 in the optimal way. Similarly, since channel b_1 can deliver 200 packets per second and channel b_2 can deliver 250 packets per second, we take $qb = 200/(200 + 250) = 4/9$. The obtained results are reported in Fig. 10.5. As we can see in the first table of Fig. 10.5 the curve representing the total system throughput is characterized by a high slope in correspondence of a low workload and a quite flat slope when the load factor increases over the 50%. This because, for packets with a given destination, the bandwidth associated with the outgoing channels directed to that destination is about one half of the maximum bandwidth of the incoming traffic. Simply put, when the parameter r of the Bernoulli distribution representing the incoming traffic reaches the 50%, the outgoing channel is almost fully occupied, hence a further increment of r gives rise to a very small increment of the outgoing throughput. Another expected result that we can observe in Fig. 10.5 is that the throughput of packets destined to a is slightly greater than the throughput of packets destined to b. This because the overall bandwidth of the outgoing channels directed to a is 500 packets per second, while it is 450 packets per second for the outgoing channels directed

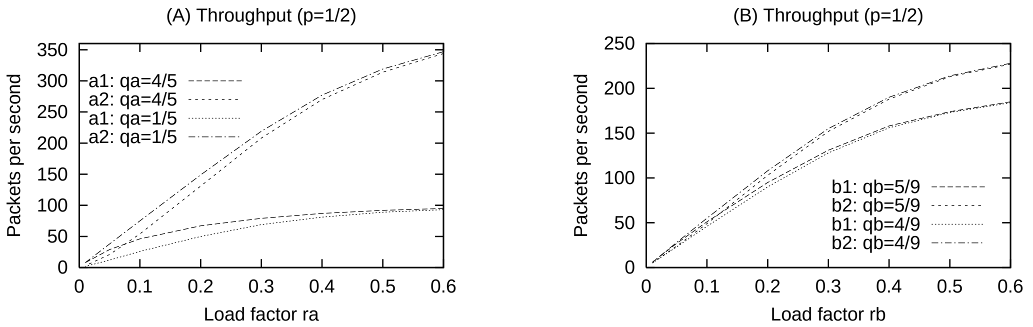


Figure 10.6: Throughput Obtained by Varying the Probabilistic Choice between the *Routing* Processes

to b. The other two tables of Fig. 10.5 report the throughput for each single channel a_1, a_2, b_1 and b_2. In the case of a_1 and a_2 the distance between the two curves is quite great, this because a_1 has just one fourth of the bandwidth of a_2. As expected such a difference is smaller in the case of b_1 and b_2, because their bandwidth is quite similar (200 and 250 packets per second, respectively).

Since in a realistic framework the value of parameters qa and qb are established by the multi-path routing algorithm governing the IMP according to the network conditions (e.g. estimated time for a packet to reach a destination via a particular path), we study the effect on the throughput of the router of adopting parameters qa and qb which do not reflect the bandwidth distribution over the outgoing channels. The results of such an analysis are reported in Fig. 10.6. The first table shows how the throughput of a_1 and a_2 varies when changing the value of qa from $\frac{1}{5}$ to $\frac{4}{5}$, i.e. by exchanging the value of qa and $1-qa$. For the sake of clarity we report the curves obtained for both $qa = \frac{1}{5}$ and $qa = \frac{4}{5}$. We can observe that the parameter qa does not play a significant role when the router is congested. This because under a heavy workload both routing processes are hardly occupied and in most cases at least one of them is busy. In such a situation the parameter qa is often not used, because when a routing process is busy an arriving packet destined to a is passed to the other routing process with probability 1. As a consequence the curves of the throughput converge to the same values when the load factor ra gets over 50%, i.e. when almost all the bandwidth of each channel is exploited. On the other hand, when the incoming workload is low, the parameter qa becomes important as it probabilistically decides which routing process will deliver the packet, hence increasing the throughput of a routing process with respect to the other one (see the quite evident difference among the curves when ra gets under the 30%). This because in the presence of a low workload both routing processes stay idle for most of time and an arriving packet a is directed to a particular channel depending on the choice made according to qa. The second table of Fig. 10.6 shows how the throughput of the two channels destined to b

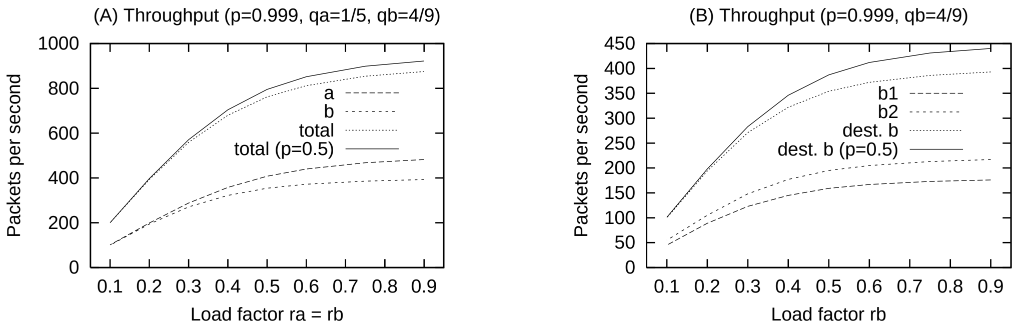


Figure 10.7: Throughput Obtained by Varying Parameter p.

varies when exchanging the value of qb and $1 - qb$. With respect to the case of the channels destined to a, in this case the difference between the old value of qb ($\frac{4}{9}$) and its new value ($\frac{5}{9}$) is smaller. This is reflected on the results presented in Fig. 10.6, where the curves for the old and the new value of qb are almost overlapped for each value of rb.

Now we show the role played by parameter p on the system throughput. In order to merely concentrate on the effects of varying parameter p we just consider the situation in which parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. Parameter p can be chosen in order to favor the internal computations of the IMP dedicated to packets destined to a (b) with respect to those dedicated to packets destined to b (a). To this aim, in Fig. 10.7 we report the throughput of the multi-path router in the case $p = 0.999$, hence when packets destined to b are managed by the IMP much more slowly than packets destined to a. As a consequence of the unfair behavior of the router, we have that the IMP delivers the packets destined to a at the usual speed (the curve for packets destined to a in the first table of Fig. 10.7 is the same as that in the first table of Fig. 10.5), but it delays the packets destined to b, hence compromising the throughput of such packets. Therefore with respect to the case $p = 0.5$ the overall system throughput decreases (for easy of comparison in the first table of Fig. 10.7 we also report the curve obtained in the case $p = 0.5$). The comparison with the case $p = 0.5$ is even more evident in the second table of Fig. 10.7, where we report the throughput of the outgoing channels directed to destination b.

As far as the idleness of the router is concerned, we simply consider the situation in which $p = \frac{1}{2}$ and parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. The curve presented in Fig. 10.8 shows the relation among the inactivity of the router and the load factor for the incoming traffic. As expected, the router is almost always idle if the workload is low, but the duration of its inactivity phases rapidly converges to zero for a load factor greater than 40%.

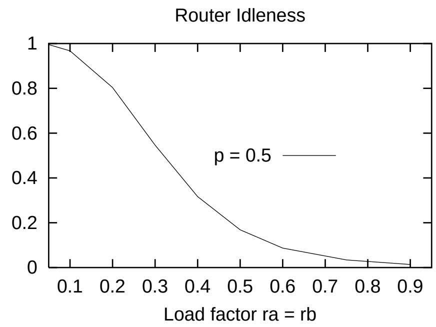


Figure 10.8: Idleness of the Multi-Path Router

Finally, we show in Fig. 10.9 the throughput and the idleness for the router with four possible channels for each destination presented in Table 10.6. Once again, the figures report the relation among the different metrics under the usual scenario where $p = \frac{1}{2}$ and parameters qa', qa'' and qa''', and qb', qb'' and qb''' reflect the bandwidth distribution over channels directed to destinations a and b, respectively (hence $qa' = qb' \approx 0.333333$, $qa'' = qb'' = 0.3$, $qa''' = qb''' \approx 0.428571$). As far as the throughput is concerned, since in the case of destination a we have preserved the overall outgoing bandwidth (500 packets per second) with respect to the router with two channels, the curve for destination a of the first and third table of Fig. 10.9 is the same as that we showed in Fig. 10.5. The total throughput for destination a is distributed among the four possible channels according to their bandwidth (see the third table of Fig. 10.9). On the other hand, the overall outgoing bandwidth for destination b is increased with respect to the router with two channels (600 packets per second vs. 450 packets per second). This is reflected on the curves of Fig. 10.9 which show that, in the case of the router with four channels for each destination, the total throughput for destination b is greater than the total throughput for destination a. The fourth table of Fig. 10.9 shows that, also in the case of b, the total throughput of packets destined to b is distributed among the four possible channels according to their bandwidth. As far as the router idleness is concerned, we can observe that the curve of the second table of Fig. 10.9 has the same shape as that seen in Fig. 10.8.

10.7 Conclusion

In this chapter we have presented a discrete time probabilistic calculus which integrates in a simple way a hybrid between the generative and reactive approaches [80] with probabilistic parallel composition [13]. Our generative-reactive model seems to be an adequate solution for modeling probabilistic behaviors of

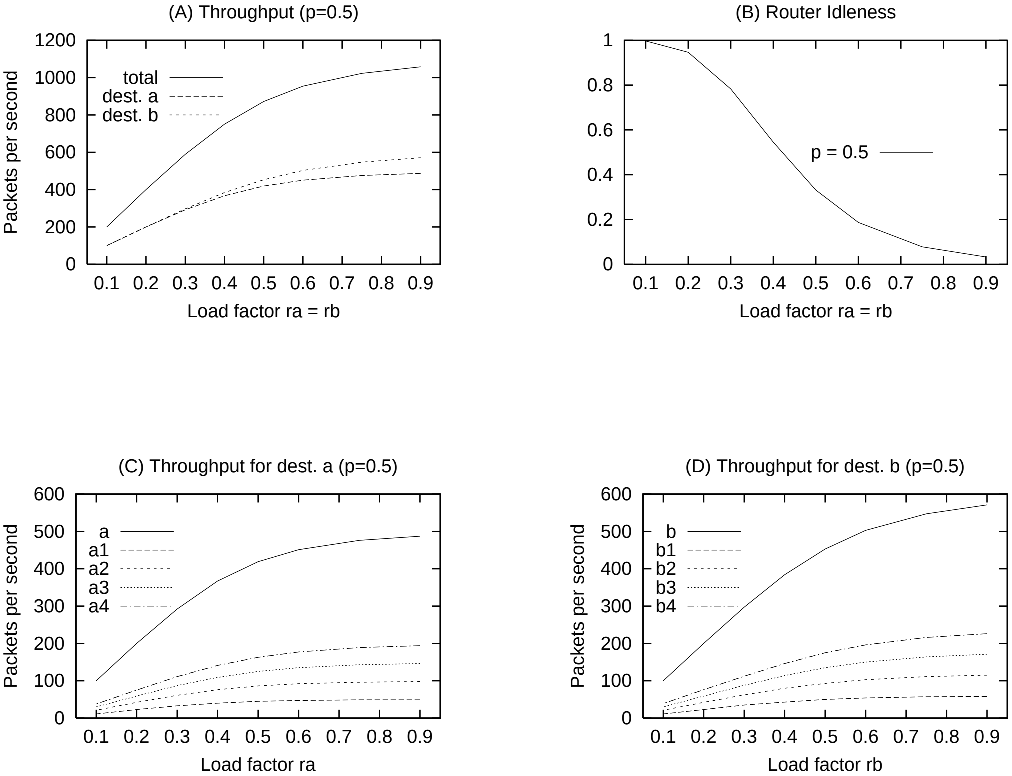


Figure 10.9: Throughput of the Multi-Path Router with 4 *Routing* per destination.

real systems, as it profitably joins the characteristics of both the generative and the reactive models, and mechanisms like probabilistic internal/external choice and multiway synchronization. Moreover, adopting a probabilistic asynchronous parallel operator in a discrete time setting gives our calculus a great modeling power. In particular, depending on how we calculate the time unit to be considered for the composed system, the parallel operator has a twofold interpretation and allows us to model:

1. multiple processes executed by a single processor machine with a probabilistic scheduling policy (in such a case, all parallel processes are assumed to be specified with respect to the same basic time unit as that representing action duration in the composed system),

2. concurrent processes which proceed with different advancing speeds, meaning that, differently from the classical synchronous approach, they may adopt different basic durations for their actions.

In the latter case, when considering the behavior of the system at a steady state, we showed that the representation of system behavior arising from our parallel operator is correct also when process advancing speeds are considered as being exact instead of probabilistic. This means that, e.g., if the action frequency of a process is $1/n$ then each action of the process takes exactly n time units to be executed. Thanks to this result we can model systems whose components are specified with respect to different basic action durations without incurring the problem of state space explosion which arises with an intuitive application (based on action splitting) of a standard synchronous approach. For instance, our technique made it possible to analyze the algebraic model of a multi-path router with 8 outgoing channels whose components are specified with respect to largely different time units: the Interface Message Processor of the router is, e.g., specified with actions (representing process computations) whose duration is half a microsecond, while the outgoing channel a_1 is specified with actions (representing the time time to send a packet through the channel) whose duration is 10 milliseconds. Specifying the router with a synchronous parallel operator would have required splitting the actions of the outgoing channels into thousands of subactions, hence making the model hard (if not impossible) to be analyzed. Moreover, the idea of expressing the probabilistic advancing speed of processes by means of a parameter of the parallel operator (as in [13]) instead of, e.g., using weights attached to actions, has turned out to be adequate from the modeling viewpoint because:

1. the modeler can first specify the behavior of processes in isolation and then establish, independently on how they are specified, their relative advancing speed when composing them in parallel,

2. it leads to specifications which are easily scalable to any number of components (e.g. outgoing channels in our case study) without changing the internal behavior of processes.

As a further advantage of adopting a probabilistic parallel operator, we have that in the case of fully specified systems (where all the reactive actions get synchronized) the resulting generative-reactive transition system is fully probabilistic and can be trivially turned into a discrete time Markov chain. Therefore deriving performance measures from specifications of complete systems can be done by applying standard techniques. For instance, we have shown how to employ an algebraic reward based method for expressing performance measures of systems and we have applied it to evaluate the throughput and idleness of our multi-path router model.

Finally, we have obtained a sound and complete axiomatization for (a simple variant of) strong probabilistic bisimulation over nonrecursive processes of our calculus which:

1. by employing a simple auxiliary operator expresses restriction of generative actions in a clean way without resorting to axioms with implications as in [13],

2. in the case of reactive actions is a smooth extension of the axiom system for a nonprobabilistic standard process algebra which expresses parallel composition through the left merge and synchronization merge operators.

Note that, differently from what we did in the second part of the thesis, here we did not consider a weak version of our probabilistic bisimulation (as, e.g., that of [16]) because, since in the context of discrete time each transition takes one time unit to be executed, we cannot merge several τ transitions into a single τ transition while preserving the temporal behavior of the system.

Part IV

Gaining Expressivity

Chapter 11

Generative-Reactive Extended Markovian Process Algebra

In this chapter we start addressing the problem of developing a specification language which integrates the two approaches presented in the second and third parts of the thesis (one based on continuous time and the other one based on discrete time) and which is endowed with additional expressive features, such as multi-level priorities and rewards. This is important since the modeling and analysis experience (e.g. with the process algebra EMPA) has shown the necessity of expressing priority, probabilistic internal/external choice, and discrete/continuous time in order to be able to faithfully model the behavior of real systems and the effectiveness of rewards as an instrument for evaluating the properties of interest.

An important open problem in this scenario is how to obtain semantic compositionality in the presence of all these features, to allow for an efficient analysis. Starting from a basic Markovian process algebra, i.e. an extension of a standard process algebra with exponentially distributed durations, our objective is to show how to add the expressive features above while preserving compositionality. Theoretically speaking, we argue that, when abandoning the classical nondeterministic setting by considering the features above, a natural solution is to break the symmetry of the roles of the processes participating in a synchronization through an extension of the generative-reactive approach introduced in the third part of the thesis. We accomplish this by distinguishing between master actions – the choice among which is carried out generatively according to their priorities/probabilities or exponentially distributed durations – and slave actions – the choice among which is carried out reactively according to their priorities/probabilities – and by imposing that a master action can synchronize with slave actions only. We also show that such an asymmetric cooperation mechanism is natural and easy to understand by means of the novel "cooperation structure model".

In this chapter the investigation above will lead to the definition of $EMPA_{\mathrm{gr}}$, a Markovian process algebra capable of representing both discrete and continuous time systems, where we can express: activities with exponentially distributed durations (which are not considered when doing discrete time specifications), activities with a discrete duration (which are assumed to take time zero in continuous time), probabilities, multi-level priorities and the generative master-reactive slaves synchronization mechanism. In Chapter 12 $EMPA_{\rm gr}$ will be extended with rewards, while in Chapter 13 we will introduce general distributions.

The main results that we present in this chapter are the following ones. We prove that the synchronization mechanism in $\rm EMPA_{gr}$ is correct w.r.t. the cooperation structure model, we show that the Markovian bisimulation equivalence is a congruence w.r.t. all the operators of $\rm EMPA_{gr}$ as well as recursion, and we present a sound and complete axiomatization of the Markovian bisimulation equivalence for nonrecursive process terms. As far as the Markovian bisimulation equivalence is concerned, we introduce a new notion of Markovian bisimulation up to Markovian bisimulation equivalence, which improves the previous definitions given in the literature, and a new proof technique for showing congruence w.r.t. recursion in Markovian process algebras, which repairs some inaccuracies in the proofs previously proposed in the literature.

11.1 Introduction

The experience of the past twenty years with process algebras (see e.g. [19, 8, 9, 28]) has shown that several expressive features are necessary to be able to model real world systems. Moreover, to be hopefully able to analyze such systems, the expressive features must be introduced in such a way that semantic compositionality is achieved, i.e. in such a way that it is possible to define a congruence that can be exploited to compositionally minimize the state space before applying the analysis techniques.

In this chapter we start with the simple process algebra inspired by [122, 109] presented in Chapter 2 and we show step by step how its expressive power can be greatly enhanced while preserving semantic compositionality. At the beginning, we introduce in the process algebra the concept of time through the capability of expressing exponentially timed actions and passive actions (whose duration becomes specified only upon synchronization with exponentially timed actions of the same type) and we show that the resulting Markovian process algebra can be given semantics in the usual interleaving style thanks to the memoryless property of exponential distributions (Sect. 11.2).

We then argue that we need a way of representing actions that are irrelevant from the timing viewpoint or just control the system behavior. We thus extend our Markovian process algebra with immediate actions, i.e. actions having duration zero (Sect. 11.3).

We subsequently observe that it often happens in practice to encounter systems where different competing actions are scheduled according to some priority assignment and/or with a certain frequency. Therefore, we extend our Markovian process algebra by attaching priorities and weights to immediate actions (Sect. 11.4 and 11.5).

Afterwards, we focus on the problem of endowing our process algebra with a general form of multiway synchronization and we argue that, when abandoning the classical nondeterministic setting by considering the expressive features above, breaking the symmetry of the roles of the processes participating in a synchronization is a natural solution to the problem of achieving semantic compositionality. We then present in some details the synchronization mechanism of our process algebra which is based on the distinction between master actions (exponentially timed and prioritized-weighted immediate actions) and slave actions (passive actions enriched with priorities and weights enforced only among passive actions of the same type) and on the constraint that a master action can synchronize with slave actions only. Following the terminology of [80], the choice among master actions is carried out generatively according to their priorities/weights or exponentially distributed durations, while the choice among slave actions of the same type is carried out reactively according to their priorities/weights (Sect. 11.6).

After introducing all the ingredients for our extended Markovian process algebra with generative-reactive synchronizations, called $\mathrm{EMPA}_{\mathrm{gr}}$, we formalize its syntax and we define its operational semantics as a mapping from terms to a new class of labeled transition systems, called master-slaves transition systems (Sect. 11.7).

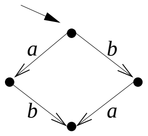
We then present the novel cooperation structure model, which formally describes the structure of the master-slaves synchronizations that can be performed in a state of a system specified by an $\mathrm{EMPA}_{\mathrm{gr}}$ term. The importance of such a model is that it shows that the $\mathrm{EMPA}_{\mathrm{gr}}$ synchronization mechanism, as defined by its operational semantics, is natural and easy to understand. In particular, the structure of the master-slaves synchronizations performable in a system state can be simply described by binary trees with two kinds of nodes: *cooperation nodes*, which represent synchronizations, and *choice nodes*, which represent choices performed according to a mixed prioritized/probabilistic mechanism, where the priority and the probability associated with a certain subtree are determined by the actions labeling its leaves. We provide a denotational semantics that maps $\mathrm{EMPA}_{\mathrm{gr}}$ terms into cooperation structures and we formally prove the $\mathrm{EMPA}_{\mathrm{gr}}$ operational semantics to be correct by showing that the structure of the synchronizations deriving from an $\mathrm{EMPA}_{\mathrm{gr}}$ term is the same as that described by the corresponding cooperation structure (Sect. 11.8).

We subsequently define a notion of equivalence in the bisimulation style – called Markovian bisimulation equivalence and denoted $\sim_{\mathrm{MB}}$ – which equates $\mathrm{EMPA}_{\mathrm{gr}}$ terms possessing the same functional, probabilistic, prioritized and exponentially timed behavior. We accompany $\sim_{\mathrm{MB}}$ with a notion of Markovian bisimulation up to $\sim_{\mathrm{MB}}$ that improves those previously given for other Markovian process algebras, in that it fixes some inaccuracies present in the definition of [107, 27] and eliminates the necessity of including symmetric pairs like in [99]. We then show that $\sim_{\mathrm{MB}}$ is a congruence, thus providing support for efficient compositional analysis and improving the result of [19] where the congruence is not achieved for the whole language. In particular, we show that $\sim_{\mathrm{MB}}$ is a congruence w.r.t. recursive definitions by introducing a new proof technique, which leads to a smooth extension of the technique based on bisimulation up to bisimulation equivalence [122] and solves some problems in the proofs of the same result previously proposed for Markovian process algebras [107, 27]. Finally, we give a sound and complete axiomatization of $\sim_{\mathrm{MB}}$ for nonrecursive process terms (Sect. 11.9).

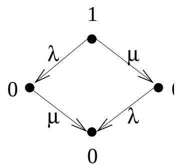
The chapter concludes with a discussion of related work (Sect. 11.10).

11.2 A Basic Markovian Process Algebra

Process algebras are compositional languages for the high level specification of concurrent systems. Hereafter we will consider the simple process algebra inspired by [122, 109] introduced in Chapter 2. We recall from Chapter 2 the semantics of process algebra terms is given by means of *rooted labeled transition systems* (LTSs for short) in which states correspond to process terms and transitions are labeled with actions. Such LTSs are defined by following the *interleaving approach*, i.e. parallel executions are serialized by representing each of them through the set of all the possible sequential executions obtained by interleaving the actions executed by the parallel components. A consequence of the interleaving approach is that the two different systems $a.\underline{0} \,\|_{\emptyset}\, b.\underline{0}$ and $a.b.\underline{0} + b.a.\underline{0}$ are assigned isomorphic LTSs:



In the field of performance evaluation, a model largely used to compute efficiency measures is that of *Markov chains* [147] (MCs for short). As explained in Chapter 2, MCs in their continuous time variant are essentially LTSs where the initial state is replaced by a probability mass function, which expresses for each state the probability that it is the initial one, and the transitions are labeled by positive real numbers, which are the rates of the exponentially distributed random variables describing transition durations. Since in Continuous Time Markov Chains (CTMCs) choice are solved according to a *race policy* (see Sect. 4.1) an "interleaving" representation naturally arises also to the case in which two activities whose durations are exponentially distributed with rate λ and μ respectively, are executed in parallel:

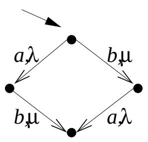


We recall from Sect. 4.1 that the CTMC above correctly depicts the aforementioned scenario thanks to the *memoryless property* of the exponential distribution, because an action can be regarded as being initiated in the same state in which it terminates its execution. Moreover no transition is possible from the initial state to the absorbing one as the probability that the two actions terminate simultaneously is zero.

When merged together, the specification languages and the stochastic models above give rise to *Markovian process algebras* [107, 19, 99]. With respect to the Markovian process algebra we introduced in Chapter 4, where standard actions and exponentially timed delays are dealt with in a separate fashion, here we exploit the similarities in the representation of parallel composition in both process algebras and CTMCs to really merge them into a language with timed actions (as done in [107, 19]).

From the syntactical viewpoint, we describe each action as a pair $<a, \tilde{\lambda}>$, where a is the type of the action and $\tilde{\lambda}$ is the rate of the action. If $\tilde{\lambda} \in \mathbb{R}_+$, then the action is called *exponentially timed* as its duration is assumed to be exponentially distributed with rate $\tilde{\lambda}$. If instead $\tilde{\lambda} = *$, then the action is called *passive* and its duration is unspecified. As for the binary operators, the alternative composition operator is governed by the race policy as long as a choice among exponentially timed actions is concerned. In the case of the parallel composition operator, instead, a synchronization between $<a, \tilde{\lambda}>$ and $<a, \tilde{\mu}>$, with a in the synchronization set, is possible only if at least one of $\tilde{\lambda}$ and $\tilde{\mu}$ is $*$, and the resulting rate is given by the other rate. This entails that, in a multiway synchronization, at most one exponentially timed action can be involved (which plays the role of the master, hence determines the rate of the synchronization), while all the other actions (which play the role of slaves) must be passive. We shall return on this master-slaves synchronization mechanism in Sect. 11.6.

From the semantic viewpoint, we observe that the interleaving approach of process algebras and the memoryless property of exponential distributions fit together well, so that the interleaving approach can be followed also in the case of Markovian process algebras. As an example, the two different systems $<a, \lambda>.\underline{0} \,\|_{\emptyset}\, <b, \mu>.\underline{0}$ and $<a, \lambda>.<b, \mu>.\underline{0} + <b, \mu>.<a, \lambda>.\underline{0}$ are assigned isomorphic LTSs:



The LTS above is called the *integrated interleaving semantics* of the process terms at hand, because each transition is labeled with both the type and the rate of the corresponding action. From such an integrated model two projected semantic models can be derived by discarding action rates or action types, respectively. The former is called the *functional semantics* as its transitions are not decorated with performance related information, thus representing only the functional behavior of the system. The latter, instead, is called the *Markovian semantics* as it expresses the CTMC governing the stochastic behavior of the system.

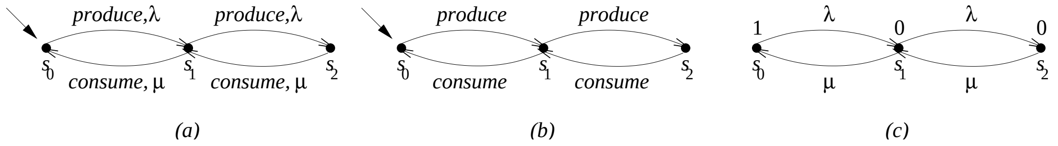


Figure 11.1: Interleaving semantic models of $PCSystem_2$

Example 11.1 A producer/consumer system is a system composed of a producer, a buffer, and a consumer. The producer repeatedly produces new items at a certain speed and puts them into the buffer until the buffer is empty, while the consumer withdraws items from the buffer at a certain rate unless the buffer is empty. Assuming for simplicity a buffer of capacity two, the architecture of this system can be modeled with our Markovian process algebra as follows:

$$PCSystem_2 \triangleq Producer \,\|_{\{produce\}}\, Buffer_0 \,\|_{\{consume\}}\, Consumer$$

Assuming that the item production process and the item consumption process are Markovian with rate λ

and μ, respectively, the producer and the consumer can be modeled as follows:

$$\begin{aligned} Producer &\stackrel{\Delta}{=} \; <produce, \lambda>.Producer \\ Consumer &\stackrel{\Delta}{=} \; <consume, \mu>.Consumer \end{aligned}$$

The buffer, instead, is at any time ready to accept new incoming items (if not full) and to deliver previously produced items (if not empty):

$$\begin{aligned} Buffer_0 &\stackrel{\Delta}{=} \; <produce, *>.Buffer_1 \\ Buffer_1 &\stackrel{\Delta}{=} \; <produce, *>.Buffer_2 + \\ & \quad\;\; <consume, *>.Buffer_0 \\ Buffer_2 &\stackrel{\Delta}{=} \; <consume, *>.Buffer_1 \end{aligned}$$

Note that only passive actions occur in *Buffer*, to reflect the fact that the interactions established by the two synchronization sets $\{produce\}$ and $\{consume\}$ are guided by the exponentially timed actions of the producer and the consumer.

In Fig. 11.1(a) we show the integrated interleaving semantics of $PCSystem_2$. The initial state s_0 corresponds to $PCSystem_2$, state s_1 to $Producer \,\|_{\{produce\}}\, Buffer_1 \,\|_{\{consume\}}\, Consumer$, state s_2 to $Producer \,\|_{\{produce\}}\, Buffer_2 \,\|_{\{consume\}}\, Consumer$. As reported in Fig. 11.1(b) and (c), from such a LTS a functional LTS and a CTMC can be derived by dropping action rates or action types, respectively. ■

11.3 Immediate Actions

The first extension of our Markovian process algebra is concerned with the introduction of *immediate actions*. They are executed in zero time, hence their rate is denoted by ∞. Introducing immediate actions is necessary to model system activities which are several orders of magnitude faster than those relevant from the performance viewpoint, as well as system activities that control the system behavior.

Since immediate actions have zero duration, they take precedence over exponentially timed ones. To make this clear, let us consider a system E that initially can perform either an exponentially timed action a or an immediate action b: $<a, \lambda>.E_1 + <b, \infty>.E_2$. The integrated interleaving semantic model of E has the two following initial transitions:

$$\begin{tikzcd} & E \arrow[dl, "{a, \lambda}"'] \arrow[dr, "{b, \infty}"] & \\ E_1 & & E_2 \end{tikzcd}$$

If E represents a closed system, i.e. a complete specification which is not part of a larger one and from which we want to derive a MC, then only the transition labeled with action $<b, \infty>$ can be actually performed and is executed immediately in zero time. If instead E represents an open system, i.e. a system that can interact with its environment, then the execution of action $<b, \infty>$ may be disabled by the environment. For instance, $E \,\|_{\{b\}}\, \underline{0}$ has a single initial transition labeled with action $<a, \lambda>$, as $\underline{0}$ is not willing to perform any b action hence no synchronization on b can occur.

Similarly to exponentially timed actions, in a synchronization at most one immediate action can be involved while all the other actions must be passive. If an immediate action is involved, then the rate of the resulting action is immediate, otherwise it is passive.

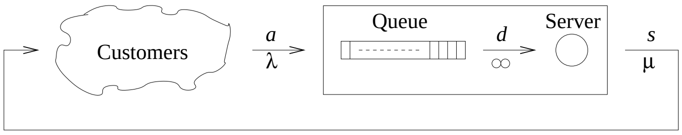


Figure 11.2: Structure of a queuing system $M/M/1/q$

Example 11.2 From now on we shall exemplify each feature added to our language by means of *queuing systems* [115] (QSs for short), which are abstract models largely used for evaluating the performance of computer and communication systems through the computation of measures such as system throughput, resource utilization, and user response time (see Chapter 2).

Let us consider a QS $M/M/1/q$ with arrival rate λ and service rate μ, whose structure is depicted in Fig. 11.2 where a stands for arrive, d for deliver, and s for serve. To faithfully represent the fact that the buffer has capacity $q-1$, an immediate action is necessary to model the fact that the customer at the beginning of the queue is passed to the server as soon as it becomes free. Without such an immediate action, the capacity of the service center would be decreased by one.

The QS at hand can be modeled as follows:

$$\begin{array}{rcl} QS_{M/M/1/q} & \triangleq & Arrivals \parallel_{\{a\}} (Queue_0 \parallel_{\{d\}} Server) \\ Arrivals & \triangleq & <a, \lambda>.Arrivals \\ Queue_0 & \triangleq & <a, *>.Queue_1 \\ Queue_h & \triangleq & <a, *>.Queue_{h+1} + \\ & & <d, *>.Queue_{h-1}, \ \ 0 < h < q-1 \\ Queue_{q-1} & \triangleq & <d, *>.Queue_{q-2} \\ Server & \triangleq & <d, \infty>.<s, \mu>.Server \end{array}$$

where we note that all the actions describing the behavior of the queue are passive. We now conclude by showing how to derive the CTMC underlying $QS_{M/M/1/q}$. In Fig. 11.3(a) we depict the model of $QS_{M/M/1/q}$ obtained from its integrated semantic model by pruning the exponentially timed transitions which have alternative immediate transitions. As explained at the beginning of this section, this is correct because $QS_{M/M/1/q}$ represents a closed system. In Fig. 11.3(b) we depict the Markovian semantic model of $QS_{M/M/1/q}$, which is obtained from the model of Fig. 11.3(a) by discarding action types and eliminating the states having outgoing immediate transitions. The reason is that the sojourn time in those states is zero, so they are irrelevant from the performance viewpoint. ■

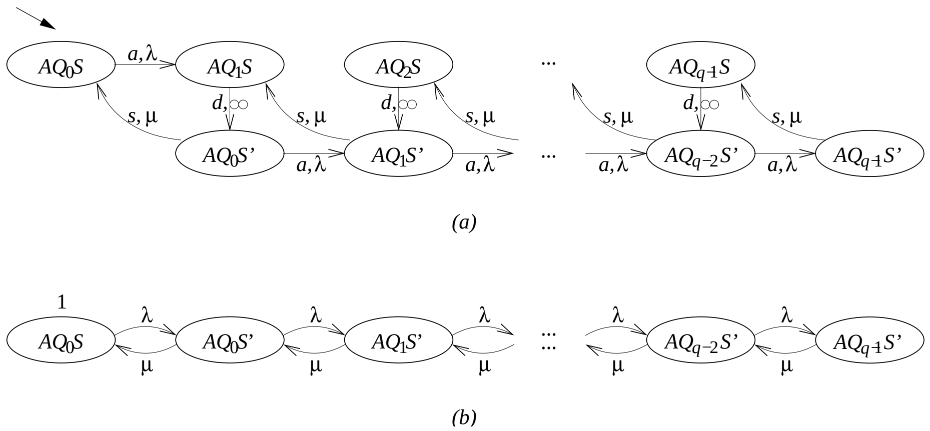
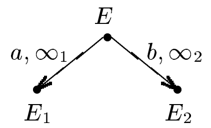


Figure 11.3: Integrated and Markovian semantic models of $QS_{M/M/1/q}$

11.4 Prioritized Choices

The second extension of our Markovian process algebra is concerned with the introduction of *priorities*, which are expressed as positive natural numbers attached to immediate action rates (∞_l). Introducing priorities is necessary to model prioritized choices and to improve the capability of expressing system control mechanisms, such as preemption.

Higher priority immediate actions take precedence over lower priority ones. To make this clear, let us consider a system E that initially can perform either an immediate action a with priority 1 or an immediate action b with priority 2: $<a, \infty_1>.E_1 + <b, \infty_2>.E_2$. The integrated interleaving semantic model of E has the two following initial transitions:



If E represents a closed system, then only the transition labeled with action $<b, \infty_2>$ can be actually executed. If instead E represents an open system, then the execution of action $<b, \infty_2>$ may be disabled by the environment. For instance, $E \,\|_{\{b\}}\, \underline{0}$ has a single initial transition labeled with action $<a, \infty_1>$.

In the case of synchronization of an immediate action and a passive action, the resulting immediate action inherits the priority of the original immediate action.

Example 11.3 Let us consider a variant of the QS of Ex. 11.2 in which there are two different classes of customers, reds and blacks, with two different arrival rates, λ_r and λ_b. The service center comprises two distinct queues of capacity $q-1$ for the two classes of customers. In the situation in which both queues are nonempty and the server is free, the first come red customer must be served, i.e. red customers take precedence over black customers. This can be easily modeled in our Markovian process algebra extended with priorities as follows:

$$\begin{aligned}
QS_{prio} &\triangleq (Arrivals_r \parallel_\emptyset Arrivals_b) \parallel_{\{a_r, a_b\}} ((Queue_{r,0} \parallel_\emptyset Queue_{b,0}) \parallel_{\{d_r, d_b\}} Server) \\
Server &\triangleq \langle d_r, \infty_r \rangle . \langle s, \mu \rangle . Server \; + \\
&\quad\;\, \langle d_b, \infty_b \rangle . \langle s, \mu \rangle . Server
\end{aligned}$$

where $Arrivals_r$ ($Arrivals_b$) is the same as $Arrivals$ in which every action type is given subscript r (b), $Queue_{r,0}$ ($Queue_{b,0}$) is the same as $Queue_0$ in which every action type is given subscript r (b), and $r > b$.

Note that in the model above, no preemption can be exercised on the black customer being served in the case a red customer arrives at the service center. To take this into account, it is sufficient to modify the model of the server as follows:

$$\begin{aligned}
Server &\triangleq \langle d_r, \infty_r \rangle . \langle s, \mu \rangle . Server_r \; + \\
&\quad\;\, \langle d_b, \infty_b \rangle . \langle s, \mu \rangle . Server_b \\
Server_r &\triangleq \langle s, \mu \rangle . Server \\
Server_b &\triangleq \langle s, \mu \rangle . Server \; + \\
&\quad\;\, \langle d_r, \infty_r \rangle . \langle s, \mu \rangle . Server_b
\end{aligned}$$

where the second summand of $Server_b$ describes the service of the newly arrived, preempting red customer. In such a model the memoryless property of exponential distributions guarantees that the remaining time to the completion of the service of a preempted black customer is still exponentially distributed with rate μ. Therefore, the first summand of $Server_b$ is used to describe both the service of a black customer with no interruption and the service of a black customer which has been interrupted several times. ■

11.5 Probabilistic Choices

The third extension of our Markovian process algebra is concerned with the introduction of *weights*, which are expressed as positive real numbers attached to immediate action rates ($\infty_{l,w}$, thus resembling immediate transitions of generalized stochastic Petri nets [6]). Introducing weights is necessary to model probabilistic choices and to improve the capability of expressing system control mechanisms, such as probabilistic events.

The execution probability of immediate actions at the same priority level is proportional to their weights. To make this clear, let us consider a system E that initially can perform either an immediate action a with priority 1 and weight 2 or an immediate action b with priority 1 and weight 3: $\langle a, \infty_{1,2} \rangle . E_1 + \langle b, \infty_{1,3} \rangle . E_2$. The integrated interleaving semantic model of E has the two following initial transitions:

$$\begin{tikzcd}
 & E \arrow[dl, "{a, \infty_{1,2}}"'] \arrow[dr, "{b, \infty_{1,3}}"] & \\
E_1 & & E_2
\end{tikzcd}$$

If E represents a closed system, then the former transition is executed with probability $2/(2+3) = 0.4$, while the latter transition is executed with probability $3/(2+3) = 0.6$. If instead E represents an open system, then the execution of one of its two actions may be disabled by the environment. For instance, $E \parallel_{\{b\}} \underline{0}$ has a single initial transition labeled with action $<a, \infty_{1,2}>$ which is executed with probability $2/2 = 1$.

In summary, the strategy adopted to choose among several alternative immediate actions is the *preselection policy*: the immediate actions having the highest priority level are singled out, then each of them is given an execution probability proportional to its weight.

In the case of synchronization of an immediate action and a passive action, the resulting immediate action inherits also the weight of the original immediate action.

Example 11.4 Let us consider a variant of the QS of Ex. 11.3 such that, in the situation in which both queues are nonempty and the server is free, the first come red customer and the first come black customer have the same priority but different frequencies with which they are served, say $r/(r+b)$ and $b/(r+b)$, respectively. This can be taken into account with our Markovian process algebra extended with weights by simply modifying the model of the server as follows:

$$\begin{aligned} Server \;\; &\stackrel{\Delta}{=} \;\; <d_r, \infty_{l,r}>.<s, \mu>.Server \; + \\ & \quad\;\; <d_b, \infty_{l,b}>.<s, \mu>.Server \end{aligned}$$

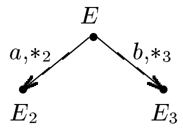
■

11.6 Master-Slaves Synchronization

Our extended Markovian process algebra employs an *asymmetric master-slaves synchronization mechanism*, where exponentially timed and immediate actions (also called active actions) play the role of the masters, in the sense that they determine the rate of the resulting action, while passive actions play the role of the slaves. Such a mechanism is enforced by imposing that, in case of multiway synchronization, at most one active action can be involved while all the other actions must be passive. More formally, we adopt a CSP [109] like parallel composition operator, which allows for multiway synchronizations by assuming that the result of the synchronization of two actions with type a is again an action with type a. In addition, we impose that a synchronization between two actions of type a may occur only if either they are both passive actions (and the result is a passive action of type a), or one of them is an active action and the other one is a passive action (and the result is an active action of type a).

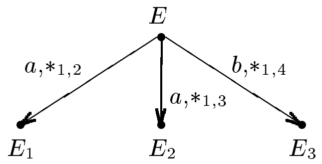
So far we have considered particular kinds of binary synchronizations in which an active action of a process could synchronize with a single passive action of another process only. However, if several alternative passive actions of a given type may synchronize with the same active action of that type, it remains to establish how we choose among those passive actions. This is accomplished in two steps.

First of all, we endow passive actions with positive natural numbers acting as *reactive priorities* ($*_l$). Unlike priorities of immediate actions, reactive priorities are enforced only among passive actions of the same type, which makes it safe to discard lower priority passive transitions of a given type (even in an open system). To make this clear, let us consider a system E that initially can perform a passive action a with priority 1, a passive action a with priority 2, or a passive action b with priority 3: $<a, *_1>.E_1 + <a, *_2>.E_2 + <b, *_3>.E_3$. The integrated interleaving semantic model of E has the two following initial transitions:



As can be noted, a transition labeled with action $<a, *_2>$ is in the model above because the highest priority transition has a different type, whereas there is no transition labeled with action $<a, *_1>$ because of the presence of a higher priority transition of the same type. Due to the reactive meaning ascribed to priorities of passive actions, the environment cannot disable the higher priority passive a action and enable the lower priority passive a action at the same time, so it is safe to neglect lower priority passive actions of a given type. Since the role of the reactive priorities is to realize a choice mechanism among passive actions of the same type, the choice among passive actions of different types is nondeterministic, i.e. it is guided by the type of the selected active action. Thus, in the example above, the choice between $<a, *_2>$ and $<b, *_3>$ is nondeterministic.

Second, we endow passive actions with positive real numbers acting as *reactive weights* ($*_{l,w}$). Unlike weights of immediate actions, reactive weights determine the choice only among passive actions of the same type. To make this clear, let us consider a system E that initially can perform a passive action a with priority 1 and weight 2, a passive action a with priority 1 and weight 3, or a passive action b with priority 1 and weight 4: $<a, *_{1,2}>.E_1 + <a, *_{1,3}>.E_2 + <b, *_{1,4}>.E_3$. The integrated interleaving semantic model of E has the three following initial transitions:

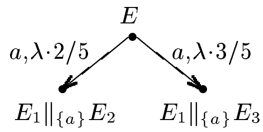


Because of the reactive interpretation of passive action weights, the first transition is executed with probability $2/(2+3) = 0.4$, the second with probability $3/(2+3) = 0.6$, and the third with probability $4/4 = 1$. Note that the sum of the probabilities of executing actions of type a is 1, and the same holds for actions of type b. This is a consequence of the fact that the role of the reactive weights is to realize a choice mechanism among passive actions of the same type and priority level. The choice among passive actions of different types is nondeterministic, i.e. it is guided by the type of the selected active action. Thus, in the example above, the choice between a passive action of type a and a passive action of type b is nondeterministic.

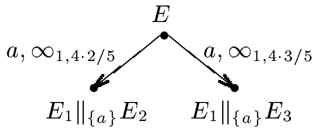
In summary, the strategy adopted to choose among several alternative passive actions is the *reactive preselection policy*: for a given type, the passive actions of that type having the highest priority level are singled out, then each of them is given an execution probability proportional to its weight.

We are now in a position of explaining how the rate of an action resulting from a master-slaves synchronization is determined. In the case of synchronization between an exponentially timed action of rate λ

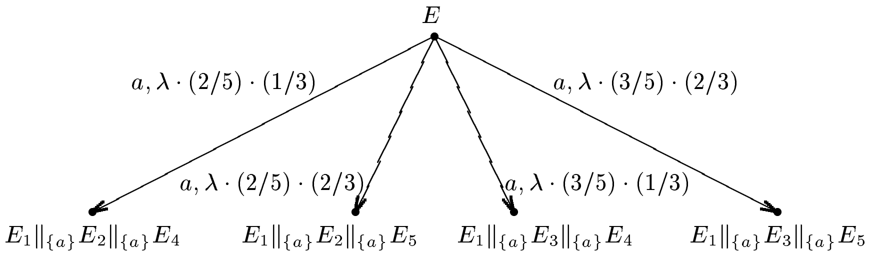
and a passive action of the same type, the resulting rate is $\lambda \cdot p$ where p is the execution probability of the passive action. As an example, term E defined by $<a, \lambda>.E_1 \parallel_{\{a\}} (<a, *_{1,2}>.E_2 + <a, *_{1,3}>.E_3)$ has the two following initial transitions:



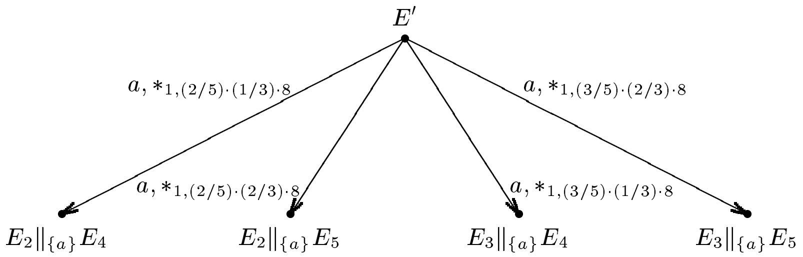
In the case of synchronization between an immediate action of rate $\infty_{l,w}$ and a passive action of the same type, the resulting rate is $\infty_{l,w \cdot p}$ where p is the execution probability of the passive action. As an example, term E defined by $<a, \infty_{1,4}>.E_1 \parallel_{\{a\}} (<a, *_{1,2}>.E_2 + <a, *_{1,3}>.E_3)$ has the two following initial transitions:



In the case of a multiway synchronization where an active action is synchronized with several passive actions, each passive action is chosen by performing an *independent* choice. That is, if an exponentially timed (immediate) action with rate λ ($\infty_{l,w}$) synchronizes with n passive actions of the same type, the resulting rate is $\lambda \cdot \prod_{i=1}^{n} p_i$ ($\infty_{l,w \cdot \prod_{i=1}^{n} p_i}$) where p_i is the execution probability of the i-th passive action involved in the synchronization. As an example, term E defined by $<a, \lambda>.E_1 \parallel_{\{a\}} (<a, *_{1,2}>.E_2 + <a, *_{1,3}>.E_3) \parallel_{\{a\}} (<a, *_{1,1}>.E_4 + <a, *_{1,2}>.E_5)$ has the four following initial transitions:



We observe that our generative master-reactive slaves synchronization mechanism complies with the *bounded capacity assumption* [107], which establishes that the rate of an action cannot be arbitrarily increased/decreased when synchronizing it with several actions, thereby imposing a safe modeling methodology from the stochastic viewpoint. In the case of exponentially timed actions, this assumption is satisfied because the generative-reactive synchronization mechanism preserves the average sojourn time in a state. For instance, in the example above with four synchronization transitions leaving E, we have that the rates of the four transitions sum up to λ, which is exactly the rate of the only active action present in E. Likewise, in the case of immediate actions, this assumption is satisfied because the generative-reactive synchronization mechanism preserves the total weight in a state. The same holds true in the case of multiway synchronizations which include passive actions only. As an example, consider term E' defined by $(<a, *_{1,2}>.E_2 + <a, *_{1,3}>.E_3) \parallel_{\{a\}} (<a, *_{1,1}>.E_4 + <a, *_{1,2}>.E_5)$ obtained from term E above by removing the synchronization with the active action $<a, \lambda>$. Term E' has four synchronization passive transitions, each of which has a weight defined as follows: first we compute the product of the probabilities of independently selecting the two participating passive actions within their processes, then we transform such a resulting joint probability into a weight via a normalization factor equal to the sum of the total weights for that action type of the two involved processes. We depict below the four synchronization passive transitions of E':



It is easy to see that the weights of the four passive transitions above sum up to the total weight $(2+3)+(1+2) = 8$ of action type a in state E', thus conforming to the bounded capacity assumption. Moreover, as we shall see in Sect. 11.8 through the introduction of the cooperation structure model, the choice of such a normalization factor makes it easy to understand the probabilities of the transitions executable by a term. For example, in $E' + {<}a, *_{1,5}{>}.\underline{0}$ the probability of performing ${<}a, *_{1,5}{>}$ is simply given by $5/(5+8)$, where 8 is determined by computing the overall weight at priority level 1 of the passive actions of type a syntactically included in E' which are executable (even through synchronization).

Additionally, we point out that in our Markovian framework extended with immediate actions it is possible to simulate a synchronization between two a actions with rate λ and μ, respectively, whose duration is the *maximum* of the two durations [99]. If we denote by τ an action type representing an invisible activity, this is easily achieved by means of a term like ${<}\tau, \lambda{>}.{<}a, \infty_{l,w}{>}.\underline{0} \,\|_{\{a\}}\, {<}\tau, \mu{>}.{<}a, *_{l',w'}{>}.\underline{0}$, as it gives rise to the first CTMC depicted in Sect. 11.2.

Example 11.5 Attaching reactive priorities and weights to passive actions turns out to be advantageous from the modeling viewpoint as it allows more compact process algebraic descriptions to be obtained. As an example, let us consider a variant of the QSs of Ex. 11.3 and 11.4 in which there are n classes of customers, with class i, $1 \leq i \leq n$, having arrival rate λ_i, service priority l_i, and service frequency $w_i / \sum_{j=1}^{n} w_j$. If we denote by $\prod$ the parallel composition of several terms which do not synchronize on any action, the QS above can be modeled in our Markovian process algebra extended with reactive priorities and weights as follows:

$$
\begin{array}{rcl}
QS_n & \stackrel{\Delta}{=} & \prod\limits_{i=1}^{n} Arrivals_i \,\|_{\{a_i \mid 1 \leq i \leq n\}} (\prod\limits_{i=1}^{n} Queue_{i,0} \,\|_{\{d\}}\, Server) \\
Arrivals_i & \stackrel{\Delta}{=} & {<}a_i, \lambda_i{>}.Arrivals_i \\
Queue_{i,0} & \stackrel{\Delta}{=} & {<}a_i, *_{1,1}{>}.Queue_{i,1} \\
Queue_{i,h} & \stackrel{\Delta}{=} & {<}a_i, *_{1,1}{>}.Queue_{i,h+1} + \\
 & & {<}d, *_{l_i,w_i}{>}.Queue_{i,h-1}, \;\; 0 < h < q-1 \\
Queue_{i,q-1} & \stackrel{\Delta}{=} & {<}d, *_{l_i,w_i}{>}.Queue_{i,q-2} \\
Server & \stackrel{\Delta}{=} & {<}d, \infty_{1,1}{>}.{<}s, \mu{>}.Server
\end{array}
$$

It is worth observing that the model above is scalable w.r.t. the number of classes, in the sense that the description of the server does not need to be modified when adding/removing a class of customers. This is made possible by the fact that the information about the service priority and frequency of each class must not necessarily be described within the server (as it would be if priorities and weights could not be attached to passive actions), but can be described in the model for the queue corresponding to the class. ■

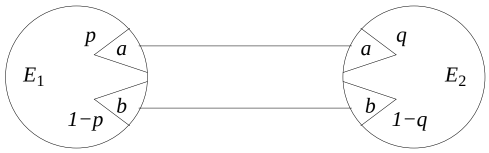


Figure 11.4: Synchronization between generative processes

In general our master-slaves synchronization mechanism can be interpreted as an extension to priorities and exponential time of the probabilistic synchronization mechanism presented in Chapter 10 based on a mixture of the generative and reactive models of probabilistic processes of [80]. As we explained in Chapter 10, when we add selection mechanisms in process algebras, based e.g. on probabilities, we have to face the following problem. Let us consider a system $E_1 \parallel_{\{a,b\}} E_2$ composed of two sequential processes E_1 and E_2 which must synchronize over actions of type a and b. Suppose that E_1 may execute an action a with probability p and an action b with probability $1 - p$, while the other process may execute an action a with probability q and an action b with probability $1 - q$. This scenario can be represented by a picture like that of Fig. 11.4, where circles describe the current state of processes, typed triangles with attached probabilities represent the alternative actions enabled by a process, and links represent possible synchronizations. The problem is: How do we choose the synchronization to be performed? The choice cannot be made independently by the two processes, because e.g. if the lefthand process locally chooses a and the righthand process locally chooses b, then no synchronization is possible between the two processes. In order to have a clear model of synchronization, it is important that each synchronizing process can *independently* choose one of its local actions.

The solution to this problem that we adopted in Chapter 10 is to designate some actions (the master actions) as behaving generatively and the other actions (the slave actions) as behaving reactively, and by imposing (as we do in this chapter) that master actions can synchronize with slave actions only. Therefore we can have a synchronization between a master and a slave action (generative-reactive synchronization) or a synchronization between two slave actions (reactive-reactive synchronization), and in general in a multiway synchronization we have a master-slaves synchronization, i.e. a master action synchronizing with several slave actions. In the master-slaves synchronization mechanism of our Markovian process algebra, which extends the mechanism explained above to priorities and exponential time, we have that, in a system state, first a *master choice* is generatively made according to the rates of the master actions. Then, if the chosen master action must synchronize, several independent *slave choices* are reactively made in each synchronizing process among the slave actions with the type of the selected master action, according to their reactive rates.

11.7 Syntax and Semantics for $\mathrm{EMPA}_{\mathrm{gr}}$

In this section we formalize the syntax and the semantics for the process algebra informally presented in the previous section. More precisely, we define the syntax of an extended Markovian process algebra with generative-reactive synchronizations called $\mathrm{EMPA}_{\mathrm{gr}}$. Then we introduce the master-slaves transition system model and we present an operational semantics that maps $\mathrm{EMPA}_{\mathrm{gr}}$ onto such a model.

11.7.1 Syntax and Informal Semantics

The main ingredients of our calculus are the actions, each composed of a type and a rate, and the algebraic operators. As far as actions are concerned, based on their rates they are classified into exponentially timed, immediate, and passive, as already seen. Moreover, based on their types they are classified into visible and invisible depending on whether they are different or equal to τ, as usual.

Definition 11.1 *Let $AType$ be the set of action types, including the invisible type τ, and $ARate = MRate \cup SRate$ be the set of action rates, where $MRate = \mathbb{R}_+ \cup \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$ is the set of rates of master actions (exponentially timed and immediate rates) and $SRate = \{*_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$ is the set of rates of slave actions (passive rates). We use a to range over $AType$, $\tilde{\lambda}$ to range over $ARate$, λ to range over $\mathbb{R}_+$ (exponentially timed rates), and $\bar{\lambda}$ to range over $MRate$. The set of actions is defined by*

$$Act = AType \times ARate$$

■

Definition 11.2 *Let $Const$ be a set of constants ranged over by A and let $ATRFun = \{\varphi : AType \longrightarrow AType \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of action type relabeling functions ranged over by φ. The set $\mathcal{L}$ of process terms of $\mathrm{EMPA}_{\mathrm{gr}}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$. ■

The *null term* "$\underline{0}$" is the term that cannot execute any action.

The *action prefix operator* "$<a, \tilde{\lambda}>.\_$" denotes the sequential composition of an action and a term. Term $<a, \tilde{\lambda}>.E$ can execute an action with type a and rate $\tilde{\lambda}$ and then behaves as term E.

The *functional abstraction operator* "$\_/L$" abstracts from the type of the actions. Term E/L behaves as term E except that the type a of each executed action is turned into τ whenever $a \in L$.

The *functional relabeling operator* "$\_[\varphi]$" changes the type of the actions. Term $E[\varphi]$ behaves as term E except that the type a of each executed action becomes $\varphi(a)$.

The *alternative composition operator* "$\_ + \_$" expresses a choice between two terms. Term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is executed. As we have already seen, the choice is solved according to the race policy in case of exponentially timed actions, the preselection policy in case of immediate actions, and the reactive preselection policy in case of passive actions.

The *parallel composition operator* "$\_ \|_S \_$" expresses the concurrent execution of two terms. Term $E_1 \|_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S according to the two following synchronization disciplines. The synchronization discipline on action types establishes that two actions can synchronize if and only if they have the same observable type in S, which becomes the resulting type. The synchronization discipline on action rates is the generative master-reactive slaves mechanism explained in Sect. 11.6. In case of synchronization of an active action a having rate $\tilde{\lambda}$ executed by E_1 (E_2) with a passive action a having rate $*_{l,w}$ executed by E_2 (E_1), the resulting active action a has rate/weight given by the original rate/weight multiplied by the probability that E_2 (E_1) chooses the passive action at hand among its passive actions of type a. Instead, in case of synchronization of two passive actions a having rate $*_{l_1,w_1}$ and $*_{l_2,w_2}$ executed by E_1 and E_2, respectively, the resulting passive action of type a has priority level given by the maximum $l_{\max}$ between l_1 and l_2 and weight given by the probability that E_1 and E_2 independently choose the two actions, multiplied by a normalization factor given by the overall weight of the passive actions of type a executable by E_1 and E_2 at the priority level $l_{\max}$. The choice of such a normalization factor and of the priority level for the resulting passive action complies with the bounded capacity assumption, and makes the structure of synchronizations in a system state easier to understand, as will be formally shown in Sect. 11.8.

Finally, let partial function $Def : Const \rightarrow\!\!\!\!\circ\;\; \mathcal{L}$ be a set of *constant defining equations* of the form $A \stackrel{\Delta}{=} E$. In order to guarantee the correctness of recursive definitions, as usual we restrict ourselves to the set $\mathcal{G}$ of terms that are closed and guarded w.r.t. Def.

11.7.2 Master-Slaves Transition Systems

The semantic model of $\mathrm{EMPA_{gr}}$ is a special kind of LTS we call *master-slaves transition system* (MSTS for short), whose transitions are labeled with elements of Act. Recalling that active actions play the role of the masters while passive actions play the role of the slaves, each state of a MSTS has a single *master bundle* composed of all the transitions labeled with an active action and, for each action type a, a single *slave bundle* of type a composed of all the transitions labeled with a passive action of type a. Since the operational semantics for $\mathrm{EMPA_{gr}}$ will be defined in such a way that lower priority active transitions are not pruned [1] while lower priority passive transitions of a given type are, all the passive transitions belonging to the same slave bundle of a generated MSTS have the same priority level.

[1]As explained in Sect. 11.6, in an open system the higher priority transitions can be disabled by a context, hence this is necessary to obtain a congruence as we shall see in Sect. 11.9,

Definition 11.3 *A master-slaves transition system (MSTS) is a triple*

$$(S, AType, \longrightarrow)$$

where: [2]

- *S is a set of states;*
- *$AType$ is a set of action types;*
- *$\longrightarrow \in \mathcal{M}(S \times Act \times S)$ is a multiset of transitions such that for all $s \in S$ and $a \in AType$*

$$(s \xrightarrow{a,*_{l_1,w_1}} s_1 \wedge s \xrightarrow{a,*_{l_2,w_2}} s_2) \implies l_1 = l_2$$

A rooted master-slaves transition system (RMSTS) is a quadruple

$$(S, AType, \longrightarrow, s_0)$$

where $(S, AType, \longrightarrow)$ is a MSTS and $s_0 \in S$ is the initial state. ■

We point out that the transition relation is a multiset, not a set. This allows the multiplicity of identically labeled transitions to be taken into account, which is necessary from the stochastic point of view. As an example, if a state has two transitions both labeled with $<a, \lambda>$, using sets instead of multisets would reduce the two transitions into a single one labeled with $<a, \lambda>$, thus erroneously altering the average sojourn time in that state.

The choice among the bundles of transitions enabled in a state is nondeterministic. The choice of a transition within the master bundle of a state is made according to the race policy, i.e. the transition sampling the least duration succeeds, with immediate transitions taking precedence over exponentially timed transitions. We consider the transitions composing a master bundle as grouped according to their priority level. The level zero is composed of all the transitions labeled with exponentially timed actions and, for each $l \in \mathbb{N}_+$, the level l is composed of all the transitions labeled with an immediate action with priority l. If all the transitions composing the master bundle are labeled with exponentially timed actions, then the master bundle includes the group of transitions at level zero only. Supposed that such a group is composed of n transitions labeled with active actions $<a_i, \lambda_i>$, $1 \leq i \leq n$, then the time to choose one of such actions is exponentially distributed with rate $\sum_{1 \leq i \leq n} \lambda_i$ and the probability of choosing a_k is given by $\lambda_k / \sum_{1 \leq i \leq n} \lambda_i$. Otherwise, if there is some transition labeled with an immediate action, the preselection policy is applied, which means that a probabilistic choice is made in zero time according to the weights of the immediate actions labeling the group of transitions at the maximum priority level $l_{\max}$. Supposed that such a group is composed of n transitions labeled with active actions $<a_i, \infty_{l_{\max}, w_i}>$, $1 \leq i \leq n$, then the probability of choosing a_k is given by $w_k / \sum_{1 \leq i \leq n} w_i$.

[2] We use "$\{\!|$" and "$|\!\}$" as brackets for multisets, "$\_ \oplus \_$" to denote multiset union, and $\mathcal{M}(S)$ ($\mathcal{P}(S)$) to denote the collection of multisets over (subsets of) set S.

The choice within a slave bundle of type a is governed by the prelection policy: each transition of the bundle is chosen with probability proportional to its weight. Supposed that such a bundle is composed of n transitions labeled with passive actions $<a_i, *_{l,w_i}>$, $1 \leq i \leq n$, then the probability of choosing a_k is given by $w_k / \sum_{1 \leq i \leq n} w_i$. Since the duration of passive actions is unspecified, the time to choose one of the actions above is unspecified.

We conclude by recalling that passive actions are seen as *incomplete* actions which must synchronize with active actions of the same type of another system component in order to form a complete system. Therefore a fully specified system is *performance closed*, in the sense that it gives rise to a fully probabilistic transition system which does not include slave bundles. If in such a transition system we keep for each state only the highest priority transitions, then we can easily derive a performance model in the form of a DTMC (where D stands for discrete) or a CTMC, depending on whether only immediate transitions occur or not. Should exponentially timed and immediate transitions coexist (in different states), i.e. the system is a semi-Markov process (see Chapter 2), a CTMC can be derived by eliminating the immediate transitions and the related source states and by suitably splitting the exponentially timed transitions entering the removed source states in such a way that they are caused to reach the target states of the removed immediate transitions. The reader interested in the details of this procedure is referred to [19] Chapter 4.

11.7.3 Operational Semantics

The formal semantics for $\mathrm{EMPA_{gr}}$ maps terms onto RMSTSs. We preliminarily provide the following shorthands to make the definition of the operational semantic rules easier.

Definition 11.4 *Given a MSTS $M = (S, AType, \longrightarrow)$, $s \in S$, and $a \in AType$, we denote by $L_a(s)$ the priority level of the slave transitions of type a executable at s ($L_a(s) = 0$ if the slave bundle a of s is empty) and we denote by $W_a(s)$ the overall weight of the slave transitions of type a executable at s:*

$$W_a(s) = \sum \{\!| \, w \mid \exists s' \in S. \, s \xrightarrow{a, *_{L_a(s), w}} s' \, |\!\}$$

Furthermore, we extend the real number multiplication to immediate rates as follows:

$$\infty_{l,w} \cdot p = \infty_{l,w \cdot p}$$

■

The operational semantics for $\mathrm{EMPA_{gr}}$ is the least MSTS $(\mathcal{G}, AType, \longrightarrow)$ satisfying the inference rules of Table 11.1, where in addition to the rules $(Ch1_l)$, $(Ch2_l)$, $(Pa1_l)$, $(Pa2_l)$, $(Sy1_l)$ referring to a move of the lefthand process E_1, we consider also the symmetrical rules $(Ch1_r)$, $(Ch2_r)$, $(Pa1_r)$, $(Pa2_r)$, $(Sy1_r)$ taking into account the moves of the righthand process E_2, obtained by exchanging the roles of terms E_1 and E_2. Similarly to [107], we consider the operational rules as generating a multiset of transitions (consistently with the definition of a MSTS), where a transition has arity m if and only if it can be derived in m possible ways from the operational rules.

$$\textbf{(Pr)}\quad <a,\tilde{\lambda}>.E \xrightarrow{a,\tilde{\lambda}} E$$

$$\textbf{(Hi1)}\quad \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E/L \xrightarrow{a,\tilde{\lambda}} E'/L}\ a \notin L \qquad\qquad \textbf{(Hi2)}\quad \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E/L \xrightarrow{\tau,\tilde{\lambda}} E'/L}\ a \in L$$

$$\textbf{(Re)}\quad \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E[\varphi] \xrightarrow{\varphi(a),\tilde{\lambda}} E'[\varphi]}$$

$$\textbf{(Ch1}_l\textbf{)}\quad \frac{E_1 \xrightarrow{a,\tilde{\lambda}} E'_1}{E_1 + E_2 \xrightarrow{a,\tilde{\lambda}} E'_1} \qquad\qquad \textbf{(Ch2}_l\textbf{)}\quad \frac{E_1 \xrightarrow{a,*_{l,w}} E'_1 \quad l \geq L_a(E_2)}{E_1 + E_2 \xrightarrow{a,*_{l,w}} E'_1}$$

$$\textbf{(Pa1}_l\textbf{)}\quad \frac{E_1 \xrightarrow{a,\tilde{\lambda}} E'_1}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}} E'_1 \parallel_S E_2}\ a \notin S$$

$$\textbf{(Pa2}_l\textbf{)}\quad \frac{E_1 \xrightarrow{a,*_{l,w}} E'_1 \quad l \geq L_a(E_2)}{E_1 \parallel_S E_2 \xrightarrow{a,*_{l,w}} E'_1 \parallel_S E_2}\ a \notin S$$

$$\textbf{(Sy1}_l\textbf{)}\quad \frac{E_1 \xrightarrow{a,\tilde{\lambda}} E'_1 \qquad E_2 \xrightarrow{a,*_{l,w}} E'_2}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}\cdot\frac{w}{W_a(E_2)}} E'_1 \parallel_S E'_2}\ a \in S$$

$$\textbf{(Sy2)}\quad \frac{E_1 \xrightarrow{a,*_{l_1,w_1}} E'_1 \qquad E_2 \xrightarrow{a,*_{l_2,w_2}} E'_2}{E_1 \parallel_S E_2 \xrightarrow{a,*_{\max(l_1,l_2),p\cdot N}} E'_1 \parallel_S E'_2}\ a \in S$$

$$\text{where:}\quad p = \frac{w_1}{W_a(E_1)} \cdot \frac{w_2}{W_a(E_2)} \qquad N = \begin{cases} W_a(E_1) + W_a(E_2) & \text{if } l_1 = l_2 \\ W_a(E_1) & \text{if } l_1 > l_2 \\ W_a(E_2) & \text{if } l_2 > l_1 \end{cases}$$

$$\textbf{(Co)}\quad \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{A \xrightarrow{a,\tilde{\lambda}} E'}\ A \triangleq E$$

Table 11.1: $\mathrm{EMPA_{gr}}$ operational semantics

Some explanations are now in order. First of all, the operational rules give rise to an interleaving semantics, which is made possible by the memoryless property of exponential distributions. The removal of lower priority passive transitions of the same type is carried out in rules $(Ch2_l)$ and $(Ch2_r)$ for the alternative composition operator and rules $(Pa1_l)$ and $(Pa1_r)$ for the parallel composition operator by using $L_a(\_)$. As we shall see in Theorem 11.3 and 11.4, discarding lower priority passive transitions does not compromise the achievement of the congruence property for the Markovian bisimulation equivalence. While higher priority active transitions can be prevented by a context which does not prevent lower priority active transitions (because of their different types), this cannot happen for passive transitions as their priorities are reactive, i.e. imposed only among passive transitions of the same type. We also note that the priorities are interpreted as being global according to the classification of [62], as their scope is not limited to sequential terms but includes terms composed in parallel.

In the case of a synchronization, the evaluation of the rate of the resulting action is carried out by rules $(Sy1_l)$, $(Sy1_r)$, and $(Sy2)$ as follows. Whenever an active action synchronizes with a passive action of the same type, the rate of the resulting active action is evaluated in rules $(Sy1_l)$ and $(Sy1_r)$ by multiplying the rate of the active action by the probability of choosing the passive action. Whenever two passive actions of type a synchronize, instead, the priority level and the weight of the resulting passive action are computed as described by rule $(Sy2)$. In particular, the weight is computed by multiplying the probability p of independently choosing the two original actions by the normalization factor N. As explained in Sect. 11.7.1, N is given by the overall weight of the passive transitions of type a with maximum priority level executable by E_1 and E_2, computed by using $W_a(\_)$.

Definition 11.5 *The integrated interleaving semantics of $E \in \mathcal{G}$ is the RMSTS*

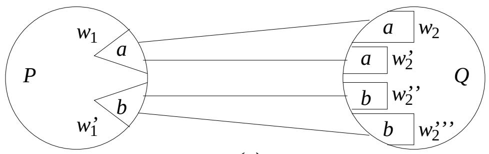
$$\mathcal{I}[\![E]\!] = (\mathcal{G}_E, AType, \longrightarrow_E, E)$$

where $\mathcal{G}_E$ is the set of terms reachable from E according to MSTS $(\mathcal{G}, AType, \longrightarrow)$ and $\longrightarrow_E$ is the restriction of $\longrightarrow$ to transitions between terms in $\mathcal{G}_E$. We say that $E \in \mathcal{G}$ is performance closed if and only if $\mathcal{I}[\![E]\!]$ does not contain passive transitions. ■

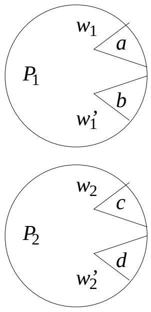
We conclude by recalling that from $\mathcal{I}[\![E]\!]$ two projected semantic models can be obtained by essentially dropping action rates or action types, respectively. Before applying such a transformation to $\mathcal{I}[\![E]\!]$, lower priority active transitions are pruned because E is considered to be a closed system, i.e. it is no longer to be composed with other terms as it describes the whole system we are interested in. The *functional semantics* $\mathcal{F}[\![E]\!]$ is a standard LTS whose transitions are decorated with action types only. The *Markovian semantics* $\mathcal{M}[\![E]\!]$ is instead a CTMC or a DTMC, as seen in Sect. 11.7.2, which is well defined only if E is performance closed.

11.8 The Cooperation Structure Model

In this section we present the model of *cooperation structures*, which allows us to represent the structure of the master-slaves synchronizations that can be performed in a system state, and we prove that the synchronization discipline for $\mathrm{EMPA_{gr}}$ is consistent with it. The importance of such a model is that it shows that the synchronization mechanism of $\mathrm{EMPA_{gr}}$ is natural and easy to understand, and in particular it justifies the choice of the priority level and the weight normalization factor of the passive action resulting from the synchronization of two passive actions.



(a)



(b)

Figure 11.5: Probabilistic cooperation scenarios

As already explained, the synchronization mechanism of $\mathrm{EMPA_{gr}}$ is based on a master-slaves discipline, which establishes that, in a system state, first a choice among the master actions is generatively performed, then a choice among the passive actions with which the selected master action can synchronize is reactively made. As an example, let us consider Fig. 11.5(a), which depicts a state of system $P \,\|_{\{a,b\}}\, Q$. Similarly to Fig. 11.4, each circle describes the current state of a sequential process. The alternative actions currently enabled by a process are represented as labeled triangles if masters and labeled boxes if slaves, with each label being the action type. Each process chooses an action according to its attached rate. [3] For instance, in process P the intended meaning of weights w_1 and w_1' is that the probability of choosing a is proportional to w_1 while the probability of choosing b is proportional to w_1'. Linked triangles/boxes denote cooperating actions. [4] According to the master-slaves approach, there are two steps. In the first step we probabilistically choose, locally to P, between the two master actions a and b according to weights w_1 and w_1' (*master choice*).

[3]In Fig. 11.5 and 11.6 we abbreviate rates simply through weights because we assume that all the occurring master actions are immediate with the same priority level and that all the occurring slave actions are passive with the same priority level.

[4]This dynamic system representation is not to be confused with the static system representation via flow graphs [122].

In the second step, we probabilistically choose, locally to Q, between the two slave actions a according to weights w_2 and w_2' or between the two slave actions b according to weights w_2'' and w_2''', depending on whether master action a or b wins in the first step (*slave choice*).

As far as the master choice is concerned, several processes may be involved. For example, consider Fig. 11.5(b), which depicts a state of system $P_1 \,\|_\emptyset\, P_2$. Conceptually, the choice of the master action can be seen as being performed in two rounds (*interprocess choice* and *intraprocess choice*). In the first round, the interprocess choice is made according to the overall rate of each process. The overall rate of a process state is obtained by summing: the rates of the enabled exponentially timed actions if no immediate action is executable, the weights of the enabled immediate actions at the maximum priority level otherwise. Therefore, in Fig. 11.5(b) the interprocess choice is made according to weights $w_1 + w_1'$ and $w_2 + w_2'$. In the second round, a local choice is performed in the winning process according to the rates of its master actions.

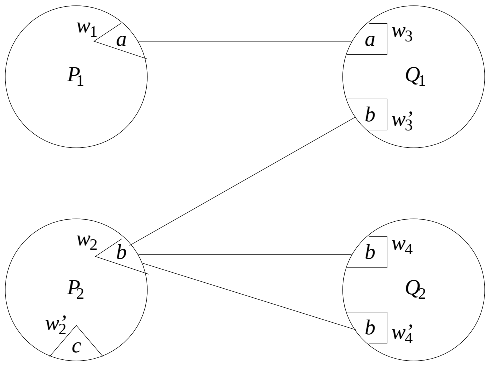


Figure 11.6: A master-slaves cooperation scenario

In general, the selection of the action to be executed in a system state is conceptually carried out in two steps (master choice and slave choice) each composed of two rounds (interprocess choice and intraprocess choice). As an example, look at the scenario depicted in Fig. 11.6, which represents a state of system $(P_1 \,\|_\emptyset\, P_2) \,\|_{\{a,b\}} (Q_1 \,\|_\emptyset\, Q_2)$. In the first step a master action is chosen in two rounds. In the first round an interprocess choice takes place among the processes enabling master actions, hence we probabilistically choose between P_1 and P_2 according to weights w_1 and $w_2 + w_2'$. In the second round a local choice among the master actions of the process that wins the first round occurs. If P_1 wins, a second round is not necessary because P_1 can do one master action only. If P_2 wins, instead, a local choice between the two actions b (weight w_2) and c (weight w_2') occurs in the second round. If the winning master action must not cooperate (action c of P_2) or can be engaged in a single cooperation (action a of P_1), then we are done. If this is not the case (action b of P_2), a second step must be undertaken, where a slave action is chosen in two rounds. In the first round an interprocess choice takes place among the processes enabling slave actions which can cooperate with the winning master action, hence we probabilistically choose between Q_1 and Q_2 according to weights w_3' and $w_4 + w_4'$. In the second round a local choice among the slave actions of the process that wins the first round occurs. If Q_1 wins, a second round is not necessary. If Q_2 wins, instead, we locally choose between the two slave actions b of Q_2, according to weights w_4 and w'_4. To be more precise, since a master action may be synchronized with several slave actions performed by different processes, once the master action has been chosen (through an interprocess choice followed by an intraprocess choice), we have an interprocess choice among groups of processes cooperating to perform the slave actions, followed by an intraprocess choice which is conducted independently within each process of the selected group.

In the following two sections we will present the formal definition of the model of cooperation structures and the result of consistency of the synchronization discipline of $\mathrm{EMPA_{gr}}$ with the cooperation structure model.

11.8.1 Definition of the Cooperation Structure Model

In this section we formally introduce the cooperation structure model which describes the structure of the master slave cooperations performable in a system state, according to the concepts already explained, as a (interprocess and intraprocess) master choice followed by a (interprocess and intraprocess) slave choice.

Given a system state, we define a *process activity* to be an action executable by a sequential process in that state. In particular, when we speak of process activities, we distinguish among actions having the same type. For example, in the scenario of Fig. 11.6 we see the two b actions executable by P_4 and the b action executable by P_3 as three distinguished process activities. In order to distinguish process activities representing actions with the same type, we give a different *name* to each process activity which is unique over the whole system state. Process activities are divided into *master* and *slave* activities, depending on whether they refer to master or slave actions. Moreover given a system state, we define a *system activity* to be an action executable by the whole system in that state. System activities are divided into *local activities* and *cooperation activities* depending on whether they are composed of one process activity only or several process activities each of a different sequential process. For example, in the scenario of Fig. 11.6 we have five system activities, which are divided into one local activity and four cooperation activities.

Let us introduce some notation. We denote by *SProcs* the set of sequential process states, i.e. the states of those processes denoted by circles in the pictures, ranged over by $P, Q, R, \ldots$. We represent a process activity as a pair $<\alpha, \tilde{\lambda}>$ composed of the name α of the activity (which uniquely identifies it in the system state) and the rate $\tilde{\lambda}$ of the action it refers to. We denote the set of names of process activities by $AName$, ranged over by $\alpha, \beta, \ldots$. Moreover we denote: the set of master activities by $MAct = AName \times MRate$, ranged over by m; the set of slave activities by $SAct = AName \times SRate$, ranged over by s; and the set of process activities by $PAct = MAct \cup SAct$. Finally, we represent a system activity as a pair $\langle m, \mathcal{S} \rangle$ composed of a master activity m and a set of slave activities $\mathcal{S}$. We denote the set of system activities by $SysAct = MAct \times \mathcal{P}(SAct)$.

Master Choice

As already explained, the choice of the master activity takes place in two rounds: interprocess choice and intraprocess choice.

We preliminarily introduce some concepts and quantities on which the master choice is based. Whenever a sequential process P can perform alternative master activities $m_{P,i}$, $1 \leq i \leq n$, according to the race policy we assume they are in a timed race condition. Thus, if master activities $m_{P,i}$, $1 \leq i \leq n$, of P are all exponentially timed with rates $\lambda_{P,i}$, $1 \leq i \leq n$, respectively, then the time to choose one of such activities is exponentially distributed with rate $\lambda_P = \sum_{1 \leq i \leq n} \lambda_{P,i}$. Otherwise the exponentially timed ones are ruled out and the selection among the remaining n' immediate master activities $m'_{P,i}$, $1 \leq i \leq n'$, of P, with rate $\infty_{l_{P,i},w_{P,i}}$, $1 \leq i \leq n'$, respectively, is carried out according to a prioritized-probabilistic choice based on the maximum priority level $l_P = \max_{1 \leq i \leq n'} l_{P,i}$ and the total weight $w_P = \sum_{1 \leq i \leq n', l_{P,i} = l_P} w_{P,i}$.

Interprocess Master Choice

The selection among processes P_j, $1 \leq j \leq h$, enabling master activities is carried out according to the overall rates of the processes (hence again to a timed race condition). We associate with each involved process P_j a rate $\tilde{\lambda}_{P_j} \in \mathit{MRate}$ representing the overall rate of the process. In particular, if all the master activities of P_j are exponentially distributed, $\tilde{\lambda}_{P_j}$ is given by λ_{P_j}, otherwise $\tilde{\lambda}_{P_j}$ is given by $\infty_{l_{P_j},w_{P_j}}$. Therefore, if all the processes are exponentially timed, the probability of choosing P_k is given by $\lambda_{P_k} / \sum_{1 \leq j \leq h} \lambda_{P_j}$. Otherwise the exponentially timed ones are ruled out and the selection among the remaining h' processes P'_j, $1 \leq j \leq h'$, is prioritized-probabilistic, where, if $l_{P'_k} = \max_{1 \leq j' \leq h'} l_{P'_{j'}}$, the probability of choosing P'_k is given by $w_{P'_k} / \sum_{1 \leq j \leq h', l_{P'_j} = l_{P'_h}} w_{P'_j}$.

Intraprocess Master Choice

A master activity of the previously chosen process P is locally selected with probability λ/λ_P if its rate is λ and all the master activities of P are exponentially timed, w/w_P if its rate is $\infty_{l,w}$ with $l = l_P$.

Slave Choice

Once the master activity has been chosen, we have an interprocess choice among groups of processes cooperating to perform slave activities, followed by an intraprocess choice which is conducted independently within each process of the selected group.

Consider the scenario of Fig. 11.7(a), where we have two alternative system activities of $P \parallel_{\{a\}} (Q_1 \parallel_{\{a\}} (Q_2 \parallel_{\emptyset} Q_3))$, each formed by a three-way cooperation including the winning master activity and two cooperating slave activities. Here we have to choose which is the pair of processes which cooperate with the master activity α. In this particular scenario we simply have to choose which process between Q_2 and Q_3 cooperates with process Q_1. We may describe this scenario as in Fig. 11.7(b), where we have represented the two possible system activities that can originate from the master activity of P as a single *cooperation tree* indicating that process Q_1 may cooperate with either process Q_2 or process Q_3.

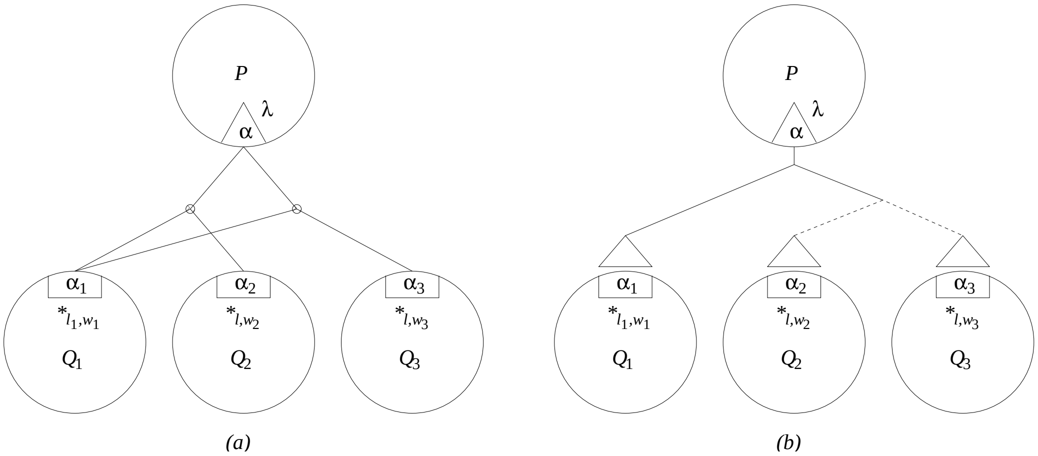


Figure 11.7: A cooperation structure

In general a cooperation tree represents a choice among a set of system activities that originate from a given master activity. A cooperation tree is a binary tree with two kinds of nodes:

- The nodes with outgoing dashed lines, which are called *choice nodes*, represent the fact that either activities of their lefthand subtree or activities of their righthand subtree can take part in the final system activity.
- The nodes with outgoing solid lines, which are called *cooperation nodes*, represent the fact that activities which are possible according to their lefthand subtree must cooperate with activities which are possible according to their righthand subtree to take part in the final system activity.

The leaves of the tree are represented by triangles. Each leaf is adjacent to a different process, where the base of the triangle singles out a subset of the slave activities of such a process (the slave activities just underneath the triangle base; see also Fig. 11.8(b)). The intended meaning of the leaf is that one and only one slave activity in this set can take part in the final system activity.

We represent cooperation trees as terms CT where choice nodes are denoted by $CT \vee CT$, cooperation nodes by $CT \wedge CT$, and leaves by sets $\mathcal{S}$ of slave activities.

Definition 11.6 *The syntax of cooperation trees is defined as follows:*

$$CT ::= \mathcal{S} \mid CT \vee CT \mid CT \wedge CT$$

where $\mathcal{S}$ are nonempty sets of process activities such that all process activities in $\mathcal{S}$ belong to a common process, hereafter denoted by proc($\mathcal{S}$), and every $\mathcal{S}$ occurring in a term CT refers to a different process. We denote by CTrees the set of cooperation trees. ■

The root of a cooperation tree is connected with a solid line to the master activity it refers to. The pair composed of a master activity and the connected cooperation tree is called a cooperation structure.

Definition 11.7 *A cooperation structure is a pair $\langle m, CT \rangle$ composed of a master activity m and a cooperation tree CT. We denote by CStructs the set of cooperation structures.* ■

Definition 11.8 *The semantic mapping $[\![\,]\!]_{\mathrm{sa}} : CStructs \longrightarrow \mathcal{P}(SysAct)$ which associates a set of system activities with a cooperation structure is defined in Table 11.2.* ■

$$[\![m, \mathcal{S}]\!]_{\mathrm{sa}} = \{\langle m, \{s\}\rangle \mid s \in \mathcal{S}\}$$

$$[\![m, CT_1 \vee CT_2]\!]_{\mathrm{sa}} = [\![m, CT_1]\!]_{\mathrm{sa}} \cup [\![m, CT_2]\!]_{\mathrm{sa}}$$

$$[\![m, CT_1 \wedge CT_2]\!]_{\mathrm{sa}} = \{\langle m, \mathcal{S}_1 \cup \mathcal{S}_2\rangle \mid \langle m, \mathcal{S}_1\rangle \in [\![m, CT_1]\!]_{\mathrm{sa}} \wedge \langle m, \mathcal{S}_2\rangle \in [\![m, CT_2]\!]_{\mathrm{sa}}\}$$

Table 11.2: Set of system activities described by a cooperation structure

Note that a cooperation structure represents a set of possible system activities such that each process participating in the cooperation can *independently* choose its own participating slave activity (one among those singled out by the leaf adjacent to such a process).

As we will formally show in Sect. 11.8.2, in the case of system states representable by $\mathrm{EMPA}_{\mathrm{gr}}$ terms, the set of system activities that originate from a winning master activity can always be represented by a cooperation structure. This is due to the fact that in $\mathrm{EMPA}_{\mathrm{gr}}$ synchronization is based on action types, so, as explained in Sect. 6, if several processes are required to cooperate with a winning master activity of a certain type, then each cooperating process can independently choose one of its executable slave activities among those with that type. As a consequence, in $\mathrm{EMPA}_{\mathrm{gr}}$ it is not necessary to break again the symmetry of cooperations at the level of the slave activities, e.g. by designating some activities as submaster activities, in order to have a clear model of synchronization.

For example, a scenario like that represented in Fig. 11.8(a), which is similar to the "problematic" scenario of Fig. 11.4 but at the level of slave cooperation, cannot arise in $\mathrm{EMPA}_{\mathrm{gr}}$. First of all we observe that such a scenario, where Q_1 and Q_2 cannot independently choose their own participating activity, is not representable by a cooperation tree. This can be formally seen as follows. Since for a given master activity we can have only a cooperation tree and each leaf must be adjacent to a different process, the only cooperation structure that could represent the scenario of Fig. 11.8(a) (hence that refers to all activities α_1, β_1, α_2, and β_2 and that makes processes Q_1 and Q_2 cooperate with the master activity α) is that of Fig. 11.8(b). However, the scenario described in Fig. 11.8(b) is different from that of Fig. 11.8(a) because it represents independent

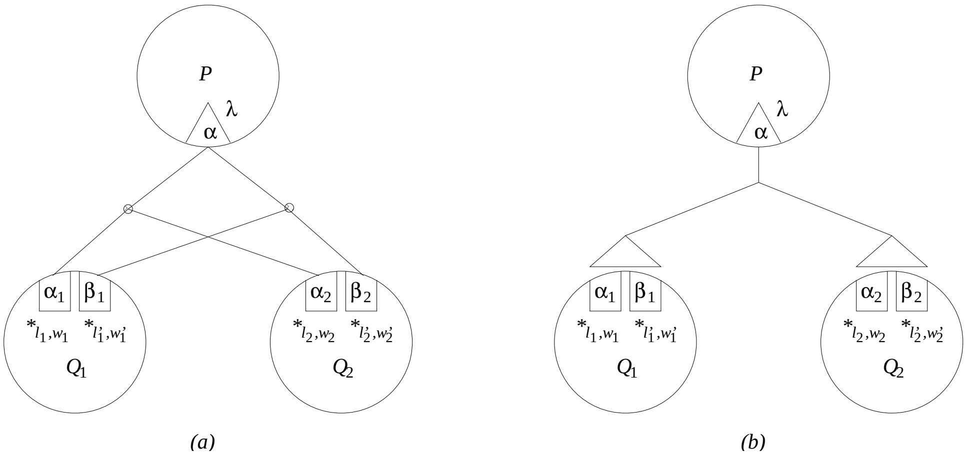


Figure 11.8: System activities not originating from a cooperation structure

local choices of the participating activity of Q_1 and Q_2, hence it admits all possible cooperations between activities of Q_1 and activities of Q_2 (hence gives rise to 4 system activities instead of 2). On the other hand, a scenario like that of Fig. 11.8(a) cannot arise from an $\mathrm{EMPA_{gr}}$ term. This because, since all the slave activities that cooperate with the master activity α must have the same action type a, the only term from which a scenario like that of Fig. 11.8(a) could arise is $P \,\|_{\{a\}}(Q_1 \,\|_{\{a\}}\, Q_2)$, where α is executable by P, α_1 and β_1 are executable by Q_1, and α_2 and β_2 are executable by Q_2. However, since in such a term each process cooperating with the master action a may independently choose its own participating slave activity, we would get all possible cooperations between activities of Q_1 and activities of Q_2, hence again 4 system activities instead of 2. Note that the structure of the cooperations performable in the $\mathrm{EMPA_{gr}}$ term described above is, instead, indeed representable by a cooperation tree: it is represented exactly by the cooperation tree of Fig. 11.8(b).

Supposing that the slave choice can always be represented by a cooperation tree leads to the possibility of a fairly simple method based on priorities and probabilities to choose the group of slave processes that must cooperate with the winning master process. For example, in the cooperation structure of Fig. 11.7(b), the choice is done, similarly to the case of the interprocess master choice, by considering the maximum priority levels l_{Q_2} and l_{Q_3} and the total weights w_{Q_2} and w_{Q_3} of the slave activities of Q_2 and Q_3 singled out by the leaves of the cooperation tree. Since $l_{Q_2} = l = l_{Q_3}$, the choice is done probabilistically according to total weights $w_{Q_2} = w_2$ and $w_{Q_3} = w_3$. Once the set of processes that must cooperate to perform the slave activity is established, similarly to the case of the intraprocess master choice each process may independently choose one of the slave activities singled out by the corresponding leaf of the cooperation tree according to their priorities and weights.

Interprocess Slave Choice

The choice among the groups of processes that can cooperate with the selected master activity is carried out according to a prioritized-probabilistic choice. Hence, given a cooperation tree, each group of cooperating processes is given a priority level and a weight. Since the probabilistic choice must be performed only among the groups of processes with the highest priority, we only consider the weight distribution over such groups of processes.

Definition 11.9 *Given an arbitrary set* Γ*,* $WDist_\Gamma$ *is the set of weight distributions over subsets of* Γ*, i.e. partial functions* $wd_\Gamma : \mathcal{P}(\Gamma) \multimap \mathbb{R}_+$*.* ■

We now define a semantic mapping that associates with every cooperation tree a priority level $l \in \mathbb{N}_+$, which is the highest priority level of the groups of cooperating processes determined by the tree, and a process weight distribution $wd_{SProcs} \in WDist_{SProcs}$, which is the weight distribution over the groups of cooperating processes with the highest priority level (the domain of the weight distribution).

Definition 11.10 *The semantic mappings* $[\![\,]\!]_{\rm pri} : CTrees \longrightarrow \mathbb{N}_+$*, which associates a priority level with a cooperation tree, and* $[\![\,]\!]_{\rm pwd} : CTrees \longrightarrow WDist_{SProcs}$*, which associates a process weight distribution with a cooperation tree, are defined in Table 11.3.* ■

In the case of a leaf $\mathcal{S}$, its priority level is the maximum priority level of the process activities in $\mathcal{S}$, while its process weight distribution is defined only over the singleton composed of the process owner of the activities in $\mathcal{S}$ and associates with it the sum of the weights of the activities in $\mathcal{S}$ which have the maximum priority level.

In the case of a choice node $CT_1 \vee CT_2$, its priority level is the maximum of the priority levels of CT_1 and CT_2. As far as its weight distribution is concerned, we have three cases. If the priority level of CT_1 is equal to the priority level of CT_2, then we consider all the groups of processes in both $[\![CT_1]\!]_{\rm pwd}$ and $[\![CT_2]\!]_{\rm pwd}$ with weights unchanged. [5] If the priority level of CT_1 (CT_2) is greater than the priority level of CT_2 (CT_1), then we consider only the sets of processes in $[\![CT_1]\!]_{\rm pwd}$ ($[\![CT_2]\!]_{\rm pwd}$) with weight unchanged.

In the case of a cooperation node $CT_1 \wedge CT_2$, we consider all the groups of processes that can be obtained as the union of a group of processes $sprocs_1$ of $[\![CT_1]\!]_{\rm pwd}$ and a group of processes $sprocs_2$ of $[\![CT_2]\!]_{\rm pwd}$. Since we want that the choice of the group of cooperating processes among those in $[\![CT_1]\!]_{\rm pwd}$ is performed independently from the choice of the group of cooperating processes among those in $[\![CT_2]\!]_{\rm pwd}$, we take as probability for $sprocs_1 \cup sprocs_2$ the product of the probabilities of $sprocs_1$ and $sprocs_2$. Similarly to the synchronization of passive actions in $\mathrm{EMPA_{gr}}$, such a probability is then normalized so that the total weight of $CT_1 \wedge CT_2$ is equal to the overall weight of the lefthand and righthand subtrees at the maximum level of priority, which is taken to be the priority level of $CT_1 \wedge CT_2$.

[5] Due to the conditions in the syntax of terms CT, we have that $[\![CT_1]\!]_{\rm pwd}$ and $[\![CT_2]\!]_{\rm pwd}$ cannot include a common group of processes.

$$\llbracket \mathcal{S} \rrbracket_{\rm pri} = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}\}$$

$$\llbracket \mathcal{S} \rrbracket_{\rm pwd} = \{\ (\{proc(\mathcal{S})\},\ \sum\{\!| w \mid <\alpha, *_{l,w}> \in \mathcal{S} \wedge l = \llbracket \mathcal{S} \rrbracket_{\rm pri} |\!\})\ \}$$

$$\llbracket CT_1 \vee CT_2 \rrbracket_{\rm pri} = \max\{\llbracket CT_1 \rrbracket_{\rm pri}, \llbracket CT_2 \rrbracket_{\rm pri}\}$$

$$\llbracket CT_1 \vee CT_2 \rrbracket_{\rm pwd} = \begin{cases} \llbracket CT_1 \rrbracket_{\rm pwd} \cup \llbracket CT_2 \rrbracket_{\rm pwd} & \text{if } \llbracket CT_1 \rrbracket_{\rm pri} = \llbracket CT_2 \rrbracket_{\rm pri} \\ \llbracket CT_1 \rrbracket_{\rm pwd} & \text{if } \llbracket CT_1 \rrbracket_{\rm pri} > \llbracket CT_2 \rrbracket_{\rm pri} \\ \llbracket CT_2 \rrbracket_{\rm pwd} & \text{if } \llbracket CT_2 \rrbracket_{\rm pri} > \llbracket CT_1 \rrbracket_{\rm pri} \end{cases}$$

$$\llbracket CT_1 \wedge CT_2 \rrbracket_{\rm pri} = \max\{\llbracket CT_1 \rrbracket_{\rm pri}, \llbracket CT_2 \rrbracket_{\rm pri}\}$$

$$\begin{aligned} \llbracket CT_1 \wedge CT_2 \rrbracket_{\rm pwd} = \{\ & (sprocs_1 \cup sprocs_2, \tfrac{w_1}{TW(\llbracket CT_1 \rrbracket_{\rm pwd})} \cdot \tfrac{w_2}{TW(\llbracket CT_2 \rrbracket_{\rm pwd})} \cdot N_{CT_1, CT_2}) \mid \\ & (sprocs_1, w_1) \in \llbracket CT_1 \rrbracket_{\rm pwd} \wedge (sprocs_2, w_2) \in \llbracket CT_2 \rrbracket_{\rm pwd}\ \} \end{aligned}$$

$$N_{CT_1, CT_2} = \begin{cases} TW(\llbracket CT_1 \rrbracket_{\rm pwd}) + TW(\llbracket CT_2 \rrbracket_{\rm pwd}) & \text{if } \llbracket CT_1 \rrbracket_{\rm pri} = \llbracket CT_2 \rrbracket_{\rm pri} \\ TW(\llbracket CT_1 \rrbracket_{\rm pwd}) & \text{if } \llbracket CT_1 \rrbracket_{\rm pri} > \llbracket CT_2 \rrbracket_{\rm pri} \\ TW(\llbracket CT_2 \rrbracket_{\rm pwd}) & \text{if } \llbracket CT_2 \rrbracket_{\rm pri} > \llbracket CT_1 \rrbracket_{\rm pri} \end{cases}$$

$$TW(wd_\Gamma) = \sum\{\!| wd_\Gamma(G) \mid G \in \mathcal{P}(\Gamma) |\!\}$$

Table 11.3: Priority level and process weight distribution of a cooperation tree

We now show that the adoption of the values above for the priority level and the total weight of $CT_1 \wedge CT_2$, which may seem to be somehow arbitrary, leads to a simple way to determine, from a given cooperation tree, a tree with prioritized-probabilistic branches representing the choice of a group of cooperating slave processes. As we will formally show in Sect. 11.8.2, the adoption of such values derives directly from the choice of the particular priority level and weight normalization factor adopted for passive action synchronization in the EMPA_{gr} semantics (see Sect. 7). Therefore this provides also a justification for such a choice in EMPA_{gr}.

It is easy to see that the process weight distribution derived according to the rules of Table 11.3 can be obtained by performing, for each choice node $CT_1 \vee CT_2$ of the cooperation tree, an independent prioritized-probabilistic choice between the lefthand subtree CT_1 and the righthand subtree CT_2. For $CT \in \{CT_1, CT_2\}$, the priority level and the weight associated with the choice of CT can be simply evaluated from the priority levels and the weights of the process activities singled out by the leaves of CT. More precisely, the priority level of CT is the maximum priority level of the process activities adjacent to the leaves of CT [6]

$$l_{CT} = \max\{l \mid \exists \mathcal{S}. <\alpha, *_{l,w}> \in \mathcal{S} \wedge \mathcal{S}\ \mathit{subt}\ CT\}$$

and the weight of CT is the total weight of the activities in the leaves of CT which have priority level l_{CT}

$$w_{CT} = \sum \{\!| \, w \mid \exists \mathcal{S}. <\alpha, *_{l,w}> \in \mathcal{S} \wedge \mathcal{S}\ \mathit{subt}\ CT \wedge l = l_{CT} \, |\!\}$$

In order to show that the rules of Table 11.3 give rise to the simple interpretation above, it is just sufficient to prove that, given a choice node $CT_1 \vee CT_2$, for $CT \in \{CT_1, CT_2\}$ we have $l_{CT} = [\![CT]\!]_{\mathrm{pri}}$ and $w_{CT} = TW([\![CT]\!]_{\mathrm{pwd}})$. This stems from the following theorem.

Theorem 11.1 *For each cooperation tree CT, $l_{CT} = [\![CT]\!]_{\mathrm{pri}}$ and $w_{CT} = TW([\![CT]\!]_{\mathrm{pwd}})$.*

Proof: An easy induction on the syntactical structure of CT, in which the definitions of $[\![CT]\!]_{\mathrm{pri}}$ and $[\![CT]\!]_{\mathrm{pwd}}$ of Table 11.3 are considered. In particular, in the case of a cooperation node the result holds because from the induction hypothesis it follows that

$$\begin{aligned}
&TW([\![CT_1 \wedge CT_2]\!]_{\mathrm{pwd}}) = \\
&\quad = \sum_{(sprocs_1, w_1) \in [\![CT_1]\!]_{\mathrm{pwd}}} \ \sum_{(sprocs_2, w_2) \in [\![CT_2]\!]_{\mathrm{pwd}}} \frac{w_1}{TW([\![CT_1]\!]_{\mathrm{pwd}})} \cdot \frac{w_2}{TW([\![CT_2]\!]_{\mathrm{pwd}})} \cdot N_{CT_1, CT_2} \\
&\quad = N_{CT_1, CT_2} \\
&\quad = w_{CT_1 \wedge CT_2}
\end{aligned}$$

□

Intraprocess Slave Choice

The actual slave activities executed by the winning group $P_1, \ldots, P_n$ of cooperating slave processes (with possibly $n = 1$) are locally determined by every process of the group. Each process P_i in this group independently chooses a slave activity according to a prioritized-probabilistic choice among the slave activities in the set $\mathcal{S}_i$ singled out by the corresponding leaf of the cooperation tree.

Now we define $[\![CT]\!]_{\mathrm{awd}}$, which describes how system activities are probabilistically chosen according to CT, accounting for both the interprocess and the intraprocess slave choice, by associating a weight with every set of cooperating slave activities $\{s_1, \ldots, s_n\}$. $[\![CT]\!]_{\mathrm{awd}}$ is derived from $[\![CT]\!]_{\mathrm{pwd}}$ as follows. In order to determine the weight of $[\![CT]\!]_{\mathrm{awd}}(\{s_1, \ldots, s_n\})$, where s_i is such that its priority level is the maximum priority level of the activities in the leaf $\mathcal{S}_i$ of CT it belongs to, the weight of $[\![CT]\!]_{\mathrm{pwd}}(\{\mathit{proc}(S_1), \ldots, \mathit{proc}(S_n)\})$ is multiplied by the probability that each activity s_i is independently chosen by its process $\mathit{proc}(S_i)$.

[6] We denote by *subt* the subterm relationship, i.e. $CT\ \mathit{subt}\ CT'$ if CT is a subterm of CT'.

Definition 11.11 *The semantic mapping* $[\![\,]\!]_{\rm awd} : \mathit{CTrees} \longrightarrow \mathit{WDist}_{\mathit{SAct}}$ *which associates a weight distribution over groups of cooperating slave activities with a cooperation tree is defined by*

$$\begin{array}{rcl} [\![CT]\!]_{\rm awd} & = & \{ (\{<\alpha_i, *_{l_i,w_i}> \mid 1 \le i \le n\}, w) \mid \exists \mathcal{S}_1, \dots, \mathcal{S}_n \ \mathit{subt}\ CT. \\ & & \quad (\forall i, j \in \{1, \dots, n\}.\, i \ne j \Longrightarrow \mathcal{S}_i \ne \mathcal{S}_j) \land \\ & & \quad (\forall i \in \{1, \dots, n\}.\, <\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \land l_i = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}_i\}) \land \\ & & \quad w = [\![CT]\!]_{\rm pwd}(\{\mathit{proc}(\mathcal{S}_i) \mid 1 \le i \le n\}) \cdot \prod\limits_{1 \le i \le n} \frac{w_i}{\sum\{\!| w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \land l = l_i |\!\}} \} \end{array}$$

■

11.8.2 Consistency with the Cooperation Structure Model

In this section we prove that the synchronization discipline of $\mathrm{EMPA_{gr}}$ is consistent with the cooperation structure model by showing that the calculations of the rates of the synchronizations are consistent with those resulting from the cooperation structures. This is accomplished by introducing the mapping $[\![\,]\!]_{\rm csm} : \mathcal{G} \longrightarrow \mathcal{P}(\mathit{AType} \times (\mathit{CStructs} \cup (\{\perp\} \times \mathit{CTrees}) \cup (\mathit{MAct} \times \{\perp\})))$ defined in Table 11.4. This mapping associates with every term a set of pairs, each composed of an action type and a possibly incomplete cooperation structure. As we shall see, the generated cooperation structures are related to the actions executable by the given term, while the action types occurring in the pairs are simply used to correctly generate the cooperation structures.

As can be noted in the rule for the action prefix operator, a cooperation structure can be incomplete in the sense that either its master or its cooperation tree is absent ($\perp$). In general, during the application of the rules in Table 11.4, the former case happens when several groups of passive actions of the same type have not been synchronized yet with an active action of that type, while the latter case happens when an active action has not been synchronized yet with passive actions of the same type. We observe that the cooperation structures generated for a performance closed term are all endowed with a master. In the table, we use $\overline{m}$ to range over $\mathit{MAct} \cup \{\perp\}$ and $\overline{CT}$ to range over $\mathit{CTrees} \cup \{\perp\}$.

Since the names of master and slave activities must be unique, in each generated cooperation structure such names are taken from the set *Loc* of locations defined by the following syntax:

$$\sigma ::= \bullet \mid +_{\swarrow}\sigma \mid +_{\searrow}\sigma \mid \|_{\swarrow}\sigma \mid \|_{\searrow}\sigma$$

Names are initialized to $\bullet$ in the rule for the action prefix operator and then updated in the rules for the alternative and parallel composition operators. In particular, updating slave activity names is carried out through function $\mathit{Upd} : (\mathit{CTrees} \cup \{\perp\}) \times \mathit{Dir} \longrightarrow \mathit{CTrees} \cup \{\perp\}$ defined in the second part of Table 11.5, where $\mathit{Dir} = \{+_{\swarrow}, +_{\searrow}, \|_{\swarrow}, \|_{\searrow}\}$ is a set of directions ranged over by δ.

As far as the rule for the alternative composition operator is concerned, we observe that two cooperation trees (of the two subterms) must be combined whenever the related action types coincide and the related masters are absent, because such trees represent two alternative groups of slaves which can be subsequently synchronized with the same active action of the given type. The combination is carried out through a choice node if at least one of the two trees is not a leaf, by a single leaf obtained by joining the two original ones otherwise.

$$
\begin{array}{rcl}
[\![\underline{0}]\!]_{\mathrm{csm}} & = & \emptyset \\
[\![<a,\tilde{\lambda}>.E]\!]_{\mathrm{csm}} & = & \left\{ \begin{array}{ll} \{(a, \langle <\bullet, \tilde{\lambda}>, \perp\rangle)\} & \text{if } \tilde{\lambda} \in \mathit{MRate} \\ \{(a, \langle \perp, \{<\bullet, \tilde{\lambda}>\}\rangle)\} & \text{if } \tilde{\lambda} \in \mathit{SRate} \end{array} \right. \\
[\![E/L]\!]_{\mathrm{csm}} & = & \{(a, \langle \overline{m}, \overline{CT}\rangle) \mid (a, \langle \overline{m}, \overline{CT}\rangle) \in [\![E]\!]_{\mathrm{csm}} \wedge a \notin L \cup \{\tau\}\} \cup \\
& & \{(\tau, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \mid \exists a \in L \cup \{\tau\}.\, (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E]\!]_{\mathrm{csm}}\} \cup \\
& & \{(\tau, \langle \perp, \bigvee\{CT \mid \exists (b, \langle \perp, CT\rangle) \in [\![E]\!]_{\mathrm{csm}}.\, b \in L \cup \{\tau\}\}\rangle) \mid \exists a \in L \cup \{\tau\}. \\
& & \qquad \exists CT.\, (a, \langle \perp, CT\rangle) \in [\![E]\!]_{\mathrm{csm}}\} \\
[\![E[\varphi]]\!]_{\mathrm{csm}} & = & \{(\varphi(a), \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \mid (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E]\!]_{\mathrm{csm}}\} \cup \\
& & \{(\varphi(a), \langle \perp, \bigvee\{CT \mid \exists (b, \langle \perp, CT\rangle) \in [\![E]\!]_{\mathrm{csm}}.\, \varphi(b) = \varphi(a)\}\rangle) \mid \\
& & \qquad \exists CT.\, (a, \langle \perp, CT\rangle) \in [\![E]\!]_{\mathrm{csm}}\} \\
[\![E_1 + E_2]\!]_{\mathrm{csm}} & = & \{(a, \langle <+_{\swarrow}\, \sigma, \tilde{\lambda}>, \mathit{Upd}(\overline{CT}, +_{\swarrow})\rangle) \mid (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E_1]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle <+_{\searrow}\, \sigma, \tilde{\lambda}>, \mathit{Upd}(\overline{CT}, +_{\searrow})\rangle) \mid (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_1, +_{\swarrow})\rangle) \mid (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \nexists CT_2.\, (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_2, +_{\searrow})\rangle) \mid (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \nexists CT_1.\, (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_1, +_{\swarrow}) \vee \mathit{Upd}(CT_2, +_{\searrow})\rangle) \mid (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \nexists \mathcal{S}_1, \mathcal{S}_2.\, CT_1 = \mathcal{S}_1 \wedge CT_2 = \mathcal{S}_2\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(\mathcal{S}_1, +_{\swarrow}) \cup \mathit{Upd}(\mathcal{S}_2, +_{\searrow})\rangle) \mid (a, \langle \perp, \mathcal{S}_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad (a, \langle \perp, \mathcal{S}_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \\
[\![E_1 \parallel_S E_2]\!]_{\mathrm{csm}} & = & \{(a, \langle <\parallel_{\swarrow}\, \sigma, \tilde{\lambda}>, \mathit{Upd}(\overline{CT}, \parallel_{\swarrow})\rangle) \mid a \notin S \wedge (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E_1]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle <\parallel_{\searrow}\, \sigma, \tilde{\lambda}>, \mathit{Upd}(\overline{CT}, \parallel_{\searrow})\rangle) \mid a \notin S \wedge (a, \langle <\sigma, \tilde{\lambda}>, \overline{CT}\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_1, \parallel_{\swarrow})\rangle) \mid a \notin S \wedge (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \nexists CT_2.\, (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_2, \parallel_{\searrow})\rangle) \mid a \notin S \wedge (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \nexists CT_1.\, (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \perp, \mathit{Upd}(CT_1, \parallel_{\swarrow}) \vee \mathit{Upd}(CT_2, \parallel_{\searrow})\rangle) \mid a \notin S \wedge \\
& & \qquad (a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad (a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}}\} \cup \\
& & \{(a, \langle \overline{m}, \mathit{Upd}(\overline{CT}_1, \parallel_{\swarrow})\ \Delta\ \mathit{Upd}(\overline{CT}_2, \parallel_{\searrow})\rangle) \mid a \in S \wedge \exists \overline{m}_1, \overline{m}_2.\, \perp \in \{\overline{m}_1, \overline{m}_2\} \wedge \\
& & \qquad (a, \langle \overline{m}_1, \overline{CT}_1\rangle) \in [\![E_1]\!]_{\mathrm{csm}} \wedge \\
& & \qquad (a, \langle \overline{m}_2, \overline{CT}_2\rangle) \in [\![E_2]\!]_{\mathrm{csm}} \wedge \\
& & \qquad \overline{m} = \left\{ \begin{array}{ll} <\parallel_{\swarrow}\, \sigma, \tilde{\lambda}> & \text{if } \overline{m}_1 = <\sigma, \tilde{\lambda}> \\ <\parallel_{\searrow}\, \sigma, \tilde{\lambda}> & \text{if } \overline{m}_2 = <\sigma, \tilde{\lambda}> \\ \perp & \text{if } \overline{m}_1 = \overline{m}_2 = \perp \end{array} \right. \} \\
[\![A]\!]_{\mathrm{csm}} & = & [\![E]\!]_{\mathrm{csm}} \qquad \text{if } A \stackrel{\Delta}{=} E
\end{array}
$$

Table 11.4: Cooperation structure model of an $\mathrm{EMPA}_{\mathrm{gr}}$ term

$\perp \, \Delta \, \perp = \perp$	$CT \, \Delta \, \perp = \perp \, \Delta \, CT = CT$	$CT_1 \, \Delta \, CT_2 = CT_1 \wedge CT_2$

$$\begin{array}{rcl} Upd(\perp, \delta) & = & \perp \\ Upd(\mathcal{S}, \delta) & = & \{<\delta\sigma, *_{l,w}> \mid <\sigma, *_{l,w}> \in \mathcal{S}\} \\ Upd(CT_1 \vee CT_2, \delta) & = & Upd(CT_1, \delta) \vee Upd(CT_2, \delta) \\ Upd(CT_1 \wedge CT_2, \delta) & = & Upd(CT_1, \delta) \wedge Upd(CT_2, \delta) \end{array}$$

Table 11.5: Auxiliary functions

As far as the rule for the parallel composition operator is concerned, we note that two cooperation trees (of the two subterms) must be combined in two cases whenever the related action types coincide. In the former case, the type is not in the synchronization set and both masters are absent, which means that the two trees represent two alternative groups of slaves residing on different sequential processes which are independent of each other (i.e. do not cooperate) w.r.t. the given type. In this case, the two trees are combined through a choice node. In the latter case, instead, the type is in the synchronization set and at least one of the two masters is absent, which means that the two trees represent two groups of slaves residing on different sequential processes which synchronize w.r.t. the given type. In this case, the two trees are combined through a cooperation node. In order to take incomplete cooperation trees into account, the actual combinator we use is $\Delta : (\mathit{CTrees} \cup \{\perp\}) \times (\mathit{CTrees} \cup \{\perp\}) \longrightarrow \mathit{CTrees} \cup \{\perp\}$ defined in the first part of Table 11.5.

Note that, since in the definition of the mapping for a term E only the actions immediately enabled in the state E are considered (i.e., in the rule for the action prefix operator no induction is performed on E) and since the mapping is applied to guarded terms only, we always obtain as a mapping of a term a finite set of finite (possibly incomplete) cooperation structures.

We are now in a position to prove that the synchronization discipline of $\mathrm{EMPA_{gr}}$ is consistent with the cooperation structure model by showing the equality between the rates associated with the actions executable by a given term and the rates derived, according to the formulas presented in Sect. 11.8.1, from the typed (possibly incomplete) cooperation structures generated for that term.

Theorem 11.2 *For all* $E \in \mathcal{G}$

$$
\begin{array}{l}
\{\!|\, <a,\tilde{\lambda}> \in Act \mid \exists E' \in \mathcal{G}.\, E \xrightarrow{a,\tilde{\lambda}} E' \,|\!\} = \\
\qquad \{\!|\, <a,\tilde{\lambda}> \in Act \mid \exists (a, \langle \overline{m}, \overline{CT} \rangle) \in [\![E]\!]_{\rm csm}. \\
\qquad\qquad ((\overline{CT} = \bot \wedge \overline{m} = <\sigma, \tilde{\lambda}>) \vee \\
\qquad\qquad (\overline{CT} \neq \bot \wedge \exists \mathcal{S} \in Dom([\![\overline{CT}]\!]_{\rm awd}). \\
\qquad\qquad\quad \tilde{\lambda} = \begin{cases} \tilde{\mu} \cdot \frac{[\![\overline{CT}]\!]_{\rm awd}(\mathcal{S})}{TW([\![\overline{CT}]\!]_{\rm awd})} & \text{if } \overline{m} = <\sigma, \tilde{\mu}> \\ *_{[\![\overline{CT}]\!]_{\rm pri}, [\![\overline{CT}]\!]_{\rm awd}(\mathcal{S})} & \text{if } \overline{m} = \bot \end{cases})) \,|\!\}
\end{array}
$$

Proof: See Appendix B.1. □

11.9 Markovian Bisimulation Equivalence

In this section we equip $\mathrm{EMPA_{gr}}$ with a Markovian bisimulation equivalence, which relates systems having the same functional, probabilistic, prioritized and exponentially timed behavior. We then show that such an equivalence is a congruence, thus providing support for compositional manipulation, and we give a sound and complete axiomatization for nonrecursive process terms.

Our Markovian bisimulation equivalence is inspired by the *probabilistic bisimulation equivalence* of [117], according to which two equivalent terms must have the same aggregated probability to reach the same equivalence class of terms by executing actions of the same type and priority level.

In the case of exponentially timed actions, we have to take into account not only the transition probabilities but also the state sojourn times. Because of the adoption of the race policy (see Sect. 11.2), this can be easily accomplished by considering the aggregated rate with which an equivalence class is reached by a term by executing actions of the same type. As an example, it must hold that

$$<a,\lambda_1>.E + <a,\lambda_2>.E \sim_{\rm MB} <a,\lambda_1+\lambda_2>.E$$

This treatment of rates, originally proposed in [107], is basically the same as that of the exact aggregation for CTMCs called *ordinary lumping* [147] (see Chapter 2) which preserves performance behavior, thus establishing a clear connection between the Markovian bisimulation equivalence and the ordinary lumping. In the case of immediate and passive actions, instead, the probabilistic bisimulation equivalence must be rephrased in terms of weights. As an example, it must hold that

$$
\begin{array}{rcl}
<a,\infty_{l,w_1}>.E + <a,\infty_{l,w_2}>.E & \sim_{\rm MB} & <a,\infty_{l,w_1+w_2}>.E \\
<a,*_{l,w_1}>.E + <a,*_{l,w_2}>.E & \sim_{\rm MB} & <a,*_{l,w_1+w_2}>.E
\end{array}
$$

This treatment of weights was originally proposed in [150].

We are now in a position of defining our Markovian bisimulation equivalence.

Definition 11.12 *We define function priority level $PL : ARate \longrightarrow \mathbb{Z}$ by:*

$$\begin{aligned} PL(*_{l,w}) &= -l \\ PL(\lambda) &= 0 \\ PL(\infty_{l,w}) &= l \end{aligned}$$

and we extend the real number summation to rates of the same priority level as follows:

$$\begin{aligned} \infty_{l,w_1} + \infty_{l,w_2} &= \infty_{l,w_1+w_2} \\ *_{l,w_1} + *_{l,w_2} &= *_{l,w_1+w_2} \end{aligned}$$

We then define partial function aggregated rate $Rate : \mathcal{G} \times AType \times \mathbb{Z} \times \mathcal{P}(\mathcal{G}) \,\circ\!\!\!\rightarrow ARate$ *by:*

$$Rate(E, a, l, C) = \sum \{\!| \, \tilde{\lambda} \mid \exists E' \in C.\, E \xrightarrow{a,\tilde{\lambda}} E' \wedge PL(\tilde{\lambda}) = l \, |\!\}$$

with $Rate(E, a, l, C) = \bot$ *whenever the multiset above is empty.* ■

Definition 11.13 *An equivalence relation* $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ *is a Markovian bisimulation if and only if, whenever* $(E_1, E_2) \in \mathcal{B}$, *then for all* $a \in AType$, $l \in \mathbb{Z}$, *and equivalence classes* $C \in \mathcal{G}/\mathcal{B}$

$$Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$$

■

Proposition 11.1 *Let* $\sim_{\rm MB}$ *be the union of all the Markovian bisimulations over* $\mathcal{G}$. *Then* $\sim_{\rm MB}$ *is the largest Markovian bisimulation over* $\mathcal{G}$.

Proof: Let us preliminarily demonstrate that $\mathcal{B} = (\cup_{i\in I}\mathcal{B}_i)^+$ is a Markovian bisimulation whenever so is $\mathcal{B}_i$ for all $i \in I$. Once observed that $\mathcal{B}$ is an equivalence relation because it is the transitive closure of the union of equivalence relations, assume that $(E_1, E_2) \in \mathcal{B}$. Since $\mathcal{B} = \cup_{n\in\mathbb{N}_+}\mathcal{B}^{(n)}$ where $\mathcal{B}^{(n)} = (\cup_{i\in I}\mathcal{B}_i)^n$, we have $(E_1, E_2) \in \mathcal{B}^{(n)}$ for some $n \in \mathbb{N}_+$. The result follows by proving by induction on $n \in \mathbb{N}_+$ that, whenever $(E_1, E_2) \in \mathcal{B}^{(n)}$, then $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$ for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}$.

- If $n = 1$ then $(E_1, E_2) \in \mathcal{B}_i$ for some $i \in I$. Let $\mathcal{G}/\mathcal{B}_i = \{C_{i,j} \mid j \in J_i\}$. Since $(E_1, E_2) \in \mathcal{B}_i$ implies $(E_1, E_2) \in \mathcal{B}$, we have that for each $C_{i,j} \in \mathcal{G}/\mathcal{B}_i$ there exists $C \in \mathcal{G}/\mathcal{B}$ such that $C_{i,j} \subseteq C$, so each equivalence class of $\mathcal{B}$ can be written as the union of a set of equivalence classes of $\mathcal{B}_i$. As a consequence, for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}$, if $C = \cup_{j\in J'_i} C_{i,j}$ where $J'_i \subseteq J_i$, then $Rate(E_1, a, l, C) = \sum\{\!| \, Rate(E_1, a, l, C_{i,j}) \mid j \in J'_i \, |\!\} = \sum\{\!| \, Rate(E_2, a, l, C_{i,j}) \mid j \in J'_i \, |\!\} = Rate(E_2, a, l, C)$ because $\mathcal{B}_i$ is a Markovian bisimulation.

- If $n > 1$ from $(E_1, E_2) \in \mathcal{B}^{(n)}$ we derive that there exists $F \in \mathcal{G}$ such that $(E_1, F) \in \mathcal{B}^{(n-1)}$ and $(F, E_2) \in \mathcal{B}_i$ for some $i \in I$. Thus for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}$, it turns out $Rate(E_1, a, l, C) = Rate(F, a, l, C)$ by the induction hypothesis and $Rate(F, a, l, C) = Rate(E_2, a, l, C)$ by applying the same argument as the previous point.

Now, let us turn to the result we have to prove. By definition, $\sim_{\rm MB}$ contains the largest Markovian bisimulation. If we prove that $\sim_{\rm MB}$ is a Markovian bisimulation, then we are done. Since $\sim_{\rm MB} \subseteq \sim_{\rm MB}^+$

trivially holds and $\sim^+_{\rm MB} \subseteq \sim_{\rm MB}$ is due to the fact that $\sim^+_{\rm MB}$ is a Markovian bisimulation, by virtue of what is demonstrated above, and that $\sim_{\rm MB}$ contains all the Markovian bisimulations by definition, we have $\sim_{\rm MB} = \sim^+_{\rm MB}$. Again $\sim^+_{\rm MB}$ is a Markovian bisimulation because of what is demonstrated above, hence so is $\sim_{\rm MB}$. □

Definition 11.14 *We call $\sim_{\rm MB}$ the Markovian bisimulation equivalence.* ■

In the following we define the notion of Markovian bisimulation up to $\sim_{\rm MB}$, an extended version of the bisimulation up to bisimulation equivalence of [122], which is helpful to avoid redundancy when constructing Markovian bisimulations. For example, suppose that we want to show that two terms E_1 and E_2 are Markovian bisimulation equivalent by constructing a Markovian bisimulation $\mathcal{B}$ such that $(E_1, E_2) \in \mathcal{B}$. Moreover suppose that in order to do that we need to include both pairs $(F_1 \,\|_S\, F_2, F_3)$ and $(F_2 \,\|_S\, F_1, F_3)$ in $\mathcal{B}$. Since $(F_2 \,\|_S\, F_1, F_3)$ trivially derives from $(F_1 \,\|_S\, F_2, F_3)$ because $F_1 \,\|_S\, F_2 \sim_{\rm MB} F_2 \,\|_S\, F_1$, we can just avoid including the redundant $(F_2 \,\|_S\, F_1, F_3)$ by constructing a Markovian bisimulation up to $\sim_{\rm MB}$ that includes $(F_1 \,\|_S\, F_2, F_3)$ only. As we will see, finding a Markovian bisimulation up to $\sim_{\rm MB}$ including the pair (E_1, E_2) is a sufficient condition in order to show that E_1 and E_2 are Markovian bisimulation equivalent.

Definition 11.15 *A relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a Markovian bisimulation up to $\sim_{\rm MB}$ if and only if, whenever $(E_1, E_2) \in \mathcal{B}$, then for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B} \,\cup\, \mathcal{B}^{-1} \cup \sim_{\rm MB})^+$*

$$Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$$

■

Our definition of Markovian bisimulation up to $\sim_{\rm MB}$ improves those previously given for other Markovian process algebras in that it fixes some inaccuracies present in the definition of [107, 27] and further eliminates redundancy in Markovian bisimulations w.r.t. the definition of [99]. The problem with the definition of [107, 27] is that, instead of the relation $(\mathcal{B} \cup \mathcal{B}^{-1} \cup \sim_{\rm MB})^+$, it considers $\sim_{\rm MB} \mathcal{B} \sim_{\rm MB}$ which in general, as we will show, is not an equivalence relation even if $\mathcal{B}$ is supposed to be an equivalence relation. For instance, if $\mathcal{B}$ is defined as follows:

$$\mathcal{B} = Id_{\mathcal{G}} \cup \{(<a,\lambda>.\underline{0}, \underline{0}), (\underline{0}, <a,\lambda>.\underline{0})\} \cup \{(<b,\mu>.\underline{0}, \underline{0} \,\|_\emptyset\, \underline{0}), (\underline{0} \,\|_\emptyset\, \underline{0}, <b,\mu>.\underline{0})\}$$

where $Id_{\mathcal{G}}$ is the identity relation over $\mathcal{G}$, then $\mathcal{B}$ is an equivalence relation but $\sim_{\rm MB} \mathcal{B} \sim_{\rm MB}$ is not transitive. This can be shown as follows. It holds that:

$$<a,\lambda>.\underline{0} \ \sim_{\rm MB}\ <a,\lambda>.\underline{0} \ \mathcal{B}\ \underline{0} \ \sim_{\rm MB}\ \underline{0} \,\|_\emptyset\, \underline{0}$$

$$\underline{0} \,\|_\emptyset\, \underline{0} \ \sim_{\rm MB}\ \underline{0} \,\|_\emptyset\, \underline{0} \ \mathcal{B}\ <b,\mu>.\underline{0} \ \sim_{\rm MB}\ <b,\mu>.\underline{0}$$

but it is not the case that:

$$<a,\lambda>.\underline{0} \ \sim_{\rm MB}\ \mathcal{B} \sim_{\rm MB}\ <b,\mu>.\underline{0}$$

It is easy to see that the relation $(\mathcal{B} \cup \mathcal{B}^{-1} \cup \sim_{\mathrm{MB}})^+$ that we consider is indeed an equivalence relation for any given relation $\mathcal{B}$. In particular $Id_{\mathcal{G}} \subseteq (\mathcal{B} \cup \mathcal{B}^{-1} \cup \sim_{\mathrm{MB}})^+$ because $Id_{\mathcal{G}} \subseteq \sim_{\mathrm{MB}}$.

As for the definition of [99], where $\mathcal{B}$ is required to be a symmetric relation and the relation $(\mathcal{B} \cup \sim_{\mathrm{MB}})^+$ is considered, our definition further eliminates redundancy in that we do not have to include symmetric pairs in a Markovian bisimulation up to $\sim_{\mathrm{MB}}$.

The following proposition shows that the new definition of bisimulation up to can indeed be used to prove equivalence of terms.

Proposition 11.2 *If $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a Markovian bisimulation up to $\sim_{\mathrm{MB}}$ and $(E_1, E_2) \in \mathcal{B}$, then $E_1 \sim_{\mathrm{MB}} E_2$.*
Proof: We first prove that $(\mathcal{B} \cup \mathcal{B}^{-1} \cup \sim_{\mathrm{MB}})^+$ is a Markovian bisimulation. Let $\mathcal{B}' = \mathcal{B} \cup \mathcal{B}^{-1}$. By proceeding by induction on $n \in \mathbb{N}_+$ we show that, whenever $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{MB}})^n$, then for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$ we have that $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$.

- If $n = 1$, then $(E_1, E_2) \in \mathcal{B}' \cup \sim_{\mathrm{MB}}$. If $(E_1, E_2) \in \mathcal{B}'$ then the result trivially follows from the fact that $\mathcal{B}$ is a Markovian bisimulation up to $\sim_{\mathrm{MB}}$. If $(E_1, E_2) \in \sim_{\mathrm{MB}}$ we observe that, since $\sim_{\mathrm{MB}} \subseteq (\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$, each equivalence class of $(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$ can be written as the union of some equivalence classes of $\sim_{\mathrm{MB}}$. As a consequence, for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$, it turns out $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$, because $E_1 \sim_{\mathrm{MB}} E_2$.
- Let $n > 1$. From $(E_1, E_2) \in (\mathcal{B}' \cup \sim_{\mathrm{MB}})^n$ we derive that there exists $F \in \mathcal{G}$ such that $(E_1, F) \in (\mathcal{B}' \cup \sim_{\mathrm{MB}})^{n-1}$ and $(F, E_2) \in \mathcal{B}' \cup \sim_{\mathrm{MB}}$. Thus for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$, it turns out $Rate(E_1, a, l, C) = Rate(F, a, l, C)$ by the induction hypothesis, and $Rate(F, a, l, C) = Rate(E_2, a, l, C)$ by applying the same argument as the previous point.

To complete the proof, we observe that $\mathcal{B} \subseteq (\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$, and $(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+ \subseteq \sim_{\mathrm{MB}}$ because $(\mathcal{B}' \cup \sim_{\mathrm{MB}})^+$ is a Markovian bisimulation, hence $\mathcal{B} \subseteq \sim_{\mathrm{MB}}$ by transitivity. $\square$

In the following we shall show that $\sim_{\mathrm{MB}}$ is a congruence for all the operators of EMPA$_{\mathrm{gr}}$ and for recursive definitions.

Theorem 11.3 *Let $E_1, E_2 \in \mathcal{G}$. If $E_1 \sim_{\mathrm{MB}} E_2$ then:*

1. *For all $<a, \tilde{\lambda}> \in Act$, $<a, \tilde{\lambda}>.E_1 \sim_{\mathrm{MB}} <a, \tilde{\lambda}>.E_2$.*
2. *For all $L \subseteq AType - \{\tau\}$, $E_1/L \sim_{\mathrm{MB}} E_2/L$.*
3. *For all $\varphi \in ATRFun$, $E_1[\varphi] \sim_{\mathrm{MB}} E_2[\varphi]$.*
4. *For all $F \in \mathcal{G}$, $E_1 + F \sim_{\mathrm{MB}} E_2 + F$ and $F + E_1 \sim_{\mathrm{MB}} F + E_2$.*
5. *For all $F \in \mathcal{G}$ and $S \subseteq AType - \{\tau\}$, $E_1 \parallel_S F \sim_{\mathrm{MB}} E_2 \parallel_S F$ and $F \parallel_S E_1 \sim_{\mathrm{MB}} F \parallel_S E_2$.*

Proof: See Appendix B.2. □

We observe that the congruence result above holds because the operational semantics is defined in such a way that lower priority active transitions are not pruned. If this were not the case, we would have e.g. $<a_1, \lambda>.\underline{0} + <a_2, \infty_{l,w}>.\underline{0} \sim_{\rm MB} <a_2, \infty_{l,w}>.\underline{0}$ as both terms would have only one transition labeled with $<a_2, \infty_{l,w}>$, but $(<a_1, \lambda>.\underline{0} + <a_2, \infty_{l,w}>.\underline{0}) \,\|_{\{a_2\}}\, \underline{0} \not\sim_{\rm MB} <a_2, \infty_{l,w}>.\underline{0} \,\|_{\{a_2\}}\, \underline{0}$ because the first term has a transition labeled with action $<a_1, \lambda>$ while the second term has no transitions at all. On the contrary, the removal of lower priority passive actions of a given type does not cause any problem.

Theorem 11.4 *$\sim_{\rm MB}$ is a congruence w.r.t. recursive constant definitions.*

Proof: See Appendix B.2. □

The proof in Appendix B.2 shows that $\sim_{\rm MB}$ extended to open terms is a congruence w.r.t. recursive definitions by introducing a new proof technique, which leads to a smooth extension of the Milner's technique based on bisimulation up to bisimulation equivalence [122] and solves some problems in the proofs of the same result previosly proposed for Markovian process algebras [107, 27]. According to the Milner's approach, a particular relation $\mathcal{B}$ is introduced such that if we are able to prove that $\mathcal{B}$ is a Markovian bisimulation up to $\sim_{\rm MB}$, then we can conclude that recursion satisfy the congruence property. In [107, 27] such a $\mathcal{B}$ is proven to be a Markovian bisimulation up to $\sim_{\rm MB}$ as follows. Given two terms $(E_1, E_2) \in \mathcal{B}$, an action type a, a priority level l, and an equivalence class C, the proofs show by induction on the maximum depth of the derivation of the transitions of E_1 labeled with a, with priority level l, and reaching C that $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$. [7] In the inductive step of the proof several cases arise depending on the outermost operator op of E_1 and E_2 as in Milner's proof. As far as static operators are concerned (i.e. functional abstraction, functional relabeling, and parallel composition), the proofs in [107, 27] wrongly assume that all the terms that, when applied the static operator op, belong to C are equivalent, i.e. they form a single equivalence class. As an example, for op = "$\_/L$" we have that in general $\{E \mid E/L \in C\}$ is not an equivalence class. This can be seen by considering the terms $(<a, \lambda>.\underline{0})/\{a, b\}$ and $(<b, \mu>.\underline{0})/\{a, b\}$: they obviously belong to the same equivalence class C, but $<a, \lambda>.\underline{0} \not\sim_{\rm MB} <b, \mu>.\underline{0}$. Therefore, we cannot apply the induction hypothesis to conclude that $E_1 \equiv E_1'/L$ and $E_2 \equiv E_2'/L$ have the same aggregated rate to reach C when performing transitions of type a and priority level l. In order to solve such a problem, we observe that the terms that belong to C when applied the operator op form in general several equivalence classes C_i, $i \in I$. As an example, for op = "$\_/L$" we have that $\{E \mid E/L \in C\} = \cup_{i \in I}\, C_i$. Therefore we should be able to apply the induction hypothesis to each class C_i. Unfortunately, this can be done only for the sets C_i that are actually reachable from E_1' via transitions of type a and priority level l (the maximum depth of such transitions is certainly less than the depth of transitions from E_1 to C). For the other sets C_i we would need a converse argument related to the transitions of E_2'. The simplest solution is

[7] To be precise in [107], where only exponential actions and weighted passive actions without priority levels are considered, the metavariable l may just assume two values, one for referring to exponential actions and one for referring to passive actions.

$$
\begin{array}{llcll}
(\mathcal{A}_1) & (E_1 + E_2) + E_3 & = & E_1 + (E_2 + E_3) & \\
(\mathcal{A}_2) & E_1 + E_2 & = & E_2 + E_1 & \\
(\mathcal{A}_3) & E + \underline{0} & = & E & \\
(\mathcal{A}_4) & <a, \tilde{\lambda}_1>.E + <a, \tilde{\lambda}_2>.E & = & <a, \tilde{\lambda}_1 + \tilde{\lambda}_2>.E & \text{if } PL(\tilde{\lambda}_1) = PL(\tilde{\lambda}_2) \\
(\mathcal{A}_5) & <a, *_{l_1,w_1}>.E_1 + <a, *_{l_2,w_2}>.E_2 & = & <a, *_{l_1,w_1}>.E_1 & \text{if } l_1 > l_2 \\
\\
(\mathcal{A}_6) & \underline{0}/L & = & \underline{0} & \\
(\mathcal{A}_7) & (<a, \tilde{\lambda}>.E)/L & = & <a, \tilde{\lambda}>.(E/L) & \text{if } a \notin L \\
(\mathcal{A}_8) & (<a, \tilde{\lambda}>.E)/L & = & <\tau, \tilde{\lambda}>.(E/L) & \text{if } a \in L \\
(\mathcal{A}_9) & (E_1 + E_2)/L & = & E_1/L + E_2/L & \\
\\
(\mathcal{A}_{10}) & \underline{0}[\varphi] & = & \underline{0} & \\
(\mathcal{A}_{11}) & (<a, \tilde{\lambda}>.E)[\varphi] & = & <\varphi(a), \tilde{\lambda}>.(E[\varphi]) & \\
(\mathcal{A}_{12}) & (E_1 + E_2)[\varphi] & = & E_1[\varphi] + E_2[\varphi] &
\end{array}
$$

$$
\begin{array}{ll}
(\mathcal{A}_{13}) & \sum\limits_{i \in I_0} <a_i, \tilde{\lambda}_i>.E_i \parallel_S \sum\limits_{i \in I_1} <a_i, \tilde{\lambda}_i>.E_i \; = \\
& \quad \sum\limits_{j \in I_0, a_j \notin S} <a_j, \tilde{\lambda}_j>.(E_j \parallel_S \sum\limits_{i \in I_1} <a_i, \tilde{\lambda}_i>.E_i) \; + \\
& \quad \sum\limits_{j \in I_1, a_j \notin S} <a_j, \tilde{\lambda}_j>.(\sum\limits_{i \in I_0} <a_i, \tilde{\lambda}_i>.E_i \parallel_S E_j) \; + \\
& \quad \sum\limits_{k \in K_0} \sum\limits_{h \in P_{1,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h / W_{1,a_k})>.(E_k \parallel_S E_h) \; + \\
& \quad \sum\limits_{k \in K_1} \sum\limits_{h \in P_{0,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h / W_{0,a_k})>.(E_h \parallel_S E_k) \; + \\
& \quad \sum\limits_{k \in P'_0} \sum\limits_{h \in P_{1,a_k}} <a_k, *_{\max(l_k, l_h), (w_k / W_{0,a_k}) \cdot (w_h / W_{1,a_k}) \cdot N_{a_k}}>.(E_k \parallel_S E_h)
\end{array}
$$

where $I_0 \cap I_1 = \emptyset$, $\tilde{\lambda}_i = *_{l_i,w_i}$ for $i \in I_0 \cup I_1 . \, PL(\tilde{\lambda}_i) < 0$, and for $j \in \{0, 1\}$

$$
\begin{array}{rcl}
L_{j,a} & = & \max\{l_k \mid k \in I_j \wedge a_k = a \wedge \tilde{\lambda}_k = *_{l_k,w_k}\} \\
P_{j,a} & = & \{k \in I_j \mid a_k = a \wedge \tilde{\lambda}_k = *_{l_k,w_k} \wedge l_k = L_{j,a}\} \\
K_j & = & \{k \in I_j \mid a_k \in S \wedge PL(\tilde{\lambda}_k) \geq 0 \wedge P_{1-j,a_k} \neq \emptyset\} \\
P'_0 & = & \{k \in I_0 \mid \exists a \in S. \, k \in P_{0,a} \wedge P_{1,a} \neq \emptyset\} \\
W_{j,a} & = & \sum \{\!| \, w_k \mid k \in P_{j,a} \wedge \tilde{\lambda}_k = *_{l_k,w_k} \, |\!\} \\
N_a & = & \begin{cases} W_{0,a} + W_{1,a} & \text{if } L_{0,a} = L_{1,a} \\ W_{0,a} & \text{if } L_{0,a} > L_{1,a} \\ W_{1,a} & \text{if } L_{1,a} > L_{0,a} \end{cases}
\end{array}
$$

Table 11.6: Axiomatization of $\sim_{\text{MB}}$

to split the proof into two symmetric parts and to change the induction assertion of the whole proof into $Rate(E_1, a, l, C) \leq Rate(E_2, a, l, C)$ similarly to [80], where however the structure of Milner's proof based on bisimulation up to bisimulation equivalence is not followed. In this way, the induction hypothesis can be exploited because the aggregated rate to an unreachable class C_i is null, hence less than or equal to any possible rate. Proving $Rate(E_1, a, l, C) \leq Rate(E_2, a, l, C)$ is enough because the reverse can be shown by a symmetric argument. In this way, we follow smoothly the scheme of Milner's proof [122] which is similarly divided into two symmetric parts: in the former E_1 is assumed to have an outgoing transition and the proof proceeds by induction on the depth of the derivation of such a transition, in the latter we do the same for E_2. On the other hand, a symmetric argument should also be used if we would have been able to induce on $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$ because this assertion is shown only in the case of E_1 having an outgoing transition labeled with a, with priority level l, and reaching C, so using $\leq$ leads to a very elegant solution of the problem.

Finally, we present an axiom system for $\sim_{\rm MB}$ which is sound and complete for the nonrecursive process terms of $\mathrm{EMPA_{gr}}$.

Theorem 11.5 *Let $\mathcal{A}$ be the set of axioms in Table 11.6. The deductive system $Ded(\mathcal{A})$ is sound and complete for $\sim_{\rm MB}$ over the set of nonrecursive terms of $\mathcal{G}$.*

Proof: See Appendix B.3. $\square$

We conclude by observing that axiom $(\mathcal{A}_4)$ is exactly the rule we wanted $\sim_{\rm MB}$ to satisfy, while axiom $(\mathcal{A}_5)$ establishes that lower priority passive actions of a given type can be left out.

11.10 Conclusion

The experience with process algebras has shown the necessity of mechanisms like priority, probabilistic internal/external choice, and time to model the behavior of real systems, as well as the necessity of compositionality for efficient system analysis. In this chapter we have developed a new process algebra called $\mathrm{EMPA_{gr}}$ that has a considerable expressive power, because it includes all the above mentioned mechanisms, and achieves semantic compositionality thanks to a suitable asymmetric synchronization mechanism, because Markovian bisimulation equivalence turns out to be a congruence for $\mathrm{EMPA_{gr}}$.

The starting point for our work has been the process algebra EMPA [19]. However, it is worth observing that the language introduced in this chapter does not only provide an improvement over EMPA, for which Markovian bisimulation equivalence is not a congruence w.r.t. the parallel composition operator. In fact, the main contribution of this chapter is the proposal of a natural and intuitive solution to an important open problem in the literature, which is independent of the considered formalism: how to obtain semantic compositionality w.r.t. an asynchronous [8] CSP like parallel composition operator in the presence of priority, probabilistic internal/external choice, and time.

The solution proposed in this chapter essentially consists of an extension to priorities and time of the generative-reactive approach introduced in Chapter 10 (which, in turn, realizes an integration of an asymmetric form of cooperation inspired by probabilistic I/O automata [155] with an approach to interprocess selection inspired by probabilistic ACP [13]). More precisely, the aforementioned extension is threefold:

- The probabilistic advancing speed of a process, which in $\mathrm{EMPA}_{\mathrm{gr}}$ is proportional to the total weight of the immediate actions it can execute, can vary for each process term from state to state, thereby allowing for a greater flexibility.
- We have extended the generative-reactive approach of Chapter 10 to deal with priorities. This has been accomplished by endowing master actions with generative priorities in addition to generative weights, and slave actions with reactive priorities in addition to reactive weights.
- We have extended the generative-reactive approach of Chapter 10 to a timed setting, where a master action may have either an exponentially distributed duration or a zero duration (in which case it is given a priority and a weight), while slave actions have unspecified duration (and are given reactive priorities and weights). This extension encompasses a synchronization mechanism similar to that of PEPA [107], where master-slaves cooperations are realized through synchronizations between exponentially timed actions and weighted passive actions.

We point out that the solution proposed in this chapter constitutes also an extension of [155] to discrete time systems. In fact, in [155] an exponentially distributed delay is associated with every state of each of the probabilistic I/O automata composed in parallel to form a system, in order to have a probabilistic means to decide which automata moves next. Our approach, instead, scales to discrete time systems because, in the case of several process terms composed in parallel which can execute only immediate actions, each process term is given a probabilistic advancing speed (which is proportional to the total weight of the immediate actions it can execute), without resorting to any exponentially distributed delay.

If we compare $\mathrm{EMPA}_{\mathrm{gr}}$ with the other Markovian process algebras appeared in the literature, we observe that none of them reaches the expressive power of $\mathrm{EMPA}_{\mathrm{gr}}$. In particular, the expressiveness of M-TIPP [104] and MPA [51] is limited because they consider exponentially timed actions only. Passive actions with reactive weights are instead introduced in PEPA [107], thus allowing for master-slaves synchronizations, but immediate actions are not included. This may reduce the capability of modeling real systems and does not allow discrete time systems to be described. On the contrary, in IM-TIPP [105] and IMC [99] immediate actions can be expressed, but neither generative nor reactive probabilistic choices can be represented. It is

[8] In the sense that the computation does not proceed in locksteps; e.g., in $E_1 \,\|_\emptyset\, E_2$ we have to choose whether the next move is made by E_1 or E_2.

however worth noting that in the approach of [105, 99] the fact that immediate actions do not bear probabilistic information makes it easy and natural the definition of a weak Markovian bisimulation congruence which abstracts from internal immediate actions (similar to that presented in Chapter 4). Such an approach is further extended in PM-TIPP [141] with reactive (but not generative) probabilistic choices, thus reaching the same expressivity as the calculus of IWMCs introduced in Chapter 4. This capability, however, is not exploited to enforce a master-slaves synchronization policy like in PEPA [107]. We conclude by observing that, unlike $\mathrm{EMPA_{gr}}$, none of the previously considered Markovian process algebras handles (generative or reactive) priorities.

Chapter 12

Expressing Performance Measures Through Rewards

The purpose of this chapter is to introduce a way to express performance measures in $EMPA_{\mathrm{gr}}$, in order to allow the modeler to capture the QoS metrics of interest. In particular we show that the standard technique of expressing stationary and transient performance measures as weighted sums of state probabilities and transition frequencies through rewards can be imported in the process algebra framework without loosing compositionality.

Technically speaking, if we denote by $n \in \mathbb{N}$ the number of performance measures of interest, in this chapter we define a family of extended Markovian process algebras with generative master-reactive slaves synchronization mechanism called $\mathrm{EMPA}_{\mathrm{gr}_n}$ including probabilities, priorities, exponentially distributed durations, and sequences of yield and bonus rewards of length n. Then, we introduce three different extensions of Markovian bisimulation equivalence $\sim_{\mathrm{MB}_n}$ accounting for yield and bonus rewards, we show that they are all congruences for $\mathrm{EMPA}_{\mathrm{gr}_n}$ which preserve the specified performance measures, and, for each of them, we give a sound and complete axiomatization over finite $\mathrm{EMPA}_{\mathrm{gr}_n}$ terms.

12.1 Introduction

In Chapter 11 we showed how to introduce expressive features like probabilities, priorities, and exponentially distributed durations in process algebras while preserving compositionality. Nevertheless, in order to to be able to really analyze real world systems, it is essential to have a simple way of specifying performance measures to be analyzed. Moreover it is important to define measure-sensible congruences that can be exploited to compositionally minimize the state space while preserving the specified performance measures.

In this chapter, starting from $\mathrm{EMPA}_{\mathrm{gr}}$, we make a further step in the direction of expressivity by introducing a way to describe performance measures, in order to allow the modeler to capture the QoS metrics of interest. We achieve this by showing that the standard technique of expressing stationary and transient performance measures as weighted sums of state probabilities and transition frequencies can be imported in $\mathrm{EMPA}_{\mathrm{gr}}$. This is done by following the approach introduced in [19] where the action format is extended to include yield and bonus rewards (see Chapter 2). In particular we extend actions with sequences of $n \in \mathbb{N}$ yield and bonus rewards, thus allowing the specification of several instant-of-time performance measures (Sect. 12.2).

After introducing our extended Markovian process algebra with generative-reactive synchronizations and rewards, called $\text{EMPA}_{\text{gr}_n}$, we formalize its syntax and we define its operational semantics as a mapping from terms to reward master-slaves transition systems of order n, an extension of master-slaves transition systems presented in Chapter 11 (Sect. 12.3).

We subsequently face the problem of defining a notion of equivalence in the bisimulation style, which equates $\text{EMPA}_{\text{gr}_n}$ terms possessing the same functional, probabilistic, prioritized and exponentially timed behavior as well as the same performance measure values. We solve this problem through successive conceptual improvements, by subsequently defining three notions of equivalence satisfying the requirement above, each coarser then the previous one. First, we define an equivalence along the lines of that defined in [19] which, whenever possible, aggregates yield rewards, but does not manipulate bonus rewards. Then, we show how to define a Markovian bisimulation congruence that aggregates bonus rewards as well. This is achieved by taking into account in the aggregation process the execution probability of the transitions to which the bonus rewards are attached. Finally, we show how to define a Markovian bisimulation congruence that allows yield rewards and bonus rewards to be used interchangeably up to suitable correcting factors, aiming at the introduction of a normal form for rewards. More precisely, we demonstrate that this is possible in the continuous time case, while it is not possible in the discrete time case because compositionality is lost. We show that all the equivalences above are congruences, thus providing support for compositional manipulation while preserving the values of the specified performance measures, and, for each of them, we give a sound and complete axiomatization for nonrecursive process terms (Sect. 12.4).

The analysis of performance measures with $\text{EMPA}_{\text{gr}_n}$ has been fully automatized in TwoTowers [26, 19], a software tool for modeling and analyzing functional and performance properties of computer, communication and software systems described in $\text{EMPA}_{\text{gr}_n}$. Here we present an overview of TwoTowers (Sect. 12.5).

The chapter concludes with a discussion about related work (Sect. 12.6).

12.2 Extending EMPA_{gr} with Rewards

In the performance evaluation area the technique of rewards (see Chapter 2) is frequently used to specify and derive measures for system models whose underlying stochastic process is a Markov chain. When using a formal description technique to represent the performance aspects of a system, the stochastic process associated with the underlying performance model is not directly provided by the modeler but automatically derived from the more abstract formal description of the system in order to ease the task of the modeler. As a consequence, rewards should not be defined at the level of the stochastic process but at the level of the formal description, and then automatically inherited by the stochastic process.

This is exactly what happens for well known and tool supported extensions of the Petri net formalism such as reward generalized stochastic Petri nets [56] and stochastic activity networks with rewards [140]. In both cases, yield rewards (also called rate rewards) are naturally associated with net markings, while bonus rewards (also called impulse rewards) are naturally associated with net transitions/activities.

The method that we use in this chapter for specifying instant-of-time performance measures for process algebras (originally proposed in [19]) consists of attaching sequences of pairs of the form (yield_reward, bonus_reward) to process algebra actions, where rewards are unspecified in the case of passive actions. We use "$*$" to denote an unspecified (yield or bonus) reward. As far as yield rewards are concerned, we assume that the yield reward earned by a state is the sum of the yield rewards of the actions it can execute *(additivity assumption)*. Since rewards are specified in the process algebraic description, we call the proposed method *algebra based*. We now assess its adequacy w.r.t. the following criteria: expressive power, ease of use, computational cost, and equational characterization.

As far as the first two criteria (expressive power and ease of use) are concerned, we observe that the algebra based method achieves a reasonable balance in that it allows many of the more frequent performance measures to be specified in a relatively easy way, which in particular does not require the knowledge of any extra formalism to describe reward structures. As an example, we show how to specify for a QS $M/M/n/n$ with arrival rate λ and service rate μ several stationary performance measures frequently occurring in practice such as those identified in [57]: rate type (e.g. throughput of a service center), counting type (e.g. mean number of customers waiting in a service center), delay type (e.g. mean response time experienced by customers in a service center), and percentage type (e.g. utilization of a service center). The QS at hand can be given two different descriptions: a state oriented description, where the focus is on the state of the set of servers (intended as the number of servers that are currently busy), and a resource oriented description, where the servers (i.e. the resources) are modeled separately [151]. With EMPA$_{\rm gr}$ the state oriented description is given by:

$$\begin{array}{rcl}
QS^{so}_{M/M/n/n} & \stackrel{\Delta}{=} & Arrivals \,\|_{\{a\}}\, Servers_0 \\
Arrivals & \stackrel{\Delta}{=} & <a, \lambda>.Arrivals \\
Servers_0 & \stackrel{\Delta}{=} & <a, *>.Servers_1 \\
Servers_h & \stackrel{\Delta}{=} & <a, *>.Servers_{h+1} + \\
 & & <s, h \cdot \mu>.Servers_{h-1}, \;\; 1 \leq h \leq n-1 \\
Servers_n & \stackrel{\Delta}{=} & <s, n \cdot \mu>.Servers_{n-1}
\end{array}$$

whereas the resource oriented description is given by:

$$\begin{array}{rcl}
QS^{ro}_{M/M/n/n} & \stackrel{\Delta}{=} & Arrivals \,\|_{\{a\}}\, Servers \\
Arrivals & \stackrel{\Delta}{=} & <a, \lambda>.Arrivals \\
Servers & \stackrel{\Delta}{=} & \underbrace{S \,\|_{\emptyset}\, S \,\|_{\emptyset} \ldots \|_{\emptyset}\, S}_{n} \\
S & \stackrel{\Delta}{=} & <a, *>.<s, \mu>.S
\end{array}$$

where a stands for arrival of a customer and s stands for service of a customer. These two different descriptions represent the same system, as it can easily be shown that they are Markovian bisimulation equivalent (see Sect. 12.4).

Let us compute for the QS above the mean number of customers in the system, which is the sum of the numbers of customers over all the states with each number weighted by the stationary probability of the corresponding state. According to formula (2.2) of Sect. 2.2.8, every state of the CTMC underlying each of the two terms above must then be given a yield reward equal to the number of customers in that state. Such a number is the number of s actions executable in that state. Therefore, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, (h, 0)>$, while in the case of $QS^{ro}_{M/M/n/n}$ every action of the form $<s, \mu>$ must be replaced with $<s, \mu, (1, 0)>$ by virtue of the additivity assumption for yield rewards. All the other actions must be given zero or unspecified rewards "$*$". More precisely, unspecified rewards must be assigned to all and only the passive actions; in case of synchronization of an active action and a passive action, the resulting action essentially inherits the rewards of the original active action.

If we want to compute the throughput of the QS, defined as the mean number of customers served per time unit, we have to take into account the rate of actions having type s. In fact, the throughput is given by the service rate multiplied by the stationary probability of being in a state where service can be provided. As a consequence, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, (h \cdot \mu, 0)>$ or equivalently $<s, h \cdot \mu, (0, 1)>$, while in the case of $QS^{ro}_{M/M/n/n}$ we must replace every action of the form $<s, \mu>$ with $<s, \mu, (\mu, 0)>$ or equivalently $<s, \mu, (0, 1)>$. [1]

If we want to compute instead the mean response time of the QS, defined as the mean time spent by one customer in the service center, we can exploit Little's law [115] which states that the mean response time experienced by a customer is equal to the mean number of customers in the service center divided by the customer arrival rate. Therefore, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, (h/\lambda, 0)>$, while in the case of $QS^{ro}_{M/M/n/n}$ we must replace every action of the form $<s, \mu>$ with $<s, \mu, (1/\lambda, 0)>$.

Finally, if we want to compute the utilization of the QS, defined as the fraction of time during which servers are busy, we have to single out those states having an outgoing transition labeled with s, because the utilization is the sum of the stationary probabilities of such states. Thus, in the case of $QS^{so}_{M/M/n/n}$ we must replace every action of the form $<s, h \cdot \mu>$ with $<s, h \cdot \mu, (1, 0)>$. However, in the case of $QS^{ro}_{M/M/n/n}$ the algebra based method fails to determine the utilization due to the additivity assumption: the yield reward to associate with actions of the form $<s, \mu>$ would be the reciprocal of the number of transitions labeled with s leaving the same state. Since one of the two main objectives of the algebra based method is its ease of use, we prefer to keep the specification of rewards as simple as possible, i.e. just by means of numbers. Thus we avoid the introduction of arithmetical expressions involving particular functions such as the one

[1] In the continuous time case, yield rewards and bonus rewards can be used interchangeably.

determining the number of transitions of a given type leaving the same state. Incidentally, the inability to compute the utilization in the case of the resource oriented description should not come as a surprise, since this description is more suited to the determination of performance indices concerning a single server instead of the whole set of servers. As it turns out, it is quite easy to measure the utilization of a given server specified in $QS^{ro}_{M/M/n/n}$, whereas this is not possible for $QS^{so}_{M/M/n/n}$. This means that the style [151] used to describe a given system through an algebraic term is strongly related to the possibility of specifying certain performance measures through the algebra based method.

For the considered QS, the algebra based method also allows transient measures to be expressed according to formula (2.3) of Sect. 2.2.8. As an example, yield rewards can be used to measure the mean number of customers in the system at a given instant or the probability that a certain server is in use at a given instant, whereas bonus rewards can be employed to assess the frequency with which customers arrive or are served at a given instant.

The third criterion (computational cost) requires associating rewards with states and transitions to be not exceedingly expensive: in particular, a full scan of the state space should be avoided. As we shall see in Sect. 12.3, the algebra based method satisfies this requirement because rewards can be computed and assigned to states and transitions at semantic model construction time.

Finally, the fourth criterion (equational characterization) requires the method to allow process terms to be compositionally manipulated without altering their performance measures. This is an important feature. As an example, if one uses a measure insensitive equivalence to reduce the state space before evaluating the performance, there is the risk to merge together states which are different w.r.t. the measures of interest, thus resulting in wrong performance figures. In Sect. 12.4 we shall see that the algebra based method permits the definition of performance measure sensitive congruences over process terms. This is the other main objective of the algebra based method and constitutes its major advantage.

12.3 Syntax and Semantics for $\mathbf{EMPA}_{\mathrm{gr}_n}$

In this section we formalize the syntax and the semantics for the process algebra informally presented in the previous section. More precisely, denoted by $n \in \mathbb{N}$ the number of performance measures of interest, we define the syntax of a family of extended Markovian process algebras with generative-reactive synchronizations $\mathrm{EMPA}_{\mathrm{gr}_n}$, where each action is extended to accommodate a sequence of n pairs of rewards. Then we introduce semantic models constituted by reward master-slaves transition systems, an extension of master-slaves transition systems introduced in Chapter 11. Finally we present an operational semantics that maps $\mathrm{EMPA}_{\mathrm{gr}_n}$ terms onto reward master-slaves transition systems of order n.

12.3.1 Syntax and Informal Semantics

The main ingredients of our calculus are the actions, each composed of a type, a rate, and a sequence of pairs of yield and bonus rewards, and the algebraic operators. As far as actions are concerned, based on their rates they are classified into exponentially timed, immediate, and passive, as for EMPA$_{gr}$ (see Chapter 11). Moreover, based on their types they are classified into visible and invisible depending on whether they are different or equal to τ, as usual.

Definition 12.1 *Let AType be the set of action types including the invisible type τ, $ARate = \mathbb{R}_+ \cup \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\} \cup \{*_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\}$ be the set of action rates, $ARew = \mathbb{R} \cup \{*\}$ be the set of action rewards. We use a to range over AType, $\tilde{\lambda}$ to range over ARate, λ to range over exponentially timed rates, $\bar{\lambda}$ to range over nonpassive rates, $\tilde{y}$ to range over yield rewards (y if not $*$), and $\tilde{b}$ to range over bonus rewards (b if not $*$). The set of actions with $n \in \mathbb{N}$ pairs of rewards is defined by*

$$\begin{array}{rcl} Act_n & = & \{<a, \tilde{\lambda}, (\tilde{y}_1, \tilde{b}_1) \dots (\tilde{y}_n, \tilde{b}_n)> \in AType \times ARate \times (ARew \times ARew)^n \mid \\ & & (\tilde{\lambda} \in \{*_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\} \wedge \forall i \in \{1, \dots, n\}. \tilde{y}_i = \tilde{b}_i = *) \vee \\ & & (\tilde{\lambda} \in \mathbb{R}_+ \cup \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\} \wedge \forall i \in \{1, \dots, n\}. \tilde{y}_i, \tilde{b}_i \in \mathbb{R})\} \end{array}$$

■

Definition 12.2 *Let Const be a set of constants ranged over by A and let $ATRFun = \{\varphi : AType \longrightarrow AType \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of action type relabeling functions ranged over by φ. The set $\mathcal{L}_n$ of process terms of EMPA$_{\mathrm{gr}_n}$ is generated by the following syntax*

$$E ::= \underline{0} \mid <a, \tilde{\lambda}, (\tilde{y}_1, \tilde{b}_1) \dots (\tilde{y}_n, \tilde{b}_n)>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $L, S \subseteq AType - \{\tau\}$. ■

The meaning of the operators is the same as for EMPA$_{\mathrm{gr}}$ (see Chapter 11). In particular, the action prefix operator "$<a, \tilde{\lambda}, (\tilde{y}_1, \tilde{b}_1) \dots (\tilde{y}_n, \tilde{b}_n)>.\_$" denotes the sequential composition of an action and a term. Term $<a, \tilde{\lambda}, (\tilde{y}_1, \tilde{b}_1) \dots (\tilde{y}_n, \tilde{b}_n)>.E$ can execute an action with type a and rate $\tilde{\lambda}$, thus making the corresponding state earn additional yield rewards $\tilde{y}_1 \dots \tilde{y}_n$ and the related transition gain bonus rewards $\tilde{b}_1 \dots \tilde{b}_n$, and then behaves as term E. As far as the parallel operator is concerned, since only the rewards of active actions are specified, in case of synchronization they are handled as follows. The yield rewards of an active action are treated exactly as the rate of that action, i.e. they are multiplied by the execution probabilities of the passive actions involved in the synchronization. Instead, the bonus rewards of an active action are just inherited, as multiplying them by the execution probabilities of the aforementioned passive actions would lead to an underestimation of the performance measures. The reason is that, in the calculation of the performance measures, each bonus reward of a transition is multiplied by a factor which is proportional to the rate of the transition itself, hence multiplying the rates by the execution probabilities of passive actions is all we have to do. In the case of synchronization between two passive actions, the rewards of the resulting passive actions are still unspecified.

Finally, let partial function $Def_n : Const \multimap \mathcal{L}_n$ be a set of *constant defining equations* of the form $A \triangleq E$. In order to guarantee the correctness of recursive definitions, as usual we restrict ourselves to the set $\mathcal{G}_n$ of terms that are closed and guarded w.r.t. Def_n.

12.3.2 Reward Master-Slaves Transition Systems

The semantic model of $\mathrm{EMPA}_{\mathrm{gr}_n}$ is a special kind of LTS we call *master-slaves transition system of order* n (RMSTS_n for short), whose transitions are labeled with elements of Act_n. As for MSTSs defined in Sect. 11.7.2, each state of a RMSTS_n has a single *master bundle* composed of all the transitions labeled with an active action and, for each action type a, a single *slave bundle* of type a composed of all the transitions labeled with a passive action of type a (which must all have the same priority level). For the sake of simplicity, in the rest of this section we shall deal with reward pair sequences of length one.

Definition 12.3 *A reward master-slaves transition system of order 1 ($RMSTS_1$) is a triple*

$$(S, AType, \longrightarrow)$$

where: [2]

- *S is a set of states;*
- *$AType$ is a set of action types;*
- *$\longrightarrow \in \mathcal{M}(S \times Act_1 \times S)$ is a multiset of transitions such that for all $s \in S$ and $a \in AType$:*
$$(s \xrightarrow{a,*_{l',w'},(*,*)} s' \wedge s \xrightarrow{a,*_{l'',w''},(*,*)} s'') \implies l' = l''$$

A rooted reward master-slaves transition system of order 1 ($RRMSTS_1$) is a quadruple

$$(S, AType, \longrightarrow, s_0)$$

where $(S, AType, \longrightarrow)$ is a $RMSTS_1$ and $s_0 \in S$ is the initial state. ■

We point out that, as for MSTSs of Chapter 11, the transition relation is a multiset, not a set, so to allow for the multiplicity of identically labeled transitions to be taken into account.

The choice among the bundles of transitions enabled in a state is nondeterministic. The choice of a transition within the master bundle and within each slave bundle of a state is performed as for MSTSs of Chapter 11. Similarly as for MSTSs, from the RRMSTS_1 originating from fully specified system we can derive a performance model in the form of a semi-Markov reward processes (see Chapter 2). If only immediate transitions occur, such process is considered to be a reward DTMC, otherwise (even if not strictly necessary)

[2] We use "$\{\!|$" and "$|\!\}$" as brackets for multisets and $\mathcal{M}(S)$ ($\mathcal{P}(S)$) to denote the collection of multisets over (subsets of) set S.

it can be turned into a reward CTMC. As far as the yield rewards are concerned, when constructing a semi-Markov reward processes from a RRMSTS_1 we proceed as follows. Whenever a state has several actions, be it due to an alternative composition operator or a parallel composition operator, we make the additivity assumption, i.e. we assume that the yield reward earned by the state is the sum of the yield rewards of its transitions. This assumption is consistent with the race inherent in the parallel composition operator and with the adoption of the race policy for the alternative composition operator, i.e. with viewing alternative actions as being in parallel execution, hence all contributing to the reward accumulation in the state.

12.3.3 Operational Semantics

The formal semantics for $\mathrm{EMPA}_{\mathrm{gr}_n}$ maps terms onto RRMSTS_n. In the following, we shall consider $\mathrm{EMPA}_{\mathrm{gr}_1}$ for the sake of simplicity. We also provide the following shorthands to make the definition of the operational semantic rules easier.

Definition 12.4 *Given a* $RMSTS_1$ $M = (S, AType, \longrightarrow)$, $s \in S$, *and* $a \in AType$, *we denote by* $L_a(s)$ *the priority level of the slave transitions of type* a *executable at* s ($L_a(s) = 0$ *if the slave bundle* a *of* s *is empty) and we denote by* $W_a(s)$ *the overall weight of the slave transitions of type* a *executable at* s*:*

$$W_a(s) = \sum \{\!|\, w \mid \exists s' \in S.\, s \xrightarrow{a, *_{L_a(s), w}, (*, *)} s' \,|\!\}$$

Furthermore, we extend the real number multiplication to immediate rates as follows:

$$\infty_{l,w} \cdot p = \infty_{l,w \cdot p}$$

■

The operational semantics for $\mathrm{EMPA}_{\mathrm{gr}_1}$ is the least RMSTS_1 $(\mathcal{G}_1, AType, \longrightarrow_1)$ satisfying the inference rules of Table 12.1, where in addition to the rules $(Ch1_l)$, $(Ch2_l)$, $(Pa1_l)$, $(Pa2_l)$, $(Sy1_l)$ referring to a move of the lefthand process E_1, we consider also the symmetrical rules $(Ch1_r)$, $(Ch2_r)$, $(Pa1_r)$, $(Pa2_r)$, $(Sy1_r)$ taking into account the moves of the righthand process E_2, obtained by exchanging the roles of terms E_1 and E_2. Similarly to [107], we consider the operational rules as generating a multiset of transitions (consistently with the definition of RMSTS_1), where a transition has arity m if and only if it can be derived in m possible ways from the operational rules.

Some explanations are now in order. As for $\mathrm{EMPA}_{\mathrm{gr}}$ we can discard lower priority passive transitions of the same type without compromising the achievement of the congruence property for a notion of Markovian bisimulation equivalence. This is carried out in rules $(Ch2_l)$ and $(Ch2_r)$ for the alternative composition operator and rules $(Pa1_l)$ and $(Pa1_r)$ for the parallel composition operator by using $L_a(E)$. In the case of a synchronization, the evaluation of the yeld and bonus rewards of the resulting action is carried out by rules $(Sy1_l)$, $(Sy1_r)$, and $(Sy2)$ as follows. Whenever an active action synchronizes with a passive action of the same type in rules $(Sy1_l)$ and $(Sy1_r)$, the yeld reward of the active action undergoes the same treatment of its rate, i.e. the yeld reward of the resulting active action is evaluated by multiplying the yeld reward of

$$(\boldsymbol{Pr}) \quad <a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E$$

$$(\boldsymbol{Hi1}) \quad \frac{E \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'}{E/L \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'/L} \quad a \notin L \qquad (\boldsymbol{Hi2}) \quad \frac{E \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'}{E/L \xrightarrow{\tau,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'/L} \quad a \in L$$

$$(\boldsymbol{Re}) \quad \frac{E \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'}{E[\varphi] \xrightarrow{\varphi(a),\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'[\varphi]}$$

$$(\boldsymbol{Ch1_l}) \quad \frac{E_1 \xrightarrow{a,\tilde{\lambda},(y,b)}_1 E_1'}{E_1 + E_2 \xrightarrow{a,\tilde{\lambda},(y,b)}_1 E_1'} \qquad (\boldsymbol{Ch2_l}) \quad \frac{E_1 \xrightarrow{a,*_{l,w},(*,*)}_1 E_1' \quad l \geq L_a(E_2)}{E_1 + E_2 \xrightarrow{a,*_{l,w},(*,*)}_1 E_1'}$$

$$(\boldsymbol{Pa1_l}) \quad \frac{E_1 \xrightarrow{a,\tilde{\lambda},(y,b)}_1 E_1'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda},(y,b)}_1 E_1' \parallel_S E_2} \quad a \notin S$$

$$(\boldsymbol{Pa2_l}) \quad \frac{E_1 \xrightarrow{a,*_{l,w},(*,*)}_1 E_1' \quad l \geq L_a(E_2)}{E_1 \parallel_S E_2 \xrightarrow{a,*_{l,w},(*,*)}_1 E_1' \parallel_S E_2} \quad a \notin S$$

$$(\boldsymbol{Sy1_l}) \quad \frac{E_1 \xrightarrow{a,\tilde{\lambda},(y,b)}_1 E_1' \quad E_2 \xrightarrow{a,*_{l,w},(*,*)}_1 E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}\cdot\frac{w}{W_a(E_2)},(y\cdot\frac{w}{W_a(E_2)},b)}_1 E_1' \parallel_S E_2'} \quad a \in S$$

$$(\boldsymbol{Sy2}) \quad \frac{E_1 \xrightarrow{a,*_{l_1,w_1},(*,*)}_1 E_1' \quad E_2 \xrightarrow{a,*_{l_2,w_2},(*,*)}_1 E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,*_{\max(l_1,l_2),p\cdot N},(*,*)}_1 E_1' \parallel_S E_2'} \quad a \in S$$

$$\text{where:} \quad p = \frac{w_1}{W_a(E_1)} \cdot \frac{w_2}{W_a(E_2)} \qquad N = \begin{cases} W_a(E_1) + W_a(E_2) & \text{if } l_1 = l_2 \\ W_a(E_1) & \text{if } l_1 > l_2 \\ W_a(E_2) & \text{if } l_2 > l_1 \end{cases}$$

$$(\boldsymbol{Co}) \quad \frac{E \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'}{A \xrightarrow{a,\tilde{\lambda},(\tilde{y},\tilde{b})}_1 E'} \quad A \triangleq E$$

Table 12.1: $\mathrm{EMPA}_{\mathrm{gr}_1}$ operational semantics

the active action by the probability of choosing the passive action, while the bonus reward is just inherited. Whenever two passive actions of type a synchronize in rule $(Sy2)$ the reactive priority level and weight of the resulting passive action are computed as for $\mathrm{EMPA_{gr}}$, while its yeld and bonus rewards are just unspecified.

Definition 12.5 *The integrated semantics of $E \in \mathcal{G}_1$ is the RRMSTS$_1$*

$$\mathcal{I}_1[\![E]\!] = (\mathcal{G}_{1,E}, AType, \longrightarrow_{1,E}, E)$$

where $\mathcal{G}_{1,E}$ is the set of terms reachable from E according to the RMSTS$_1$ $(\mathcal{G}_1, AType, \longrightarrow_1)$ and $\longrightarrow_{1,E}$ is the restriction of $\longrightarrow_1$ to transitions between terms in $\mathcal{G}_{1,E}$. We say that $E \in \mathcal{G}_1$ is performance closed if and only if $\mathcal{I}_1[\![E]\!]$ does not contain passive transitions. We denote by $\mathcal{E}_1$ the set of performance closed terms of $\mathcal{G}_1$. ■

We conclude by recalling that from $\mathcal{I}_1[\![E]\!]$ two projected semantic models can be obtained by essentially dropping action rates or action types, respectively. Before applying such a transformation to $\mathcal{I}_1[\![E]\!]$, lower priority active transitions are pruned because E is no longer to be composed with other terms as it describes the whole system we are interested in. The *functional semantics* $\mathcal{F}_1[\![E]\!]$ is a standard LTS whose transitions are decorated with action types only. The *Markovian semantics* $\mathcal{M}_1[\![E]\!]$ is instead a reward CTMC or DTMC, as seen in Sect. 12.3.2, which is well defined only if E is performance closed.

12.4 Reward Based Markovian Bisimulation Equivalence

In this section we face the problem of defining a notion of equivalence in the bisimulation style, which equates $\mathrm{EMPA_{gr_n}}$ terms possessing the same functional, probabilistic, prioritized and exponentially timed behavior as well as the same performance measure values. We solve this problem through successive conceptual improvements, by subsequently defining three notions of equivalence satisfying the requirement above, each coarser then the previous one. First (in Sect. 12.4.1), we define an equivalence along the lines of that defined in [19], which, whenever possible, aggregates yield rewards, but does not manipulate bonus rewards. Then (in Sect. 12.4.2), we show how to define a Markovian bisimulation congruence that aggregates bonus rewards as well. This is achieved by taking into account in the aggregation process the execution probability of the transitions to which the bonus rewards are attached. As a final step (in Sect. 12.4.3), we show how to define a Markovian bisimulation congruence that allows yield rewards and bonus rewards to be used interchangeably up to suitable correcting factors, aiming at the introduction of a normal form for rewards. More precisely, we demonstrate that this is possible in the continuous time case, while it is not possible in the discrete time case because compositionality is lost. We show that all the equivalences above are congruences, thus providing support for compositional manipulation while preserving the values of the specified performance measures, and, for each of them, we give a sound and complete axiomatization for nonrecursive process terms.

12.4.1 Aggregating Yield Rewards

Our Markovian bisimulation equivalence which aggregates yield rewards extends the Markovian bisimulation equivalence $\sim_{\rm MB}$ for EMPA$_{\rm gr}$ presented in Chapter 11 by using the following approach introduced in [19]. According to the observations made in Sect. 12.2 and to the formulae in Sect. 2.2.8 we have that, because of the additivity assumption, yield rewards must be handled in the same way as rates, i.e. equivalent terms must gain the same aggregated yield reward when reaching the same equivalence class of terms by executing actions of the same type and priority level. Bonus rewards of actions of the same type and priority level, instead, cannot be summed up, as this would result in an overestimation of the specified performance measures. The reason is that, in the calculation of the performance measures, the bonus reward of a transition is multiplied by a factor which is proportional to the rate of the transition itself, hence summing rates up is all we have to do. As an example, in EMPA$_{\rm gr_1}$ it must hold that $<a, \lambda_1, (y_1, b)>.E + <a, \lambda_2, (y_2, b)>.E$ is equivalent to $<a, \lambda_1 + \lambda_2, (y_1 + y_2, b)>.E$.

We are now in a position of defining a Markovian bisimulation equivalence that is sensitive to performance measures. For the sake of simplicity, we shall be working with EMPA$_{\rm gr_1}$, with the understanding that, when working with arbitrarily long sequences of pairs of rewards, all the yield rewards must be treated in the same way and that all the bonus rewards must be treated in the same way. In order to present the equivalence for EMPA$_{\rm gr_1}$, we recall the definition of function priority level PL from Chapter 11 and we extend real number summation to rates and unspecified rewards similarly as we did in such chapter.

Definition 12.6 *We define function priority level* $PL : ARate \longrightarrow \mathbb{Z}$ *by:*

$$\begin{aligned} PL(*_{l,w}) &= -l \\ PL(\lambda) &= 0 \\ PL(\infty_{l,w}) &= l \end{aligned}$$

and we extend the real number summation to rates of the same priority level and to unspecified rewards as follows:

$$\begin{aligned} *_{l,w_1} + *_{l,w_2} &= *_{l,w_1+w_2} \\ \infty_{l,w_1} + \infty_{l,w_2} &= \infty_{l,w_1+w_2} \\ * + * &= * \end{aligned}$$

We define function aggregated rate-yield $RY_1 : \mathcal{G}_1 \times AType \times \mathbb{Z} \times ARew \times \mathcal{P}(\mathcal{G}_1) \multimap ARate \times ARew$ *by:*

$$RY_1(E, a, l, \tilde{b}, C) = (Rate_1(E, a, l, \tilde{b}, C),\, Yield_1(E, a, l, \tilde{b}, C))$$

where:

$$\begin{aligned} Rate_1(E, a, l, \tilde{b}, C) &= \sum\{\!| \tilde{\lambda} \mid \exists \tilde{y}.\, \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \wedge PL(\tilde{\lambda}) = l \,|\!\} \\ Yield_1(E, a, l, \tilde{b}, C) &= \sum\{\!| \tilde{y} \mid \exists \tilde{\lambda}.\, \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \wedge PL(\tilde{\lambda}) = l \,|\!\} \end{aligned}$$

with $RY_1(E, a, l, \tilde{b}, C) = \bot$ *whenever the multisets above are empty.* ■

Definition 12.7 *An equivalence relation $\mathcal{B} \subseteq \mathcal{G}_1 \times \mathcal{G}_1$ is a RY-Markovian bisimulation of order 1 if and only if, whenever $(E_1, E_2) \in \mathcal{B}$, then for all $a \in AType$, $l \in \mathbb{Z}$, $\tilde{b} \in ARew$, and equivalence classes $C \in \mathcal{G}_1/\mathcal{B}$*

$$RY_1(E_1, a, l, \tilde{b}, C) = RY_1(E_2, a, l, \tilde{b}, C)$$

■

It is easy to see that the union of all the RY-Markovian bisimulations of order 1 is a RY-Markovian bisimulation of order 1.

Definition 12.8 *We call $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$, defined as the union of all the RY-Markovian bisimulations of order 1, the RY-Markovian bisimulation equivalence of order 1.* ■

Theorem 12.1 *Let $E_1, E_2 \in \mathcal{G}_1$. If $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2$ then:*

1. *For all $<a, \tilde{\lambda}, (\tilde{y}, \tilde{b})> \in Act_1$, $<a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} <a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E_2$.*
2. *For all $L \subseteq AType - \{\tau\}$, $E_1/L \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2/L$.*
3. *For all $\varphi \in ATRFun$, $E_1[\varphi] \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2[\varphi]$.*
4. *For all $F \in \mathcal{G}_1$, $E_1 + F \sim_{\mathrm{MB}_1} E_2 + F$ and $F + E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} F + E_2$.*
5. *For all $F \in \mathcal{G}_1$ and $S \subseteq AType - \{\tau\}$, $E_1 \,\|_S\, F \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2 \,\|_S\, F$ and $F \,\|_S\, E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} F \,\|_S\, E_2$.*

Additionally, $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$ is a congruence w.r.t. recursive constant definitions.
Proof: See Appendix C.1. □

We observe that the congruence result above holds because the operational semantics is defined in such a way that lower priority active transitions are not pruned. If this were not the case, we would have e.g. $<a_1, \lambda, (y_1, b_1)>.\underline{0} + <a_2, \infty_{l,w}, (y_2, b_2)>.\underline{0} \sim_{\mathrm{MB}_1^{\mathrm{RY}}} <a_2, \infty_{l,w}, (y_2, b_2)>.\underline{0}$ as both terms would have only one transition labeled with $<a_2, \infty_{l,w}, (y_2, b_2)>$, but $(<a_1, \lambda, (y_1, b_1)>.\underline{0} + <a_2, \infty_{l,w}, (y_2, b_2)>.\underline{0}) \,\|_{\{a_2\}}\, \underline{0} \not\sim_{\mathrm{MB}_1^{\mathrm{RY}}} <a_2, \infty_{l,w}, (y_2, b_2)>.\underline{0} \,\|_{\{a_2\}}\, \underline{0}$ because the first term has a transition labeled with action $<a_1, \lambda, (y_1, b_1)>$ while the second term has no transitions at all. On the contrary, the removal of lower priority passive actions of a given type does not cause any problem.

Theorem 12.2 *Let $\mathcal{A}_1^{\mathrm{RY}}$ be the set of axioms in Table 12.2. The deductive system $Ded(\mathcal{A}_1^{\mathrm{RY}})$ is sound and complete for $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$ over the set of nonrecursive terms of $\mathcal{G}_1$.*
Proof: The proof is similar to that of the corresponding Theorem 11.5 of Chapter 11 (presented in Appendix B.3) with the difference that a nonrecursive term $E \in \mathcal{G}_1$ is defined to be in sum normal form (snf) if and only if E is $\underline{0}$ or $\sum_{i \in I} <a_i, \tilde{\lambda}_i, (\tilde{y}_i, \tilde{b}_i)>.E_i$ with every E_i in snf, where the nonempty finite set I is such that there are no $i, i' \in I$ for which $a_i = a_{i'} \wedge \tilde{\lambda}_i = *_{l_i, w_i} \wedge \tilde{\lambda}_{i'} = *_{l_{i'}, w_{i'}} \wedge l_i \neq l_{i'}$. □

We observe that axiom $(\mathcal{A}_4)_1^{\mathrm{RY}}$ is exactly the rule we wanted our equivalence to satisfy, while axiom $(\mathcal{A}_5)_1^{\mathrm{RY}}$ establishes that lower priority passive actions of a given type can be left out.

$$
\begin{array}{lrcll}
(\mathcal{A}_1)_1^{\mathrm{RY}} & (E_1 + E_2) + E_3 & = & E_1 + (E_2 + E_3) & \\
(\mathcal{A}_2)_1^{\mathrm{RY}} & E_1 + E_2 & = & E_2 + E_1 & \\
(\mathcal{A}_3)_1^{\mathrm{RY}} & E + \underline{0} & = & E & \\
(\mathcal{A}_4)_1^{\mathrm{RY}} & \multicolumn{4}{l}{<a, \tilde{\lambda}_1, (\tilde{y}_1, \tilde{b})>.E + <a, \tilde{\lambda}_2, (\tilde{y}_2, \tilde{b})>.E \ =} \\
 & & & <a, \tilde{\lambda}_1 + \tilde{\lambda}_2, (\tilde{y}_1 + \tilde{y}_2, \tilde{b})>.E & \text{if } PL(\tilde{\lambda}_1) = PL(\tilde{\lambda}_2) \\
(\mathcal{A}_5)_1^{\mathrm{RY}} & \multicolumn{4}{l}{<a, *_{l_1,w_1}, (*,*)>.E_1 + <a, *_{l_2,w_2}, (*,*)>.E_2 \ =} \\
 & & & <a, *_{l_1,w_1}, (*,*)>.E_1 & \text{if } l_1 > l_2 \\
 & & & & \\
(\mathcal{A}_6)_1^{\mathrm{RY}} & \underline{0}/L & = & \underline{0} & \\
(\mathcal{A}_7)_1^{\mathrm{RY}} & (<a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E)/L & = & <a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.(E/L) & \text{if } a \notin L \\
(\mathcal{A}_8)_1^{\mathrm{RY}} & (<a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E)/L & = & <\tau, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.(E/L) & \text{if } a \in L \\
(\mathcal{A}_9)_1^{\mathrm{RY}} & (E_1 + E_2)/L & = & E_1/L + E_2/L & \\
 & & & & \\
(\mathcal{A}_{10})_1^{\mathrm{RY}} & \underline{0}[\varphi] & = & \underline{0} & \\
(\mathcal{A}_{11})_1^{\mathrm{RY}} & (<a, \tilde{\lambda}, (\tilde{y}, \tilde{b})>.E)[\varphi] & = & <\varphi(a), \tilde{\lambda}, (\tilde{y}, \tilde{b})>.(E[\varphi]) & \\
(\mathcal{A}_{12})_1^{\mathrm{RY}} & (E_1 + E_2)[\varphi] & = & E_1[\varphi] + E_2[\varphi] &
\end{array}
$$

$$
\begin{array}{ll}
(\mathcal{A}_{13})_1^{\mathrm{RY}} & \sum\limits_{i \in I_0} <a_i, \tilde{\lambda}_i, (\tilde{y}_i, \tilde{b}_i)>.E_i \parallel_S \sum\limits_{i \in I_1} <a_i, \tilde{\lambda}_i, (\tilde{y}_i, \tilde{b}_i)>.E_i \ = \\
 & \quad \sum\limits_{j \in I_0, a_j \notin S} <a_j, \tilde{\lambda}_j, (\tilde{y}_j, \tilde{b}_j)>.(E_j \parallel_S \sum\limits_{i \in I_1} <a_i, \tilde{\lambda}_i, (\tilde{y}_i, \tilde{b}_i)>.E_i) \ + \\
 & \quad \sum\limits_{j \in I_1, a_j \notin S} <a_j, \tilde{\lambda}_j, (\tilde{y}_j, \tilde{b}_j)>.(\sum\limits_{i \in I_0} <a_i, \tilde{\lambda}_i, (\tilde{y}_i, \tilde{b}_i)>.E_i \parallel_S E_j) \ + \\
 & \quad \sum\limits_{k \in K_0} \sum\limits_{h \in P_{1,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h / W_{1,a_k}), (\tilde{y}_k \cdot (w_h / W_{1,a_k}), \tilde{b}_k)>.(E_k \parallel_S E_h) \ + \\
 & \quad \sum\limits_{k \in K_1} \sum\limits_{h \in P_{0,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h / W_{0,a_k}), (\tilde{y}_k \cdot (w_h / W_{0,a_k}), \tilde{b}_k)>.(E_h \parallel_S E_k) \ + \\
 & \quad \sum\limits_{k \in P'_0} \sum\limits_{h \in P_{1,a_k}} <a_k, *_{\max(l_k, l_h), (w_k / W_{0,a_k}) \cdot (w_h / W_{0,a_k}) \cdot N_{a_k}}, (*,*)>.(E_k \parallel_S E_h) \\
 & \text{where } I_0 \cap I_1 = \emptyset, \ \tilde{\lambda}_i = *_{l_i, w_i} \text{ for } i \in I_0 \cup I_1.\, PL(\tilde{\lambda}_i) < 0, \text{ and for } j \in \{0, 1\}
\end{array}
$$

$$
\begin{array}{rcl}
L_{j,a} & = & \max\{ l_k \mid k \in I_j \land a_k = a \land \tilde{\lambda}_k = *_{l_k, w_k} \} \\
P_{j,a} & = & \{ k \in I_j \mid a_k = a \land \tilde{\lambda}_k = *_{l_k, w_k} \land l_k = L_{j,a} \} \\
K_j & = & \{ k \in I_j \mid a_k \in S \land PL(\tilde{\lambda}_k) \geq 0 \land P_{1-j, a_k} \neq \emptyset \} \\
P'_0 & = & \{ k \in I_0 \mid \exists a \in S.\, k \in P_{0,a} \land P_{1,a} \neq \emptyset \} \\
W_{j,a} & = & \sum \{\!| w_k \mid k \in P_{j,a} \land \tilde{\lambda}_k = *_{l_k, w_k} |\!\} \\
N_a & = & \begin{cases} W_{0,a} + W_{1,a} & \text{if } L_{0,a} = L_{1,a} \\ W_{0,a} & \text{if } L_{0,a} > L_{1,a} \\ W_{1,a} & \text{if } L_{1,a} > L_{0,a} \end{cases}
\end{array}
$$

Table 12.2: Axiomatization of $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$

Theorem 12.3 *Let $E_1, E_2 \in \mathcal{E}_1$. If $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2$ then the value of the reward based performance measure is the same for E_1 and E_2.*

Proof: Let us consider only the case of stationary performance measures, for which formula (2.2) of Sect. 2.2.8 applies; the case of transient performance measures is similar. From $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$ it follows that E_1 and E_2 are Markovian bisimulation equivalent. Therefore their underlying MCs are lumping equivalent. This means that their underlying MCs minimized w.r.t. $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$, in which every maximal set of equivalent original states results in a single macrostate, are isomorphic. As a consequence, if we take two macrostates C_1, C_2 from the two minimized MCs such that the contained original states are all equivalent, then $\pi_{C_1} = \pi_{C_2}$ with $\pi_{C_k} = \sum_{s \in C_k} \pi_s$ for $k = 1, 2$. From the definition of $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$, it follows that every original state s of C_1 and C_2 has the same aggregated reward $\sum_{a \in AType} \mathit{Yield}_1(s, a, l_{\max}(s), \mathcal{E}_1)$, where $l_{\max}(s)$ is the maximum priority level of the transitions of s. Therefore $\sum_{s \in C_1} (\sum_{a \in AType} \mathit{Yield}_1(s, a, l_{\max}(s), \mathcal{E}_1)) \cdot \pi_s = \sum_{s \in C_2} (\sum_{a \in AType} \mathit{Yield}_1(s, a, l_{\max}(s), \mathcal{E}_1)) \cdot \pi_s$. By summing over all the macrostates C_1 of the minimized MC underlying E_1 and over all the macrostates C_2 of the minimized MC underlying E_2, we obtain that the first summand of formula (2.2) of Sect. 2.2.8, i.e. $\sum_i y_i \cdot \pi_i$, is the same for E_1 and E_2.
As far as the second summand of formula (2.2) is concerned, since we do not aggregate bonus rewards, we have that it is obviously the same for E_1 and E_2. In conclusion, the value of the performance measure is the same for E_1 and E_2. $\square$

We conclude by pointing out that all the results above smoothly extend to an arbitrary length n of the sequences of pairs of rewards within the actions. In particular, the set of axioms $\mathcal{A}_n^{\mathrm{RY}}$ is a trivial extension of $\mathcal{A}_1^{\mathrm{RY}}$.

12.4.2 Aggregating Bonus Rewards

As witnessed by axiom $(\mathcal{A}_4)_1^{\mathrm{RY}}$, $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$ aggregates rates and yield rewards without manipulating bonus rewards at all. However, if we look at the way performance measures are computed according to formulae (2.2) and (2.3) of Sect. 2.2.8, we note that whenever two actions are merged into a single one, then their bonus rewards can be aggregated as well. Unlike the rates and the yield rewards of those two actions, the bonus rewards are not just summed up as each of them needs to be preliminarily multiplied by the execution probability of the corresponding action. As an example, $<a, \lambda_1, (y_1, b_1)>.E + <a, \lambda_2, (y_2, b_2)>.E$ can be equated to $<a, \lambda_1 + \lambda_2, (y_1 + y_2, \frac{\lambda_1}{\lambda_1+\lambda_2} \cdot b_1 + \frac{\lambda_2}{\lambda_1+\lambda_2} \cdot b_2)>.E$; similarly $<a, \infty_{l,w_1}, (y_1, b_1)>.E + <a, \infty_{l,w_2}, (y_2, b_2)>.E$ can be equated to $<a, \infty_{l,w_1+w_2}, (y_1 + y_2, \frac{w_1}{w_1+w_2} \cdot b_1 + \frac{w_2}{w_1+w_2} \cdot b_2)>.E$. To be more precise, the probability by which each bonus reward involved in the aggregation must be multiplied is the probability of executing the corresponding action conditioned on the fact that one of the actions involved in the aggregation is executed. Considering such a conditional execution probability instead of just the execution probability is not only necessary to preserve the value of the performance measures according to formulae (2.2) and (2.3) of Sect. 2.2.8, but is also crucial to get the congruence property. We introduce below an improved reward based RY-Markovian bisimulation congruence that aggregates bonus rewards as well.

Definition 12.9 *We extend the real number division to rates of the same priority level as follows:*

$$\begin{array}{rcl} *_{l,w_1} \; / \; *_{l,w_2} & = & *_{l,w_1/w_2} \\ \infty_{l,w_1} \; / \; \infty_{l,w_2} & = & w_1/w_2 \end{array}$$

and we extend the real number multiplication to passive rates and unspecified rewards as follows:

$$*_{l,w} \;\cdot\; * \;=\; *$$

We define function aggregated rate-yield-bonus $RYB_1 : \mathcal{G}_1 \times AType \times \mathbb{Z} \times \mathcal{P}(\mathcal{G}_1) \longrightarrow\!\!\!\!\!\!\circ\;\; ARate \times ARew \times ARew$ *by:*

$$RYB_1(E, a, l, C) = (Rate_1(E, a, l, C),\, Yield_1(E, a, l, C), Bonus_1(E, a, l, C))$$

where:

$$\begin{array}{rcl} Rate_1(E, a, l, C) & = & \sum\{\!|\, \tilde{\lambda} \mid \exists \tilde{y}, \tilde{b}. \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \wedge PL(\tilde{\lambda}) = l \,|\!\} \\ Yield_1(E, a, l, C) & = & \sum\{\!|\, \tilde{y} \mid \exists \tilde{\lambda}, \tilde{b}. \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \wedge PL(\tilde{\lambda}) = l \,|\!\} \\ Bonus_1(E, a, l, C) & = & \sum\{\!|\, \frac{\tilde{\lambda}}{Rate_1(E,a,l,C)} \cdot \tilde{b} \mid \exists \tilde{y}. \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \;\wedge PL(\tilde{\lambda}) = l \,|\!\} \end{array}$$

with $RYB_1(E, a, l, C) = \perp$ *whenever the multisets above are empty.* ■

Definition 12.10 *An equivalence relation* $\mathcal{B} \subseteq \mathcal{G}_1 \times \mathcal{G}_1$ *is a RYB-Markovian bisimulation of order 1 if and only if, whenever* $(E_1, E_2) \in \mathcal{B}$*, then for all* $a \in AType$*,* $l \in \mathbb{Z}$*, and equivalence classes* $C \in \mathcal{G}_1/\mathcal{B}$

$$RYB_1(E_1, a, l, C) = RYB_1(E_2, a, l, C)$$

■

It is easy to see that the union of all the RYB-Markovian bisimulations of order 1 is a RYB-Markovian bisimulation of order 1.

Definition 12.11 *We call* $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$*, defined as the union of all the RYB-Markovian bisimulations of order 1, the RYB-Markovian bisimulation equivalence of order 1.* ■

Theorem 12.4 $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$ *is a congruence w.r.t. all the algebraic operators as well as recursive constant definitions.*

Proof: See Appendix C.2. □

Theorem 12.5 *Let* $\mathcal{A}_1^{\mathrm{RYB}}$ *be the set of axioms obtained from those in Table 12.2 by replacing* $(\mathcal{A}_4)_1^{\mathrm{RY}}$ *with*

$$\begin{array}{ll} (\mathcal{A}_4)_1^{\mathrm{RYB}} & <a, \tilde{\lambda}_1, (\tilde{y}_1, \tilde{b}_1)>.E + <a, \tilde{\lambda}_2, (\tilde{y}_2, \tilde{b}_2)>.E \;= \\ & \quad <a, \tilde{\lambda}_1 + \tilde{\lambda}_2, (\tilde{y}_1 + \tilde{y}_2, \frac{\tilde{\lambda}_1}{\tilde{\lambda}_1 + \tilde{\lambda}_2} \cdot \tilde{b}_1 + \frac{\tilde{\lambda}_2}{\tilde{\lambda}_1 + \tilde{\lambda}_2} \cdot \tilde{b}_2)>.E \end{array}$$

if $PL(\tilde{\lambda}_1) = PL(\tilde{\lambda}_2)$*. The deductive system* $Ded(\mathcal{A}_1^{\mathrm{RYB}})$ *is sound and complete for* $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$ *over the set of nonrecursive terms of* $\mathcal{G}_1$*.*

Proof: The proof is similar to that of Theorem 12.2, with the difference that every term in sum normal form contains at most one summand with a certain action type, priority level, and derivative term. □

Theorem 12.6 *Let $E_1, E_2 \in \mathcal{E}_1$. If $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$ then the value of the reward based performance measure is the same for E_1 and E_2.*

Proof: Let us consider only the case of stationary performance measures, for which formula (2.2) of Sect. 2.2.8 applies; the case of transient performance measures is similar. From $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$ it follows that E_1 and E_2 are Markovian bisimulation equivalent. Therefore their underlying MCs are lumping equivalent. This means that their underlying MCs minimized w.r.t. $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$, in which every maximal set of equivalent original states results in a single macrostate, are isomorphic. As a consequence, if we take two macrostates C_1, C_2 from the two minimized MCs such that the contained original states are all equivalent, then $\pi_{C_1} = \pi_{C_2}$ with $\pi_{C_k} = \sum_{s \in C_k} \pi_s$ for $k = 1, 2$. From the proof of Theorem 12.3 we derive that the value of the first summand of formula (2.2) of Sect. 2.2.8 is the same for E_1 and E_2. As far as the second summand of formula (2.2) is concerned, fixed an equivalence class C of original states, from the definition of $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$ it follows that, for every action type a, every original state s of C_1 and C_2 has the same aggregated rate $\mathit{Rate}_1(s, a, l_{\max}(s), C)$ and the same aggregated conditional bonus reward $\mathit{Bonus}_1(s, a, l_{\max}(s), C)$ to reach C. As a consequence, summing over all the action types it turns out that every original state s of C_1 and C_2 has the same value for the product $\sum_{a \in AType} (\mathit{Bonus}_1(s, a, l_{\max}(s), C) \cdot \mathit{Rate}_1(s, a, l_{\max}(s), C))$, which is $\sum_{a \in AType} \sum \{\!| \, \tilde{b} \cdot \tilde{\lambda} \mid \exists \tilde{y}. \exists s' \in C. \, s \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 s' \wedge PL(\tilde{\lambda}) = l_{\max}(s) \, |\!\}$. From $\pi_{C_1} = \pi_{C_2}$ with $\pi_{C_k} = \sum_{s \in C_k} \pi_s$ for $k = 1, 2$, it follows that $\sum_{s \in C_1} (\sum_{a \in AType} \sum \{\!| \, \tilde{b} \cdot \tilde{\lambda} \mid \exists \tilde{y}. \exists s' \in C. \, s \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 s' \wedge PL(\tilde{\lambda}) = l_{\max}(s) \, |\!\}) \cdot \pi_s = \sum_{s \in C_2} (\sum_{a \in AType} \sum \{\!| \, \tilde{b} \cdot \tilde{\lambda} \mid \exists \tilde{y}. \exists s' \in C. \, s \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 s' \wedge PL(\tilde{\lambda}) = l_{\max}(s) \, |\!\}) \cdot \pi_s$. By summing over all the macrostates C_1 of the minimized MC underlying E_1 and over all the macrostates C_2 of the minimized MC underlying E_2 (and by transforming the occurring rates into the corresponding probabilities in the discrete time case), we obtain that the second summand of formula (2.2) of Sect. 2.2.8 is the same for E_1 and E_2. In conclusion, the value of the performance measure is the same for E_1 and E_2. $\square$

12.4.3 Mixing Yield and Bonus Rewards

Having the objective of defining a reward based Markovian bisimulation congruence that aggregates as much as possible, the question arises as to whether it is possible to consider just one type of reward instead of two. From the point of view of an equivalence, this can be rephrased in terms of being able to jointly consider yield and bonus rewards. By looking at formulae (2.2) and (2.3) of Sect. 2.2.8 and the way transition frequencies are computed, we note that in the continuous time case $<a, \lambda_1, (y_1, b_1)>.E + <a, \lambda_2, (y_2, b_2)>.E$ can be equated to $<a, \lambda_1 + \lambda_2, (y_1 + y_2 + \lambda_1 \cdot b_1 + \lambda_2 \cdot b_2, 0)>.E$, while in the discrete time case $<a, \infty_{l, w_1}, (y_1, b_1)>.E + <a, \infty_{l, w_2}, (y_2, b_2)>.E$ can be equated to $<a, \infty_{l, w_1 + w_2}, (y_1 + y_2 + \frac{w_1}{w_1 + w_2} \cdot b_1 + \frac{w_2}{w_1 + w_2} \cdot b_2, 0)>.E$. This gives rise to a normal form where only yield rewards are actually present, with bonus rewards being zero (or unspecified in the case of passive actions). However, in the discrete time case, we observe that the factor by which each bonus reward must be multiplied is equal to the execution probability of the transition to which it is attached. Since such a probability varies depending on the context in which the term is plugged, compositionality is lost. As an example, if we call E_1 and E_2 the two equivalent terms above, respectively, and we take E_3 defined by $<a, \infty_{l,w_3}, (y_3, b_3)>.E$, we have that the aggregated yield reward for $E_1 + E_3$ is $y_1 + y_2 + y_3 + \frac{w_1}{w_1+w_2+w_3} \cdot b_1 + \frac{w_2}{w_1+w_2+w_3} \cdot b_2 + \frac{w_3}{w_1+w_2+w_3} \cdot b_3$, while the aggregated yield reward for $E_2 + E_3$ is $y_1 + y_2 + \frac{w_1}{w_1+w_2} \cdot b_1 + \frac{w_2}{w_1+w_2} \cdot b_2 + y_3 + \frac{w_1+w_2}{w_1+w_2+w_3} \cdot 0 + \frac{w_3}{w_1+w_2+w_3} \cdot b_3$. We introduce below a further improved reward based Markovian bisimulation congruence that mixes yield and bonus rewards in the continuous time case.

Definition 12.12 *We define function aggregated rate-reward* RR_1: $\mathcal{G}_1 \times AType \times \mathbb{Z} \times \mathcal{P}(\mathcal{G}_1) \longrightarrow\!\!\!\!\!\circ\;\; (ARate \times ARew) \cup (ARate \times ARew \times ARew)$ *by:*

$$RR_1(E, a, l, C) = \begin{cases} (Rate_1(E, a, l, C), Reward_1(E, a, C)) & \text{if } l = 0 \\ RYB_1(E, a, l, C) & \text{if } l \neq 0 \end{cases}$$

where:

$$Reward_1(E, a, C) = \sum \{\!| \tilde{y} + \tilde{\lambda} \cdot \tilde{b} \mid \exists E' \in C.\, E \xrightarrow{a, \tilde{\lambda}, (\tilde{y}, \tilde{b})}_1 E' \wedge PL(\tilde{\lambda}) = 0 \,|\!\}$$

with $RR_1(E, a, l, C) = \perp$ *whenever the multisets above are empty.* ■

Definition 12.13 *An equivalence relation* $\mathcal{B} \subseteq \mathcal{G}_1 \times \mathcal{G}_1$ *is a RR-Markovian bisimulation of order 1 if and only if, whenever* $(E_1, E_2) \in \mathcal{B}$*, then for all* $a \in AType$*,* $l \in \mathbb{Z}$*, and equivalence classes* $C \in \mathcal{G}_1/\mathcal{B}$

$$RR_1(E_1, a, l, C) = RR_1(E_2, a, l, C)$$

■

It is easy to see that the union of all the RR-Markovian bisimulations of order 1 is a RR-Markovian bisimulation of order 1.

Definition 12.14 *We call* $\sim_{\mathrm{MB}_1^{\mathrm{RR}}}$*, defined as the union of all the RR-Markovian bisimulations of order 1, the RR-Markovian bisimulation equivalence of order 1.* ■

Theorem 12.7 $\sim_{\mathrm{MB}_1^{\mathrm{RR}}}$ *is a congruence w.r.t. all the algebraic operators as well as recursive constant definitions.*

Proof: The proof is an easy reworking of that of Theorem 12.4 (presented in Appendix C.2), with the difference that in the case of exponentially timed actions the equalities for $Yield_1$ and $Bonus_1$ do not come into play and the equalities for $Reward_1$ have the same form as the equalities for $Yield_1$. □

Theorem 12.8 *Let* $\mathcal{A}_1^{\mathrm{RR}}$ *be the set of axioms obtained from* $\mathcal{A}_1^{\mathrm{RYB}}$ *by adding*

$$(\mathcal{A}_{4'})_1^{\mathrm{RR}} \quad <a, \lambda, (y, b)>.E = <a, \lambda, (y + \lambda \cdot b, 0)>.E$$

The deductive system $Ded(\mathcal{A}_1^{\mathrm{RR}})$ *is sound and complete for* $\sim_{\mathrm{MB}_1^{\mathrm{RR}}}$ *over the set of nonrecursive terms of* $\mathcal{G}_1$*.*

Proof: The proof is similar to that of Theorem 12.2, with the differencethat the sum normal form additionally requires that each occurring exponentially timed action has bonus reward zero. □

Theorem 12.9 *Let $E_1, E_2 \in \mathcal{E}_1$. If $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RR}}} E_2$ then the value of the reward based performance measure is the same for E_1 and E_2.*

Proof: Similar to that of Theorem 12.6. □

12.5 An Overview of TwoTowers

The analysis of performance measures with $\mathrm{EMPA}_{\mathrm{gr}_n}$ has been fully automatized in TwoTowers [26, 19], a software tool for modeling and analyzing functional and performance properties of computer, communication and software systems described in $\mathrm{EMPA}_{\mathrm{gr}_n}$. As shown in Fig. 12.1, TwoTowers is composed of a graphical user interface, a compiler, a functional analyzer, a performance analyzer, and an integrated analyzer.

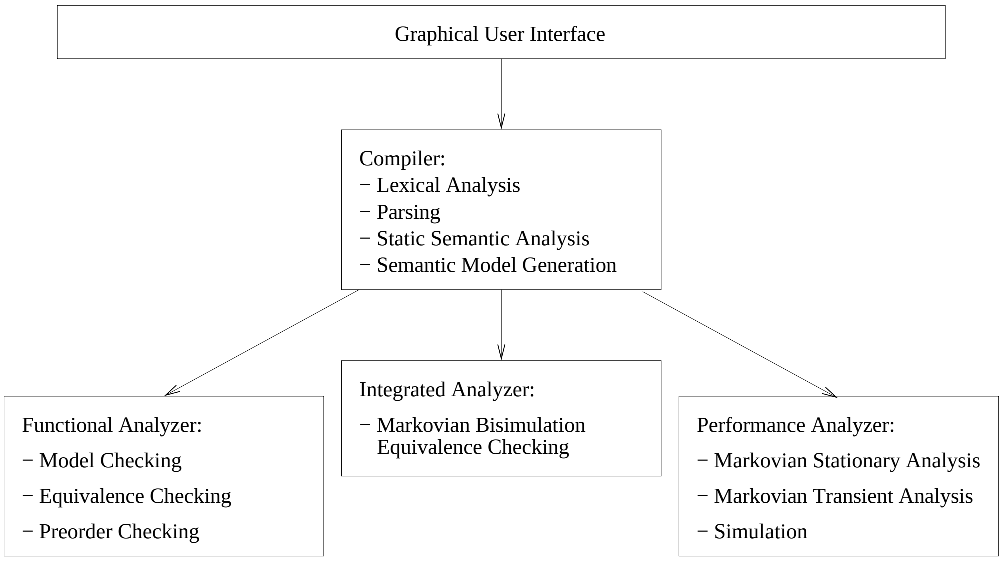


Figure 12.1: Architecture of TwoTowers

The graphical user interface allows the user to edit the specifications of the systems in $\mathrm{EMPA}_{\mathrm{gr}_n}$, compile them, and run the various analysis routines. Additionally, it permits to edit the specifications of functional requirements and QoS metrics for the systems under investigation.

The compiler is in charge of parsing $\mathrm{EMPA}_{\mathrm{gr}_n}$ specifications and pinpointing lexical, syntactical and static semantical errors. If a specification is correct, the compiler can produce the semantic model (integrated LTS, functional LTS, CTMC/DTMC) on which further analysis is based.

The integrated analyzer conducts those investigations that require both functional and performance information. It thus contains a routine to check two correct $\mathrm{EMPA}_{\mathrm{gr}_n}$ specifications for Markovian bisimulation equivalence of order n.

The functional analyzer takes care of verifying that certain functional requirements are satisfied by the functional LTS derived from a correct $\mathrm{EMPA}_{\mathrm{gr}_n}$ specification. This is achieved by interfacing TwoTowers with the Concurrency Workbench of New Century (CWB-NC) [63], thereby providing support for model checking in the μ-calculus or CTL [59], equivalence checking (strong and weak bisimulation equivalences [122] and may and must testing equivalences [74]), and preorder checking (may and must testing preorders [74]).

Finally, the performance analyzer computes certain performance measures on the CTMC/DTMC derived from a correct $\mathrm{EMPA}_{\mathrm{gr}_n}$ specification. This can be done via numerical analysis through the Markov Chain Analyzer (MarCA) [147] or via simulation. As far as the attachment of rewards is concerned, we observe that in TwoTowers it is separated from the system specification for operational convenience. In other words, two distinguished files must be prepared: one with the $\mathrm{EMPA}_{\mathrm{gr}_0}$ specification of the system and one with the specification of the performance measures, each under the form $id = (reward\_list)$ where id is the measure identifier and $reward\_list$ is a list of reward assignments of the form "$a\ y\ b$", which means that every nonpassive action with type a must be given yield reward y and bonus reward b for the measure under specification. This way of specifying rewards, which is equivalent to attaching sequences of reward pairs to actions, has two advantages. On the one hand, the system specification is not obfuscated by performance measure related details. On the other hand, the specification of performance measures can be easily updated without changing the specification of the system they refer to and can be reused for other system specifications.

12.6 Conclusion

In this chapter we have made a further step in the direction of adding expressive features to process algebras by introducing a way to describe performance measures, in order to allow the modeler to capture the QoS metrics of interest. In particular, following the approach of [19], we specify performance measures by attaching sequences of yield and bonus reward pairs to process algebra actions, thus resulting in a family of process algebras $\mathrm{EMPA}_{\mathrm{gr}_n}$. We have shown that this method achieves an acceptable expressive power and ease of use and, most importantly, allows performance measure sensitive congruences to be defined. In particular the coarsest reward based Markovian bisimulation equivalence that we have defined aggregates yield rewards as well as bonus rewards, provided that they are multiplied by the conditional probability of executing the actions to which they are attached, and allows yield rewards and bonus rewards to be used interchangeably in the continuous time case, provided that they are divided/multiplied by the rates of the actions to which they are attached. The impossibility result of this chapter, i.e. the fact that it is not possible to define a performance measure sensitive Markovian bisimulation congruence that allows yield and bonus rewards to be used interchangeably in the discrete time case, emphasizes the necessity of the bonus rewards. In the literature of Markov reward processes it is well known that yield and bonus rewards can be used interchangeably in the continuous time case, and in this chapter we have verified that such a property does not violate compositionality. In the continuous time case the yield rewards work well because of the race policy. In particular, the additivity assumption is sound because in every state all the transitions are viewed as being in parallel execution, hence each of them contributes with its yield reward to the accumulation of reward at the state. In the discrete time case, instead, the preselection policy applies, hence the bonus rewards are more natural to express performance measures. Besides being more convenient from the modeling viewpoint, in the discrete time case the bonus rewards are also necessary from the compositionality viewpoint, i.e. they cannot be transformed into yield rewards if we want to get a congruence. In fact, if we transform them into yield rewards, we have that the contribution of the transitions to the accumulation of reward at the state is given by their average bonus reward, i.e. the weighted sum of their bonus rewards with each of them multiplied by the execution probability of the corresponding transition. Since the above mentioned execution probabilities (unlike the rates in the continuous time case) vary depending on the environment in which the state is placed, compositionality is lost.

We would like to point out the following practice related observation. Describing the rewards directly within the Markovian process algebra specifications of the systems has the advantage of allowing the specifications to be compositionally manipulated while preserving the value of the performance measures. This advantage on the analysis side is unfortunately diminished by a drawback on the modeling side: the system specifications are obfuscated with performance measure related details. However, the Markovian process algebra specification of a system and the specification of its reward based performance measures of interest can be easily decoupled by separately describing the rewards to be attached to the actions occurring in the system specification as, e.g., we did in Chapter 10 for bonus rewards. A syntactical preprocessing step, like the one performed by the $\mathrm{EMPA}_{\mathrm{gr}_n}$ based software tool TwoTowers [19], then permits to automatically insert the rewards into the system specification. This avoids burdening the system specifications with rewards at modeling time, eases the specification of additional performance measures for the same system specification, and allows every performance measure specification to be reused for different system specifications.

Prior to our algebra based method, a different method was proposed in [57] to specify reward based performance measure in a process algebraic framework. Such a method was inspired by the preliminary work in [101], where it is proposed to use a temporal logic formula to partition the semantic model of a Markovian process algebra description in such a way that each part exhibits or not a particular behavior formalized through the logic formula itself. The idea is to define a reward structure as a function of such a partition, which associates a unique (yield) reward to all the states of the same class.

In [57] this logic based method is further elaborated on. The process of specifying performance measures is split into two stages. The first stage consists of defining a reward specification, which is a pair composed of a Hennessy-Milner logic formula [122] and an expression: every state satisfying the modal logic formula is assigned as a yield reward the value of the expression, which may consist of the usual arithmetic operators applied to real numbers, action rates, and special variables storing previously or currently assigned rewards. The second stage, instead, consists of defining a reward attachment that determines at which process derivatives a particular reward specification is evaluated. Such a method addresses only yield rewards and stationary measures.

If we compare the algebra based method and the logic based method w.r.t. the four criteria of Sect. 12.2, we see that in general the algebra based method is less powerful than the logic based method, as rewards are simply expressed as real numbers in the former method and particular behaviors formalizable through logic formulae cannot be captured (see e.g. the specification of the utilization for $QS^{ro}_{M/M/n/n}$ in Sect. 12.2), but easier to learn and use, as it does not require the knowledge of any extra formalism to specify rewards (consider e.g. the logic formula necessary to specify the mean number of customers for $QS^{ro}_{M/M/n/n}$ in Sect. 12.2). The logic based method is more time consuming than the algebra based method, as it would require in principle an additional scan of the state space in order to check states against the modal logic formulae in order to attach yield rewards: fortunately, model checking on the fly should be possible. Finally, an equational characterization is possible in the case of the algebra based method but not in the case of the logic based method, hence with the latter method a compositional, performance measure preserving term manipulation cannot be conducted. Because of the results about the relationship between bisimulation equivalence and Hennessy-Milner logic formula satisfiability [122], the logic based method only guarantees that if two terms are Markovian bisimualtion equivalent then equivalent states get the same yield reward, hence the performance index under study has the same value for the two terms. The converse does not hold: if two terms satisfy a given set of Hennessy-Milner logic formulae, then the two terms may be bisimulation equivalent but not necessarily Markovian bisimulation equivalent, which means that the value of the specified performance measures for the two terms may be different.

Recently, in [58] the lack of equational characterization for the logic based method has been remedied. This has been accomplished by using a Markovian modal logic inspired by the probabilistic modal logic of [117], instead of the Hennessy-Milner logic, and by showing that two terms satisfy the same Markovian modal logic formulae if and only if they are Markovian bisimulation equivalent. As a consequence, Markovian bisimulation equivalent states get the same reward according to the approach of [58]. In this respect, the algebra based method turns out to be more flexible, as it allows different rewards to be associated with Markovian bisimulation equivalent states, hence the need for the previously presented family of Markovian bisimulation equivalences that take rewards into account. Additionally, in [58] the ease of use of the logic based method has been enhanced by proposing a high level language for enquiring about the stationary performance characteristics possessed by a process term. Such a language, whose formal underpinning is constituted by the Markovian modal logic (which thus becomes transparent to the user), is based on the combination of the standard mathematical notation (arithmetical, relational and logical operators as well as probability), a notation based on the Markovian bisimulation equivalence which is useful to focus queries directly on states, and a notation expressing the potential to perform an action of a given type.

We conclude by mentioning that, as far as the problem of specifying performance measures is concerned, more recently a different logic based approach has been proposed in [17], which has a relationship with the Markovian bisimulation equivalence. Unlike [57, 58], where the logic is used to single out those states to which a certain yield reward must be attached, this new approach relies on the logic CSL, a continuous stochastic time variant of CTL [59], to inquiry (similarly to the high level language of [58]) about the value of stationary and transient performability measures of a process term. Based on the observation that the progress of time can be regarded as the earning of reward, a variant of CSL called CRL has been subsequently proposed in [15], where yield rewards are assumed to be attached to the states. A drawback of this approach is that the way in which yield rewards should be specified and attached to the states is not provided. We thus envision that this novel approach may be profitably integrated with the algebra based method of this chapter or the logic based method of [58].

Chapter 13

Generalized Semi-Markovian Process Algebra

The aim of this chapter is to show how to extend EMPA_{gr} so that timed actions with generally distributed durations can be expressed. In particular, we show that the techniques for representing general distributed durations in process algebra introduced in Chapters 6 and 7 based on ST semantics and preselection policy for choices can be naturally used for expressing generally distributed timed actions.

Differently from the calculus of IGSMPs introduced in Chapter 7, in the process algebra that we introduce here, called Generalized Semi-Markovian Process Algebra (GSMPA), we use ST semantics based on static names instead of dynamic names. The reason for this is not related to the fact of considering timed actions instead of untyped time delays: we could use a dynamic technique for deriving clock names also in the case of timed actions. We use static names because it allows us to obtain smaller initial semantic models for algebraic specifications and to decouple the specification of system behavior from the specification of action durations which, as we will show, is convenient when dealing with general distributions.

Moreover, with respect to the calculus of IGSMPs, we introduce the possibility of representing contemporaneous termination of timed actions in a very simple way by using step semantics for termination events, hence improving the expressivity of the resulting process algebra. In particular, this means that we can represent systems which employ really any kind of duration distributions (even non-continuous ones) and whose performance models may, strictly speaking, even not be generalized semi-Markov processes where contemporaneous termination of elements is not allowed.

On the other hand, due to the usage of static names and to the fact that we represent contemporaneous termination of timed actions, the definition of a bisimulation based equivalence for GSMPA is much more complex with respect to the calculus of IGSMPs and there is no easy way to produce an axiomatization for such an equivalence. In particular, the complexity of the equivalence mainly derives from the coexistence of probabilistic choices among clock start events and the necessity, due to the usage of a static technique, of associating names of clocks when establishing bisimulation of systems.

GSMPA is introduced together with its formal semantics, an example of performance evaluation, and the notion of bisimulation equivalence which is shown to be a congruence.

13.1 Introduction

Extending $\mathrm{EMPA_{gr}}$ to generally distributed time is very important to make it really capable of modeling real world systems. Especially because such an extension allows us to express both the real-time and the stochastic time aspects of systems, as explained in Sect. 1.

13.1.1 The Basic Idea

In this chapter we show that $\mathrm{EMPA_{gr}}$ can be naturally extended to timed actions with general distributions by using the techniques introduced in Chapters 6 and 7 based on ST semantics and preselection policy for choices. In particular, similarly as for the calculus of IGSMPs, we use semantic models which represent the execution of generally distributed timed actions by means of clocks through events of clock starts and terminations. Moreover, given an algebraic specification, we derive clock names and start and termination transitions from timed actions via ST semantics and we associate the choice information syntactically attached to the timed action (e.g. a priority level and a weight) to the start transition of the corresponding clock, thus obtaining a preselection policy for choosing among alternative timed actions. On the other hand, instead of considering ST semantics based on dynamic names as in the calculus of IGSMPs, here we consider a probabilistic variant of the static name technique introduced in Chapter 3, where the location of an action (i.e. its syntactical position with respect to parallel operators plus its type) is used as its name. Moreover we define the operational semantics in such a way that step semantics [134] is used for termination events, so to represent contemporaneous termination of timed actions. As we will show this design choice allows us to obtain smaller initial semantic models and to further increase the expressive power of the algebra at the price of a complex notion of equivalence among systems. Another drawback with respect to the calculus of IGSMPs, is that, as for the basic algebra considered in Chapter 3, we do not have an easy way to produce an axiomatization by using static names.

In order to simplify the presentation, in the language considered in this chapter we will not include priority levels and the generative-reactive synchronization mechanism, i.e. we will not distinguish between master (generative) and slave (reactive) actions as in EMPA_{gr}. Similarly as for delays of the calculus of IGSMPs, we will just implement a probabilistic preselection policy among alternative timed actions by associating a weight w with each of them. Moreover, when synchronizing timed actions of the same type via the CSP [109] parallel operator we just take the weight of the resulting action to be the product of the weights of the two initial actions, so to represent independent choices on both sides of the parallel operator. Even if simple, this approach is not very elegant, because e.g. the meaning of the semantic model originated from the usual example $<a, w'> + <b, w''> \|_{a,b} <a, w'> + <b, w''>$, where both processes want to decide on the action type to be performed, is somehow obscure. Nevertheless, simply taking the product of weights does not break the congruence property for the parallel operator and, since the treatment of probabilistic (and prioritized) choices is completely orthogonal to the management of general distributions, allow us to present a simpler language which can be easily extended with the more adequate generative-reactive synchronization mechanism and with priority levels.

13.1.2 A Language for Generally Distributed Timed Actions

In the following we informally present the main features of the language, called Generalized Semi-Markovian Process Algebra (GSMPA), that we introduce in this chapter. GSMPA is an extension of the basic process algebra introduced in Chapter 2 where action prefix "$a.\_$" is replaced by generally distributed timed action prefix "$<a, w>.\_$", where a is the action type and w is the weight associated with the action (in the following we will consider "$a.\_$" to be a shorthand for "$<a, 1>.\_$"). The general distribution associated with an action is not expressed directly in GSMPA terms as done in the calculus of IGSMPs. On the contrary, duration distributions to be considered for timed actions are separately described, similarly as in GSMPs, through functions which associate, for each sequential process composing the specified system, a general probability distribution with each action type. Decoupling the specification of system behavior from the specification of action duration distributions, which is allowed in the context of a probabilistic variant of ST semantic based on static names, although not strictly necessary, is the most natural solution when considering general distributions. This because it resembles the structure of GSMPs where duration distributions are separately associated with clocks and makes it possible to extend GSMPA with interruption mechanisms (see [42]): in the case a timed action a is interrupted, its execution can be continued by another action a of the same process (they have the same location, hence they are identified by the same clock in the semantics) which is guaranteed to have the same duration distribution. The specification of action duration distributions includes, besides the duration of local actions of sequential processes, also binary functions describing, for each action type, the synchronization policy, i.e. the way in which the duration distribution of an action obtained via synchronization must be computed from the two duration distributions of the synchronizing actions (see [108] for an overview of synchronization policies). Note that such an expressive feature is not incompatible with the introduction of a generative-reactive synchronization mechanism in GSMPA, this because, thanks to the adoption of the preselection policy, duration distributions of timed actions are not involved in the mechanism for selecting among alternative actions which is, instead, applied to transitions representing action starts. Therefore, distinguishing between master (generative) and slave (reactive) actions would only affect the way in which the choice information syntactically attached to the timed action (e.g. the priority level and the weight) is treated and, e.g., does not mean that slave actions must have an unspecified duration as for exponentially timed actions of EMPA_{gr} (see Chapter 11).

As already explained the semantics of GSMPA is based on the static name technique of Chapter 3 where global locations are used as (clock) names for timed actions. In this chapter action locations [1] are denoted as follows. We will use "$\swarrow\_$" ("$\searrow\_$") to denote that an action is positioned in the subterm to the left (to the right) of the topmost parallel composition operator and "$\langle\_|\_\rangle$" to denote that an action is obtained

[1] In the following we will refer to global locations of Chapter 3, just by using "locations".

by synchronization of the two subterms of the topmost parallel composition operator. For example in term $(a.E_1 \,\|_\emptyset\, b.E_2) \,\|_{\{b\}}\, b.E_3$: "$a_{\swarrow\swarrow}$" is the name for the action formed by the single local action identified with a executed by the leftmost sequential process, while $b_{\langle\searrow|\rangle}$ is the name for the action formed by the synchronization of the two local actions with type b. We explicitly remark that the use of locations is correct because for the purpose of deriving a clock-based representation of system behavior (similar to that used for GSMPs) it is not necessary to identify *every* action of a term as a different clock. For example two consecutive actions of a term executed by the same sequential process cannot be concurrently executed, but are causally related. Thus the two actions cannot overlap during their execution and once an action has started it must terminate before the other one can start.

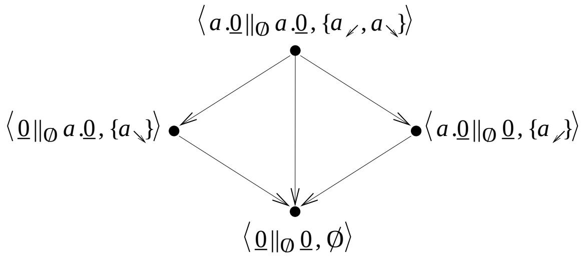
As in Chapter 3 we record in the states of semantic models the events of action starts by labeling states with the set of names of started actions. Therefore system states are represented by pairs $\langle E, \mathit{Exec}\rangle$, where E is a process term and Exec is a label expressing the set of names of actions "in execution". We recall from Chapter 3 that, since in CSP the actions obtained by synchronization are observable, we cannot simply label in some way the local action (or the local actions in the synchronization case) within the process term as done, e.g., in [4]. Consider for example the term $(a.\underline{0} \,\|_\emptyset\, a.\underline{0}) \,\|_{\{a\}} (a.\underline{0} \,\|_\emptyset\, a.\underline{0})$. In this term two actions may start execution before any action terminate. By labeling local actions inside the term we could not distinguish e.g. if the two actions that have started are $a_{\langle\swarrow|\swarrow\rangle}$ and $a_{\langle\searrow|\searrow\rangle}$, or $a_{\langle\swarrow|\searrow\rangle}$ and $a_{\langle\searrow|\swarrow\rangle}$. On the other hand our approach allows us to generate semantic models very similar to GSMPs where states are labeled with active elements, and does not require the definition of a particular syntax for state terms.

As shown in Chapter 3, the fact of considering static names instead of dynamic names significantly reduces the size of semantic models directly derived from system specifications because, while in the dynamic approach the name assigned to actions depends on the order in which actions are started, in the static approach this does not happen. For instance, while with a dynamic approach the system $a.\underline{0} \,\|_\emptyset\, a.\underline{0}$ may reach two different states where both actions a are in execution (one where the lefthand a gets name a_1 and the righthand a gets name a_2 and the other one where we are in the opposite situation), with a static approach, no matter in which order we start actions, it reaches the same state $\langle a.\underline{0} \,\|_\emptyset\, a.\underline{0}, \{a_\swarrow, a_\searrow\}\rangle$.

The semantics of GSMPA describes the system behavior also in the case of contemporaneous termination of actions. This is done very naturally by defining termination transitions as in the step semantics [134]. For example consider the following GSMPA specification:

$$a.\underline{0} \,\|_\emptyset\, a.\underline{0}$$
$$\mathit{PDF}_\swarrow(a) = \mathit{Det}(5), \qquad \mathit{PDF}_\searrow(a) = \mathit{Gauss}(0.5, 1)$$

where we express the durations of local actions by referring to the locations of the sequential processes they belong to (the lefthand a has a deterministic duration 5, while the righthand a has a Gaussian duration with mean 0.5 and variance 1). Once the two actions have started execution the system reaches the state $\langle a.\underline{0} \,\|_\emptyset\, a.\underline{0}, \{a_\swarrow, a_\searrow\}\rangle$. The evolution of this state at the semantic level is depicted below.



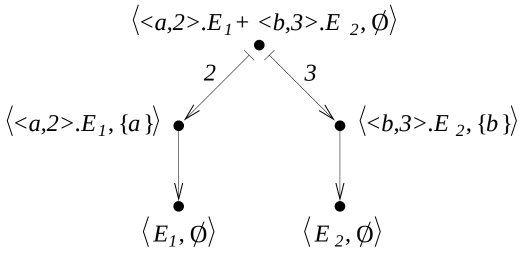
$$PDF(a_{\swarrow}) = Det(5), \qquad PDF(a_{\searrow}) = Geom(.5)$$

where the transitions presented are *termination transitions*. In semantic models the termination transitions are not labeled with terminating actions since this would be a redundant information. The actions a transition refers to can easily be derived from the state labels. It is worth noting that the three termination transitions above are assumed to be possible irregardless of duration distributions (e.g. if in the example above $PDF_{\searrow}(a) = Det(4)$, only the rightmost computation would be actually possible). This is necessary in order to achieve the congruence property when defining the notion of equivalence as we will see in Sect. 13.5. The capability of expressing contemporaneous termination of timed actions is important because it allows us to represent systems which employ really any kind of duration distributions (even non-continuous ones) and whose performance models may, strictly speaking, even not be GSMPs where contemporaneous termination of elements is not allowed. In general the stochastic processes that we obtain from GSMPA specifications, whihc we call Extended Generalized Semi-Markov Processes (EGSMPs), are GSMPs extended with contemporaneous termination (see [41] for a precise definition).

As for the calculus of IGSMPs, in GSMPA choices are solved through the preselection policy. Choices among actions are carried out by giving each of them a probability proportional to its weight and then executing the selected action. For example, consider the following GSMPA specification:

$$<a,2>.E_1 + <b,3>.E_2$$
$$PDF(a) = Det(5), \qquad PDF(b) = Gauss(0.5,1)$$

At the semantic level the system is represented as follows:



$$PDF(a) = Det(5), \qquad PDF(b) = Gauss(0.5,1)$$

where each transition $\longmapsto$ represent the event of an action start. Since the choice of an action for execution corresponds to the event of action start, the transitions $\longmapsto$ are labeled with weights and are called *start transitions*. Like termination transitions, start transitions are not labeled with the starting action since this information can easily be derived from the state labels.

13.1.3 Chapter Outline

This chapter is organized as follows. In Sect. 13.2 we show how to specify a system with GSMPA. In Sect. 13.3 we formally define the semantics for GSMPA. In Sect. 13.4 we present an example of performance evaluation: we compute the utilization of the server in a queuing system $M/D/1/2/2$ specified with GSMPA. Finally, in Sect. 13.5 we introduce a probabilistic bisimulation based notion of equivalence for GSMPA which we show to be a congruence.

13.2 Specification of a Concurrent System

The specification of a system is composed of the specification of its behavior (a term of GSMPA) and the specification of the duration of its actions. The separation of the two specifications reflects the same separation at the semantic model level stemming from the structure of GSMPs. The two separated specifications are presented in Sect. 13.2.1 and 13.2.2, respectively.

13.2.1 Specification of System Behavior

By following the approach of [30] we introduce the notation of action syntactical positions which comprises locations (denoted by $\swarrow\_$, $\searrow\_$ and $\langle\_|\_\rangle$) as well as positions with respect to the alternative position operators (denoted by $\lhd\_$ and $\rhd\_$) and applications of a constant definition (denoted by $\diamond$). The role of $\lhd\_$, $\rhd\_$ and $\diamond$ will be explained in Sect. 13.3.1.

Definition 13.1 *The set Pos of action syntactical positions is generated by the following syntax*

$$pos ::= \bullet \mid \swarrow pos \mid \searrow pos \mid \langle pos|pos\rangle \mid \lhd pos \mid \rhd pos \mid \diamond pos$$ [2]

The set Loc of action locations is the subset of Pos whose elements include only $\swarrow\_$, $\searrow\_$ and $\langle\_|\_\rangle$.

The set AId of (top level) action identifiers is defined by

$$AId = AType \times Loc$$

where AType is the set of action types ranged over by $a, b, c, \ldots$. ■

[2] In the following we omit the $\bullet$ when writing syntactical positions.

Representation of Actions

Each *action* is represented as $<a^v_{loc}, w>$ and consists of a *type* a, a *location* loc, a *visibility* v, and a *weight* w.

The type and the location of an action together constitute the *identifier* (or name) of the action as described in Sect. 13.1. The *visibility* v of an action specifies if the action must be executed internally or may synchronize with other actions. In GSMPA we cannot use a distinguished unique action type for invisible actions such as the traditional τ, because in case some actions of a process are made hidden to the environment (by means of the hiding operator "$\_/L$" presented in Sect. 13.2.1) we must preserve the identifiers of actions (which include the action types) in order to distinguish actions belonging to the internal behavior of the process so that its performance is modeled correctly. As a consequence, visibility reduces to an action attribute.

Definition 13.2 *The set AVis of action visibilities, ranged over by v, is defined by*

$$AVis = \{o, h\}$$

where o stands for observable (default value) and h stands for hidden. The set AVId of (top level) action identifiers with visibility is defined by

$$AVId = AId \times AVis$$

■

The *weight* $w \in \mathbb{R}_+$ of an action is used, as stated in Sect. 13.1, for choosing actions according to the preselection policy (the default value for the weight is 1).

Definition 13.3 *The set LAct of local actions is defined by*

$$LAct = AType \times AVis \times \mathbb{R}_+$$

and the set Act of (top-level) actions is defined by

$$Act = AVId \times \mathbb{R}_+$$

■

Syntax of Terms and Informal Semantics of Operators

Let *Const* be a set of *constants* ranged over by $A, B, C, \ldots$, and let $ARFun = \{\varphi : AType \longrightarrow AType\}$ be a set of *action relabeling functions*.

Definition 13.4 *The set $\mathcal{L}$ of process terms of GSMPA is generated by the following syntax*

$$E ::= \underline{0} \mid <a^v, w>.E \mid E/L \mid E[\varphi] \mid E + E \mid E \parallel_S E \mid A$$

where $<a^v, w> \in LAct$ and $L, S \subseteq AType$. Set $\mathcal{L}$ will be ranged over by $E, E', E'', \ldots$. We denote by $\mathcal{G}$ the set of closed and guarded terms of $\mathcal{L}$. ■

The *null term* $\underline{0}$ has the usual meaning of termination.

The *prefix operator* "$<a^v, w>.\_$" denotes the sequential composition of an action and a term: term $<a^v, w>.E$ can execute a local action with type a, visibility v and weight w, and behaves as term E after the completion of the action.

The *hiding operator* "$\_/L$" changes the visibility of the actions: term E/L behaves as term E except that actions whose type belong to L get visibility h.

The *relabeling operator* "$\_[\varphi]$" modifies the type of the actions: term $E[\varphi]$ behaves as term E except that the type of each executed action is modified according to φ. Note that also the type of hidden actions can be changed by this operator.

The *alternative composition operator* "$\_ + \_$" expresses a choice between two terms: term $E_1 + E_2$ behaves as either term E_1 or term E_2 depending on whether an action of E_1 or an action of E_2 is selected for execution. The choice is made according to the preselection policy, hence the weights of the executable actions of the two terms are taken into account.

The *parallel composition operator* "$\_ \|_S \_$" expresses the concurrent execution of two terms according to the CSP [109] synchronization policy, thus the two terms must synchronize when executing visible actions of the same type belonging to S, whilst they must execute independently all the other actions. Given term $E_1 \|_S E_2$, in case of synchronization of a visible action of E_1 with type $a \in S$ and location loc_1 and a visible action of E_2 with the same type a and location loc_2, the resulting action is visible, has the same type a, has location $\langle loc_1 | loc_2 \rangle$, and has weight given by the *product* of the original weights [150]. In case of an action executed independently by E_1 (E_2) which is invisible or visible with type $a \notin S$ and has location loc_1 (loc_2), the resulting action has obviously the same type, has location $\swarrow loc_1$ ($\searrow loc_2$) and the other attributes are kept unchanged.

Finally, *constant* A behaves like the only term E such that $A \stackrel{\Delta}{=} E$.

13.2.2 Specification of Action Durations

The durations of the actions are specified in a compositional way. The system specification must include the description of the duration of every local action of every sequential process as well as the description of the way in which the duration of an action resulting from the synchronization of several actions is computed. We identify sequential processes through locations just as we do for actions.

Definition 13.5 *The set PId of process identifiers is generated by the following syntax*

$$pid ::= \bullet \mid pid\swarrow \mid pid\searrow$$ [3]

■

[3] Contrary to action locations, process locations are defined through a left recursive syntax since this is convenient for the definition of the function *PDFCalc* of Sect. 13.3.2. In the following we omit the $\bullet$ when writing process identifiers so that we express a location in the same way for both actions and processes.

If we denote by *Dist* the set of probability distribution functions, the specification of durations includes:

- For every sequential process *pid* a function

$$PDF_{pid} : AType \longrightarrow Dist$$

  that associates with every local action of process *pid* a duration distribution.

- For any action type $a \in AType$ a function

$$PDFSynch_a : (Dist \times Dist) \longrightarrow Dist$$

  that determines the duration of an action derived by the synchronization of two actions of type a. This operator determines the synchronization paradigm used for action type a and can be *arbitrary* (at least as long as we are able to do calculations). A suitable function, however, should be at least commutative and associative.

When specifying the durations of a concurrent system described by a term E, what is actually necessary is to specify the durations of the local actions executable by the sequential processes of E and to specify the synchronization paradigm for the action types belonging to the synchronization sets occurring in E.

13.3 Integrated Semantics for GSMPA

In this section we define the integrated semantics for GSMPA. The structure of this section parallels the structure of the previous one in order to take the separate specification of behaviors and durations into account.

13.3.1 Representation of System Behavior

We now define the formal integrated semantics for GSMPA according to Sect. 13.2.1 in the form of a labeled transition system (LTS) with two types of transitions.

States of the LTS

As we have already sketched in Sect. 13.1 the *states* of the LTS are represented by pairs $\langle E, Exec \rangle$ consisting of a GSMPA term E and a set of identifiers with visibility $Exec \subseteq AVId$ which single out the actions in execution. By analogy with GSMPs, states are observable and their observable part is the set of actions in execution.

There are some constraints that the pair $\langle E, Exec \rangle$ must satisfy in order to be considered as a legal state. In order to define these constraints we introduce the two auxiliary functions *act* and *share*.

$$act(\underline{0}) = \emptyset$$

$$act(<a^v, w>.E) = \{a^v\}$$

$$act(E/L) = \{a^h_{pos} \mid a^v_{pos} \in act(E) \wedge a \in L\} \cup \{a^v_{pos} \in act(E) \mid a \notin L\}$$

$$act(E[\varphi]) = \{\varphi(a)^v_{pos} \mid a^v_{pos} \in act(E)\}$$

$$act(E_1 + E_2) = \{a^v_{\lhd pos} \mid a^v_{pos} \in act(E_1)\} \cup \{a^v_{\rhd pos} \mid a^v_{pos} \in act(E_2)\}$$

$$\begin{aligned} act(E_1 \parallel_S E_2) = {} & \{a^v_{\swarrow pos} \mid a^v_{pos} \in act(E_1) \wedge (a \notin S \vee v = h)\} \cup \\ & \{a^v_{\searrow pos} \mid a^v_{pos} \in act(E_2) \wedge (a \notin S \vee v = h)\} \cup \\ & \{a^o_{\langle pos' \mid pos'' \rangle} \mid a^o_{pos'} \in act(E_1) \wedge a^o_{pos''} \in act(E_2) \wedge a \in S\} \end{aligned}$$

$$act(A) = \{a^v_{\diamond pos} \mid a^v_{pos} \in act(E)\} \qquad \text{if } A \triangleq E$$

$$share(\bullet, \bullet)$$

$$share(\swarrow pos', \swarrow pos'') \Leftarrow share(pos', pos'')$$

$$share(\langle pos' | pos \rangle, \swarrow pos'') \Leftarrow share(pos', pos'')$$

$$share(\searrow pos', \searrow pos'') \Leftarrow share(pos', pos'')$$

$$share(\langle pos | pos' \rangle, \searrow pos'') \Leftarrow share(pos', pos'')$$

$$share(\langle pos'_1 | pos'_2 \rangle, \langle pos''_1 | pos''_2 \rangle) \Leftarrow share(pos'_1, pos''_1) \vee share(pos'_2, pos''_2)$$

Table 13.1: Function *act* and predicate *share*

Definition 13.6 *The function act* : $\mathcal{G} \longrightarrow \mathcal{P}(AType \times Pos \times AVis)$ *that evaluates the set of executable actions of a GSMPA term is defined in the first part of Table 13.1.* ■

Definition 13.7 *The predicate share over* $\{(loc', loc'') \in Loc \times Loc \mid \exists E \in \mathcal{G}; a, b \in AType; v', v'' \in AVis : a^{v'}_{loc'}, b^{v''}_{loc''} \in act(E)\}$*, that determines if two actions executable by a term, which have syntactical position coinciding with location, share a local action, is defined in the second part of Table 13.1.* ■

Definition 13.8 *The set* $LabS \subseteq \mathcal{G} \times \mathcal{P}(AVId)$ *of GSMPA states is defined by letting* $\langle E, Exec \rangle \in LabS$ *if and only if*

- $a^v_{loc} \in Exec \Rightarrow a^v_{loc} \in act(E)$
- $a^{v'}_{loc'}, b^{v''}_{loc''} \in Exec \Rightarrow \neg share(loc', loc'')$ ■

The first condition assures that each action in execution is an action executable by E whose syntactical position coincides with its location. This because when an action is chosen for execution: (i) all alternative behaviors are eliminated (together with operators "+"), so $\lhd$ and $\rhd$ cannot appear in its syntactical position, (ii) involved constants are replaced by their defining equations, so $\diamond$ cannot appear in its syntactical position.

The second condition assures that actions in execution do not share local actions. This because in a GSMPA state only actions which are not composed of local actions belonging to an action already in execution are considered to be selectable for beginning execution.

Transitions of the LTS

The two types of transition of the LTS are related to the two events of action beginning and action ending.

Start transitions represent the choice of actions for execution according to the preselection policy and are labeled with the weight w associated with the chosen action. They are represented by the transition relation $\longmapsto$ defined as the least subset of $LabS \times \mathbb{R}_+ \times LabS$ that satisfies the inference rule in the first part of Table 13.2. This rule determines the transitions leaving a state $\langle E, Exec\rangle$ beginning from the multiset of start moves of the state $SM(\langle E, Exec\rangle)$. As in [19] we represent the start moves of a state by a multiset since several moves with the same weight and the same derivative state may be inferred. This multiset is defined by structural induction as the least element of $\mathcal{M}u_{fin}(L_C \times LabS)$ satisfying the rules in the second part of Table 13.2. [4] A start move represents the choice of a single action and is a pair composed of the weight of the action and the state reached after the choice. Note that in the definition of $SM(\langle E, Exec\rangle)$ we consider only the sets $Exec$ such that $\langle E, Exec\rangle \in LabS$. The transition relation is determined from $SM(\langle E, Exec\rangle)$ through function $melt : \mathcal{M}u_{fin}(L_C \times LabS) \longrightarrow \mathcal{P}_{fin}(L_C \times LabS)$, defined in the third part of Table 13.2, which merges together the start moves with the same derivative state by summing their weights. The auxiliary functions $left : \mathcal{P}(AVId) \longrightarrow \mathcal{P}(AVId)$, $right : \mathcal{P}(AVId) \longrightarrow \mathcal{P}(AVId)$, $hide : \mathcal{P}(AVId) \longrightarrow \mathcal{P}(AVId)$, and $relab : \mathcal{P}(AVId) \longrightarrow \mathcal{P}(AVId)$ defined in the fourth part of Table 13.2 are employed in order to help the compositional definition of SM.

Termination transitions represent the contemporaneous termination of a set of actions in execution according to the race policy. They are represented by the transition relation $\longrightarrow$ defined as the least subset of $LabS \times LabS$ that satisfies the inference rule in the first part of Table 13.3. This rule determines a transition leaving a state $\langle E, Exec\rangle$ for each nonempty subset Ter of $Exec$ using the function $TM(\langle E, Exec\rangle) : \mathcal{P}(Exec) - \{\emptyset\} \longrightarrow \mathcal{G}$ that describes the termination moves of the state. Since there is one and only one derivative term for each nonempty subset of $Exec$ we represent the termination moves of a state by a function. This function is defined in the second part of Table 13.3 by structural induction on states with a nonempty set $Exec$.

[4] We denote by $\mathcal{M}u_{fin}(S)$ the set of finite multiset over S, we use $\{\!|$ and $|\!\}$ as multiset parentheses, and we use $\oplus$ to denote multiset union.

$$\frac{(w, \langle E', \mathit{Exec}' \rangle) \in \mathit{melt}(\mathit{SM}(\langle E, \mathit{Exec} \rangle))}{\langle E, \mathit{Exec} \rangle \stackrel{w}{\longmapsto} \langle E', \mathit{Exec}' \rangle}$$

$$\mathit{SM}(\langle \underline{0}, \emptyset \rangle) = \emptyset$$

$$\mathit{SM}(\langle <a^v, w>.E, \emptyset \rangle) = \{\!|\, (w, \langle <a^v, w>.E, \{a^v\} \rangle) \,|\!\}$$

$$\mathit{SM}(\langle <a^v, w>.E, \{a^v\} \rangle) = \emptyset$$

$$\begin{aligned} \mathit{SM}(\langle E/L, \mathit{hide}(\mathit{Exec}, L) \rangle) = \{\!|\, & (w, \langle E'/L, \mathit{hide}(\mathit{Exec} \cup \{a^v_{loc}\}, L) \rangle) \mid \\ & (w, \langle E', \mathit{Exec} \cup \{a^v_{loc}\} \rangle) \in \mathit{SM}(\langle E, \mathit{Exec} \rangle) \,|\!\} \end{aligned}$$

$$\begin{aligned} \mathit{SM}(\langle E[\varphi], \mathit{relab}(\mathit{Exec}, \varphi) \rangle) = \{\!|\, & (w, \langle E'[\varphi], \mathit{relab}(\mathit{Exec} \cup \{a^v_{loc}\}, \varphi) \rangle) \mid \\ & (w, \langle E', \mathit{Exec} \cup \{a^v_{loc}\} \rangle) \in \mathit{SM}(\langle E, \mathit{Exec} \rangle) \,|\!\} \end{aligned}$$

$$\mathit{SM}(\langle E_1 + E_2, \emptyset \rangle) = \mathit{SM}(\langle E_1, \emptyset \rangle) \oplus \mathit{SM}(\langle E_2, \emptyset \rangle)$$

$$\begin{aligned} \mathit{SM}(\langle E_1 \parallel_S E_2, \mathit{Exec} \rangle) = {} & \{\!|\, (w, \langle E_1' \parallel_S E_2, \mathit{Exec} \cup \{a^v_{\swarrow\, loc}\} \rangle) \mid (a \notin S \vee v = h) \wedge \\ & \quad (w, \langle E_1', \mathit{left}(\mathit{Exec}) \cup \{a^v_{loc}\} \rangle) \in \mathit{SM}(\langle E_1, \mathit{left}(\mathit{Exec}) \rangle) \,|\!\} \oplus \\ & \{\!|\, (w, \langle E_1 \parallel_S E_2', \mathit{Exec} \cup \{a^v_{\searrow\, loc}\} \rangle) \mid (a \notin S \vee v = h) \wedge \\ & \quad (w, \langle E_2', \mathit{right}(\mathit{Exec}) \cup \{a^v_{loc}\} \rangle) \in \mathit{SM}(\langle E_2, \mathit{right}(\mathit{Exec}) \rangle) \,|\!\} \oplus \\ & \{\!|\, (w, \langle E_1' \parallel_S E_2', \mathit{Exec} \cup \{a^o_{<loc' \mid loc''>}\} \rangle) \mid a \in S \wedge \\ & \quad (w', \langle E_1', \mathit{left}(\mathit{Exec}) \cup \{a^o_{loc'}\} \rangle) \in \mathit{SM}(\langle E_1, \mathit{left}(\mathit{Exec}) \rangle) \wedge \\ & \quad (w'', \langle E_2', \mathit{right}(\mathit{Exec}) \cup \{a^o_{loc''}\} \rangle) \in \mathit{SM}(\langle E_2, \mathit{right}(\mathit{Exec}) \rangle) \wedge \\ & \quad w = w' \cdot w'' \,|\!\} \end{aligned}$$

$$\mathit{SM}(\langle A, \emptyset \rangle) = \mathit{SM}(\langle E, \emptyset \rangle) \qquad A \stackrel{\Delta}{=} E$$

$$\begin{aligned} \mathit{melt}(\mathit{SM}) = \{ & (w, \langle E, \mathit{Exec} \rangle) \mid \exists w' : (w', \langle E, \mathit{Exec} \rangle) \in \mathit{SM} \wedge \\ & w = \textstyle\sum \{\!|\, w'' \mid (w'', \langle E, \mathit{Exec} \rangle) \in \mathit{SM} \,|\!\} \} \end{aligned}$$

$$\mathit{left}(\mathit{vids}) = \{a^v_{loc} \mid a^v_{\swarrow loc} \in \mathit{vids} \vee \exists loc' : a^v_{\langle loc | loc' \rangle} \in \mathit{vids}\}$$

$$\mathit{right}(\mathit{vids}) = \{a^v_{loc} \mid a^v_{\searrow loc} \in \mathit{vids} \vee \exists loc' : a^v_{\langle loc' | loc \rangle} \in \mathit{vids}\}$$

$$\mathit{hide}(\mathit{vids}, L) = \{a^h_{loc} \mid a^v_{loc} \in \mathit{vids} \wedge a \in L\} \cup \{a^v_{loc} \in \mathit{vids} \mid a \notin L\}$$

$$\mathit{relab}(\mathit{vids}, \varphi) = \{\varphi(a)^v_{loc} \mid a^v_{loc} \in \mathit{vids}\}$$

Table 13.2: Rules for start transitions

$$\frac{TM(\langle E, Exec\rangle)(Ter) = E'}{\langle E, Exec\rangle \longrightarrow \langle E', Exec - Ter\rangle} \quad Ter \subseteq Exec,\ Ter \neq \emptyset$$

$$TM(\langle <a^v, w>.E, \{a^v\}\rangle)(\{a^v\}) = E$$

$$TM(\langle E/L, hide(Exec, L)\rangle)(hide(Ter, L)) = TM(\langle E, Exec\rangle)(Ter)/L$$

$$TM(\langle E[\varphi], relab(Exec, \varphi)\rangle)(relab(Ter, \varphi)) = TM(\langle E, Exec\rangle)(Ter)[\varphi]$$

$$TM(\langle E_1 \parallel_S E_2, Exec\rangle)(Ter) = E_1' \parallel_S E_2'$$

where :

$$E_1' \equiv \begin{cases} TM(\langle E_1, left(Exec)\rangle)(left(Ter)) & \text{if } left(Ter) \neq \emptyset \\ E_1 & \text{if } left(Ter) = \emptyset \end{cases}$$

$$E_2' \equiv \begin{cases} TM(\langle E_2, right(Exec)\rangle)(right(Ter)) & \text{if } right(Ter) \neq \emptyset \\ E_2 & \text{if } right(Ter) = \emptyset \end{cases}$$

Table 13.3: Rules for termination transitions

13.3.2 Computation of Action Durations

The function $PDF : AId \longrightarrow Dist$ that assigns to each identifier its distribution of duration is computed as follows

$$PDF(a_{loc}) = PDFCalc(a_{loc}, \bullet)$$

where function $PDFCalc : (AId \times PId) \longrightarrow Dist$ deals with functions $PDFSynch_a$ and PDF_{pid}, defined at the specification level, in the following way: [5]

$$\begin{aligned} PDFCalc(a_{\langle loc_1 | loc_2\rangle}, pid) &= PDFSynch_a(PDFCalc(a_{loc_1}, pid\swarrow), PDFCalc(a_{loc_2}, pid\searrow)) \\ PDFCalc(a_{\swarrow loc}, pid) &= PDFCalc(a_{loc}, pid\swarrow) \\ PDFCalc(a_{\searrow loc}, pid) &= PDFCalc(a_{loc}, pid\searrow) \\ PDFCalc(a, pid) &= PDF_{pid}(a) \end{aligned}$$

[5] We assume the two functions $PDFSynch_a$ and PDF_{pid} to be always defined.

13.3.3 Definition of the Integrated Semantics

Definition 13.9 *The integrated semantics of $E \in \mathcal{G}$ is given by the following tuple composed of a LTS and a set of duration distributions*

$$\mathcal{I}[\![E]\!] = (S_E, \longrightarrow_E, \longmapsto_E, r_E; PDF_E)$$

where S_E is the set of states reachable from the initial state $r_E = \langle E, \emptyset \rangle$ via transitions in $\longmapsto_E$, which is the restriction of $\longmapsto$ to $S_E \times \mathbb{R}_+ \times S_E$, and in $\longrightarrow_E$, which is the restriction of $\longrightarrow$ to $S_E \times S_E$. PDF_E is the restriction of function PDF to the set $Ids_E = \{a_{loc} \mid \exists \langle E, Exec \rangle \in S_E, v \in AVis : a^v_{loc} \in Exec\}$ of the identifiers of the actions that may be executed by term E. ■

13.3.4 Functional and Performance Semantics

From the integrated model we can derive by projection a functional model and a performance model. The functional model is obtained by removing the quantitative information related to duration of actions and probability of choices. The performance model is an extended generalized semi-Markov process, i.e. a GSMP extended with contemporaneous termination of elements (see [42]), obtained by abstracting from functional information, i.e. from action types and locations.

13.4 A Simple Example: Queuing Systems $M/D/1/2/2$

A queuing system (QS for short) [115] is a model largely used for performance evaluation purposes to represent a service center composed of a waiting queue and a given number of servers, which provides a certain service (following a given discipline) to the customers arriving at the service center.

Here we consider a QS $M/D/1/2/2$ with two classes of customers: it is composed of a single server and a FIFO queue with capacity 1, providing service to two customers each belonging to a different class. The first (second) customer stays outside the system for a time exponentially distributed with some parameter λ (μ), then requires service. Service times have a deterministic duration k. If we represent by w the waiting of a customer outside the system, by e the event of a customer entering the system, by d the delivery of a customer to the server, by s the service of a customer, and by l the event of a customer leaving the system, such a QS can be modeled as follows with GSMPA by compositionally specifying behaviors and durations:

- $QS_{M/D/1/2/2} \stackrel{\Delta}{=} (Customer \,\|_\emptyset\, Customer) \,\|_{\{e,l\}} (Queue \,\|_{\{d\}}\, Server)$

  $PDFSynch_e = PDFSynch_l = PDFSynch_d = \max$ [6]

  - $Customer \stackrel{\Delta}{=} w.e.l.Customer$

[6] Operator max computes the pointwise maximum of two functions.

$$PDF_{\swarrow\swarrow}(w) = Exp(\lambda), PDF_{\swarrow\swarrow}(e) = PDF_{\swarrow\swarrow}(l) = Det(0)$$
$$PDF_{\swarrow\searrow}(w) = Exp(\mu), PDF_{\swarrow\searrow}(e) = PDF_{\swarrow\searrow}(l) = Det(0)$$

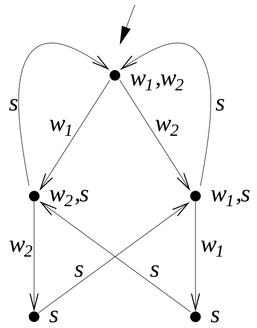
– $Queue \triangleq e.d.Queue$

$$PDF_{\searrow\swarrow}(e) = PDF_{\searrow\swarrow}(d) = Det(0)$$

– $Server \triangleq d.s.l.Server$

$$PDF_{\searrow\searrow}(s) = Det(k), PDF_{\searrow\searrow}(d) = PDF_{\searrow\searrow}(l) = Det(0)$$

Suppose we want to compute the utilization of the server. The first step consists of deriving by projection a GSMP from $\mathcal{I}[\![QS_{M/D/1/2/2}]\!]$. The resulting GSMP is depicted below.



$$PDF(w_1) = Exp(\lambda), PDF(w_2) = Exp(\mu), PDF(s) = Det(k)$$

where element w_1 corresponds to action $w_{\swarrow\swarrow}$, element w_2 to action $w_{\swarrow\searrow}$, and element s to action $s_{\searrow\searrow}$. Note that actions e, d, and l do not occur since the GSMP states with zero sojourn time have been eliminated (see [42] for a precise definition of this procedure).

This GSMP is not insensitive due to the behavior of element s in the states labeled with w_1, s and w_2, s. By applying the technique of state amalgamation presented in [93] we merge the state labeled with w_1, s with the state labeled with s it reaches when w_1 terminates, and the state labeled with w_2, s with the state labeled with s it reaches when w_2 terminates. In this way we obtain an insensitive GSMP which in this case turns out to be even an SMP. As we can substitute an exponential distribution with rate k^{-1} for the deterministic duration k, we obtain a MC. Finally, to compute the utilization of the server we resort to the technique of rewards (see Chapter 2). With this technique the performance measure of interest is the weighted sum of the steady state probabilities of the states of the Markov chain. In our case we have to single out those states in which the server is providing service. This is accomplished by simply assigning reward 1 to the states whose label includes s. If e.g. $\lambda = 0.2$, $\mu = 0.1$, and $k = 3$, the utilization turns out to be 57%.

13.5 A Notion of Equivalence for GSMPA

In the following we propose a notion of integrated equivalence for GSMPA which combines a bisimulation in the form of that presented in Chapter 3 for the static name technique with the notion of *probabilistic bisimulation* proposed in [117]. According to [117], a probabilistic bisimulation must be an equivalence relation such that two bisimilar terms have the same aggregated probability to reach the same equivalence class. In the case of GSMPA, such a notion must be refined in order to take durations into account. This was a relatively easy task for Markovian process algebras (see Chapter 11) thanks to the memoryless property of exponential distributions. Now, since general distributions are involved, we have to deal with the fact that the sojourn time in a state depends not only on the duration distributions of the actions in execution in that state, but also on the spent lifetimes of such actions. Consequently, the best we can hope for is to equate some pairs of states whenever we can rely on a correspondence between their sets of actions in execution (following the same approach as in the static name technique of Chapter 3) such that matching actions have the same duration distribution. In order to describe such a correspondence, we define the notion of consistent action association which relates actions by abstracting from their locations.

Definition 13.10 *A relation $\psi \subseteq AVId \times AVId$ is a consistent location association if and only if there exist $vids_1, vids_2 \subseteq AVId$ such that ψ is a bijection from $vids_1$ to $vids_2$ and*

$$(a^{v_1}_{loc_1}, b^{v_2}_{loc_2}) \in \psi \Rightarrow (a = b \wedge v_1 = v_2 \wedge PDF(a_{loc_1}) = PDF(b_{loc_2}))$$

We let Ψ be the set of all consistent location associations. ■

As in the static name technique of Chapter 3 we deal with a family of bisimulations each indexed by a location association ψ. A bisimulation $\mathcal{B}_\psi$ equates two states s_1 and s_2 ($s_1 \, \mathcal{B}_\psi \, s_2$), whose actions in execution have been matched according to ψ, whenever they behave the same according to probabilistic bisimulation as far as start transitions are concerned, and classical bisimulation as far as termination transitions are concerned. More precisely two states are equated if they have: (i) the same aggregated weight to reach corresponding equivalence classes by choosing actions having the same type, the same visibility, and the same duration distribution, where classes are related by a bisimulation indexed by ψ augmented with a pair consisting of two chosen actions, (ii) the same possibility to reach corresponding equivalence classes by terminating a set of actions which are pointwise matched in ψ, where classes are related by a bisimulation indexed by ψ diminished by the terminated actions.

Before defining the equivalence we give some auxiliary definitions.

13.5.1 Preliminary Definitions

A relation $\mathcal{B}$ over a set A can be seen as representing a correspondence among the elements of two copies of the same set A in this way: whenever $a \, \mathcal{B} \, b$, a is an element of the first copy of A and b is an element of the second copy of A. If transitivity is guaranteed such a relation can be seen as representing a bijective mapping between classes of elements of the first copy of A and classes of elements of the second copy of A.

Definition 13.11 *Given a set A we say that a relation $\mathcal{B} \subseteq A \times A$ is a class mapping over A, denoted $\mathcal{B} \in cmap(A)$, if and only if*

$$\mathcal{B} = \mathcal{B} \circ \mathcal{B}^{-1} \circ \mathcal{B}$$

■

Proposition 13.1 *Given a set A and a relation $\mathcal{B} \subseteq A \times A$ we have that $\mathcal{B} \in cmap(A)$ if and only if either $\mathcal{B} = \emptyset$ or there is a unique way to find $A_1, A_2 \subseteq A$, π_1 partition of A_1, π_2 partition of A_2, and a bijection $f : \pi_1 \longrightarrow \pi_2$ such that:*

$$\mathcal{B} = \bigcup_{(C_1,C_2)\in f} C_1 \times C_2$$

Proof: ($\Rightarrow$) Defined $\mathcal{B}(a) = \{b \mid a\,\mathcal{B}\,b\}$, we let $\pi_1 = \{\mathcal{B}^{-1}(b) \mid \mathcal{B}^{-1}(b) \neq \emptyset \wedge b \in A\}$ and $\pi_2 = \{\mathcal{B}(a) \mid \mathcal{B}(a) \neq \emptyset \wedge a \in A\}$. If we let $A_1 = \bigcup_{A' \in \pi_1} A'$ and $A_2 = \bigcup_{A' \in \pi_2} A'$ and we suppose that $\mathcal{B} \neq \emptyset$ we have that π_1 is a partition of $A_1 \neq \emptyset$ and π_2 is a partition of $A_2 \neq \emptyset$. The latter statement can be shown as follows (the proof of the former is completely similar). From $\mathcal{B}(a) \cap \mathcal{B}(a') \neq \emptyset$ we derive that there exists b such that $a\,\mathcal{B}\,b$ and $a'\,\mathcal{B}\,b$. Since $\mathcal{B}$ is a class mapping over A, it follows that: $b' \in \mathcal{B}(a)$ (i.e. $b'\mathcal{B}a$) if and only if $b' \in \mathcal{B}(a')$ (i.e. $b'\mathcal{B}a'$). Therefore $\mathcal{B}(a) = \mathcal{B}(a')$.

Finally let us define the function $f : \pi_1 \longrightarrow \pi_2$ as follows: $f(\mathcal{B}^{-1}(b)) = \mathcal{B}(a)$ for any $a \in \mathcal{B}^{-1}(b)$ (i.e. $a\mathcal{B}b$). The function f is well defined since from $a\,\mathcal{B}\,b$ and $a'\,\mathcal{B}\,b$ we derive $\mathcal{B}(a) \cap \mathcal{B}(a') \neq \emptyset$, hence $\mathcal{B}(a) = \mathcal{B}(a')$. The function f is injective since from $f(\mathcal{B}^{-1}(b)) = \mathcal{B}(a)$, $f(\mathcal{B}^{-1}(b')) = \mathcal{B}(a')$ and $\mathcal{B}(a) = \mathcal{B}(a')$ we derive that $a\,\mathcal{B}\,b$ implies $a'\,\mathcal{B}\,b$, hence $a' \in \mathcal{B}^{-1}(b) \cap \mathcal{B}^{-1}(b')$; it follows that $\mathcal{B}^{-1}(b) = \mathcal{B}^{-1}(b')$. The function f is surjective since from $\mathcal{B}(a) \in \pi_2$ and $b \in \mathcal{B}(a)$ it follows that $f(\mathcal{B}^{-1}(b)) = \mathcal{B}(a)$. f^{-1} results obviously as follows: $f^{-1}(\mathcal{B}(a)) = \mathcal{B}^{-1}(b)$ for any $b \in \mathcal{B}(a)$ (i.e. $a\mathcal{B}b$).

Finally since (i) $(a', b') \in \mathcal{B}^{-1}(b) \times \mathcal{B}(a)$ and $f(\mathcal{B}^{-1}(b)) = \mathcal{B}(a)$ implies $\mathcal{B}(a') = \mathcal{B}(a)$, hence $a'\,\mathcal{B}\,b'$, and (ii) $a'\,\mathcal{B}\,b'$ implies $(a', b') \in \mathcal{B}^{-1}(b') \times \mathcal{B}(a')$ with $f(\mathcal{B}^{-1}(b')) = \mathcal{B}(a')$; it follows that $\mathcal{B} = \bigcup_{(C_1,C_2)\in f} C_1 \times C_2$.

($\Leftarrow$) If $\mathcal{B} = \emptyset$ then obviously $\mathcal{B} \in cmap(A)$. Otherwise we show that $\mathcal{B} = \bigcup_{(C_1,C_2)\in f} C_1 \times C_2$ is a class mapping over A as follows. From $a \;\; \mathcal{B} \circ \mathcal{B}^{-1} \;\; b$ and $b\,\mathcal{B}\,c$ we derive that there exist $(C_1, C_2) \in f$ such that $a, b \in C_1$ and $c \in C_2$. Therefore $(a, c) \in C_1 \times C_2 \subseteq \mathcal{B}$. □

Given a class mapping $\mathcal{B}$ of a set A, we define $crel(\mathcal{B})$ as the extension to the whole A of the class bijection f induced by $\mathcal{B}$.

Definition 13.12 *Given a set A and $\mathcal{B} \in cmap(A)$ we define the class relation associated with $\mathcal{B}$, denoted $crel(\mathcal{B}) \subseteq \mathcal{P}(A) \times \mathcal{P}(A)$ as follows:*

- *if $\mathcal{B} = \emptyset$ then $crel(\mathcal{B}) = \emptyset$,*

- *otherwise* $crel(\mathcal{B}) = f \cup \{(A - A_1, \emptyset), (\emptyset, A - A_2)\} - \{(\emptyset, \emptyset)\}$ *where* f, A_1 *and* A_2 *are as in the previous proposition.* ■

13.5.2 Generalized Semi-Markovian Bisimulation Equivalence

We are now in a position to define the equivalence for GSMPA. Let us first define some auxiliary functions.

Definition 13.13 *We define function tweight* : $LabS \times \mathcal{P}(LabS) \longrightarrow \mathbb{R}$, *the total weight with which a given state reaches a given set of states, as follows:*

$$tweight(s, C) = \sum \{\!| \, w \mid \exists s' \in C : s \stackrel{w}{\longmapsto} s' \, |\!\}$$

Besides let ter be a predicate over $LabS \times \mathcal{P}(LabS)$ *that determines whether a given state reaches a given set of states through a termination transition, defined by:*

$$ter(s, C) \Leftrightarrow \exists s' \in C : s \longrightarrow s'$$

■

Definition 13.14 *Given a* Ψ*-indexed family* $\mathbf{B} = \{\mathcal{B}_\psi \subseteq LabS \times LabS \mid \psi \in \Psi\}$ *of relations over* $LabS$ *and a location association* $\psi \in \Psi$, *we define* $stmap(\mathbf{B}, \psi)$, *the start mapping related to the location association* ψ, *and* $temap(\mathbf{B}, \psi)$, *the termination mapping related to the location association* ψ, *as follows:*

$$stmap(\mathbf{B}, \psi) = \bigcup_{\substack{a^v_{loc_1}, a^v_{loc_2} \in AVId \\ PDF(a_{loc_1}) = PDF(a_{loc_2})}} \mathcal{B}_{\psi \cup \{a^v_{loc_1}, a^v_{loc_2}\}}$$

$$temap(\mathbf{B}, \psi) = \bigcup_{nter \subset dom(\psi)} \mathcal{B}_{\psi|_{nter}}$$ [7]

Whenever stmap and temap are class mapping over $LabS$, *we denote with strel the class relation* $crel(stmap)$ *and with terel the class relation* $crel(temap)$. ■

Definition 13.15 *A strong generalized semi-Markovian bisimulation family (strong GSMBF) is a* Ψ*-indexed family* $\mathbf{B} = \{\mathcal{B}_\psi \subseteq LabS \times LabS \mid \psi \in \Psi\}$ *of relations over* $LabS$ *such that:*

- $I_{\{\langle E, Exec\rangle \mid E \in \mathcal{G}\}} \subseteq \mathcal{B}_{I_{Exec}} \qquad \forall\, Exec \subseteq AVId$ [8]
- $s_1 \, \mathcal{B}_\psi \, s_2 \Rightarrow s_2 \, \mathcal{B}_{\psi^{-1}} \, s_1 \qquad \forall\, \psi \in \Psi$
- $s_1 \, \mathcal{B}_{\psi'} \, s_2 \wedge s_2 \, \mathcal{B}_{\psi''} \, s_3 \Rightarrow s_1 \, \mathcal{B}_{\psi' \circ \psi''} \, s_3 \qquad \forall\, \psi', \psi'' \in \Psi$
- $(s_1, s_2) \in \mathcal{B}_\psi \Rightarrow$

[7] $nter$ denotes a set of nonterminating actions and $\psi\,|_{nter}$ the restriction of ψ to the set $nter$.

[8] I_A denotes the identity relation over set A and $I_\emptyset = \emptyset$.

– $s_1 \equiv \langle E_1, dom(\psi)\rangle \wedge s_2 \equiv \langle E_2, range(\psi)\rangle$

– $\forall\, (C_1, C_2) \in strel(\mathbf{B}, \psi).\, tweight(s_1, C_1) = tweight(s_2, C_2)$

– $\forall\, (C_1, C_2) \in terel(\mathbf{B}, \psi).\, ter(s_1, C_1) = ter(s_2, C_2)$ ■

It is easy to see that some consequences of the first three conditions are that: $\forall \psi \in \Psi$, $\mathcal{B}_\psi \in cmap(LabS)$, $stmap(\mathbf{B}, \psi) \in cmap(LabS)$, $temap(\mathbf{B}, \psi) \in cmap(LabS)$; $\forall Exec \subseteq AVId$, $\mathcal{B}_{I_{Exec}}$ is an equivalence relation over $\{\langle E, Exec\rangle \in LabS \mid E \in \mathcal{G}\}$, $stmap(\mathbf{B}, I_{Exec})$ is an equivalence relation over $\{\langle E, Exec \cup \{a^v_{loc}\}\rangle \in LabS \mid E \in \mathcal{G} \wedge a^v_{loc} \in AVId\}$, $temap(\mathbf{B}, I_{Exec})$ is an equivalence relation over $\{\langle E, nter\rangle \in LabS \mid E \in \mathcal{G} \wedge nter \subset Exec\}$.

Let us now extend some standard operators to the Ψ-indexed relation families.

Definition 13.16 *Given two Ψ-indexed relation families $\mathbf{B}' = \{\mathcal{B}'_\psi \mid \psi \in \Psi\}$ and $\mathbf{B}'' = \{\mathcal{B}''_\psi \mid \psi \in \Psi\}$, we define:*

- $\mathbf{B}' \subseteq \mathbf{B}'' \;\;\Leftrightarrow\;\; \mathcal{B}'_\psi \subseteq \mathcal{B}''_\psi \;\forall \psi \in \Psi$
- $\mathbf{B} = \mathbf{B}' \cup \mathbf{B}'' \;\;\Leftrightarrow\;\; \mathbf{B} = \{\mathcal{B}_\psi \mid \mathcal{B}_\psi = \mathcal{B}'_\psi \cup \mathcal{B}''_\psi\}$ ■

Definition 13.17 *Given a Ψ-indexed relation family $\mathbf{B} = \{\mathcal{B}_\psi \mid \psi \in \Psi\}$ we inductively define $\mathbf{B}^n = \{\mathcal{B}^{(n)}_\psi \mid \psi \in \Psi\}$ as follows:*

- $\mathcal{B}^{(1)}_\psi = \mathcal{B}_\psi \qquad \forall \psi \in \Psi$
- $\mathcal{B}^{(n)}_\psi = \bigcup\limits_{\substack{\psi', \psi'' \in \Psi \\ \psi = \psi' \circ \psi''}} \mathcal{B}^{(n-1)}_{\psi'} \circ \mathcal{B}_{\psi''} \qquad \forall \psi \in \Psi$

■

Definition 13.18 *Given a Ψ-indexed relation family $\mathbf{B} = \{\mathcal{B}_\psi \mid \psi \in \Psi\}$ we define $\mathbf{B}^+ = \{\mathcal{B}^{(+)}_\psi \mid \mathcal{B}^{(+)}_\psi = \bigcup\limits_{n \in \mathbb{N}_+} \mathcal{B}^{(n)}_\psi\}$.* ■

Lemma 13.1 *Let $\{\mathbf{B_i} \mid i \in I\}$ with $\mathbf{B_i} = \{\mathcal{B}_{i,\psi} \mid \psi \in \Psi\}$ be a set of strong GSMBFs. Then $\mathbf{B} = (\cup_{i \in I} \mathbf{B_i})^+$ is a strong GSMBF.*

Proof: Once easily observed that $\mathbf{B}$ satisfies the first three conditions of the definition of strong GSMBF because it is the transitive closure of the union of strong GSMBFs, assume that $(s_1, s_2) \in \mathcal{B}_\psi$. Since $\mathcal{B}_\psi = \cup_{n \in \mathbb{N}_+} (\cup_{i \in I} \mathcal{B}_i)^{(n)}_\psi$, we have $(s_1, s_2) \in (\cup_{i \in I} \mathcal{B}_i)^{(n)}_\psi$ for some $n \in \mathbb{N}_+$. The result follows by proving by induction on $n \in \mathbb{N}_+$ that, whenever $(s_1, s_2) \in (\cup_{i \in I} \mathcal{B}_i)^{(n)}_\psi$, then for all $(C_1, C_2) \in strel(\mathbf{B}, \psi)$ we have $tweight(s_1, C_1) = tweight(s_2, C_2)$ and for all $(C_1, C_2) \in terel(\mathbf{B}, \psi)$ we have $ter(s_1, C_1) = ter(s_2, C_2)$.

- If $n = 1$, then $(s_1, s_2) \in \mathcal{B}_{i,\psi}$ for some $i \in I$. Let $\mathit{strel}(\mathbf{B_i}, \psi) = \{(C'_{i,j}, C''_{i,j}) \mid j \in J_i\}$. Since $(s_1, s_2) \in \mathcal{B}_{i,\psi'}$ implies $(s_1, s_2) \in \mathcal{B}_{\psi'}$ for all $\psi' \in \Psi$, we have that for each $(C'_{i,j}, C''_{i,j}) \in \mathit{strel}(\mathbf{B_i}, \psi)$ there exists $(C_1, C_2) \in \mathit{strel}(\mathbf{B}, \psi)$ such that $C'_{i,j} \subseteq C_1$ and $C''_{i,j} \subseteq C_2$. Therefore each pair $(C_1, C_2) \in \mathit{strel}(\mathbf{B}, \psi)$ corresponds to a set of pairs $\{(C'_{i,j}, C''_{i,j}) \in \mathit{strel}(\mathbf{B_i}, \psi) \mid j \in J'_i\}$ where $J'_i \subseteq J_i$, such that $C_1 = \bigcup_{j \in J'_i} C'_{i,j}$ and $C_2 = \bigcup_{j \in J'_i} C''_{i,j}$. As a consequence for all $(C_1, C_2) \in \mathit{strel}(\mathbf{B}, \psi)$ we have that $\mathit{tweight}(s_1, C_1) = \sum \{\!| \mathit{tweight}(s_1, C'_{i,j}) \mid j \in J'_i |\!\} = \sum \{\!| \mathit{tweight}(s_2, C''_{i,j}) \mid j \in J'_i |\!\} = \mathit{tweight}(s_2, C_2)$; because $\mathbf{B_i}$ is a strong GSMBF. By following a similar argument for $\mathit{terel}(\mathbf{B_i}, \psi) = \{(C'_{i,k}, C''_{i,k}) \mid k \in K_i\}$ we have also that for all $(C_1, C_2) \in \mathit{terel}(\mathbf{B}, \psi)$: $\mathit{ter}(s_1, C_1) = \bigvee_{k \in K'_i} \mathit{ter}(s_1, C'_{i,k}) = \bigvee_{k \in K'_i} \mathit{ter}(s_2, C''_{i,k}) = \mathit{ter}(s_2, C_2)$.

- Let $n > 1$. From $(s_1, s_2) \in (\cup_{i \in I} \mathcal{B}_i)^{(n)}_{\psi}$ we derive that there exists $s \in \mathit{LabS}$ and ψ', ψ'' such that $(s_1, s) \in (\cup_{i \in I} \mathcal{B}_i)^{(n-1)}_{\psi'}$ and $(s, s_2) \in \mathcal{B}_{i,\psi''}$ for some $i \in I$. For each $(C_1, C_2) \in \mathit{strel}(\mathbf{B}, \psi)$ we have two cases depending on whether $\exists s'$ such that $(s'_1, s') \in \mathit{stmap}(\mathbf{B}, \psi')$ with $s'_1 \in C_1 \quad \Leftrightarrow \quad \exists s'$ such that $(s', s'_2) \in \mathit{stmap}(\mathbf{B}, \psi'')$ with $s'_2 \in C_2$ or not.

  - In the case the assertion holds, it follows that there exists $C \neq \emptyset$ such that $(C_1, C) \in \mathit{strel}(\mathbf{B}, \psi')$ and $(C, C_2) \in \mathit{strel}(\mathbf{B}, \psi'')$. It turns out: $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s, C)$, by the induction hypothesis; and $\mathit{tweight}(s, C) = \mathit{tweight}(s_2, C_2)$, by applying the same argument as the previous point (case of $n = 1$).
  - In the case the assertion does not hold, it follows that there exist C'_1, C'_2 with $C_1 \subseteq C'_1$ and $C_2 \subseteq C'_2$ such that $(C'_1, \emptyset) \in \mathit{strel}(\mathbf{B}, \psi')$ and $(\emptyset, C'_2) \in \mathit{strel}(\mathbf{B}, \psi'')$. It turns out: $\mathit{tweight}(s_1, C_1) = 0$, by the induction hypothesis; and $\mathit{tweight}(s_2, C_2) = 0$, by applying the same argument as the previous point (case of $n = 1$).

  For each $(C_1, C_2) \in \mathit{terel}(\mathbf{B}, \psi)$ we follow a similar argument and we obtain: $\mathit{ter}(s_1, C_1) = \mathit{ter}(s, C)$ and $\mathit{ter}(s, C) = \mathit{ter}(s_2, C_2)$ in the first case; and $\mathit{ter}(s_1, C_1) = \mathit{False}$ and $\mathit{ter}(s_2, C_2) = \mathit{False}$ in the second second case.

□

Proposition 13.2 *Let* $\sim_{\mathbf{GSMBF}} = \{\sim_{GSMBF,\psi} \mid \psi \in \Psi\}$ *be the union of all the strong GSMBFs. Then* $\sim_{\mathbf{GSMBF}}$ *is the largest strong GSMBF.*

Proof: By definition, $\sim_{\mathbf{GSMBF}}$ contains the largest strong GSMBF. If we prove that $\sim_{\mathbf{GSMBF}}$ is a strong GSMBF, then we are done. Since $\sim_{\mathbf{GSMBF}} \subseteq \sim_{\mathbf{GSMBF}}^+$ trivially holds, and $\sim_{\mathbf{GSMBF}}^+ \subseteq \sim_{\mathbf{GSMBF}}$ is due to the fact that $\sim_{\mathbf{GSMBF}}^+$ is a strong GSMBF by virtue of Lemma 13.1 and that $\sim_{\mathbf{GSMBF}}$ contains all the strong GSMBFs by definition, we have $\sim_{\mathbf{GSMBF}} = \sim_{\mathbf{GSMBF}}^+$. Again $\sim_{\mathbf{GSMBF}}^+$ is a strong GSMB because of Lemma 13.1, hence so is $\sim_{\mathbf{GSMBF}}$. □

Definition 13.19 *We define* $\sim_{\mathrm{GSMBE}} \subset \mathcal{G} \times \mathcal{G}$*, the strong generalized semi-Markovian bisimulation equivalence (strong GSMBE), as follows:*

$$\sim_{\text{GSMBE}} = \{(E_1, E_2) \in \mathcal{G} \times \mathcal{G} \mid (<E_1, \emptyset>, <E_2, \emptyset>) \in \sim_{GSMBF,\emptyset}\}$$

■

We recall that, as stated in Sect. 13.1.2, in order to get a congruence we need to represent in the semantic models also those computations which are not actually possible because of timing considerations. As an example consider

$$\begin{array}{lcl} E_1 & \equiv & (a.c.\underline{0} \,\|_{\emptyset}\, b.c.\underline{0}) \,\|_{\{c\}}\, c.\underline{0} \\ E_2 & \equiv & (a.c.\underline{0} \,\|_{\emptyset}\, b.\underline{0}) \,\|_{\{c\}}\, c.\underline{0} \end{array}$$

where any a has duration 4 and any b has duration 5. If the possibility of b terminating before a were not taken into account in the semantics, then it would turn out

$$\begin{array}{ccc} E_1 & \sim_{\text{GSMBE}} & E_2 \\ E_1 \,\|_{\{a\}}\, \underline{0} & \not\sim_{\text{GSMBE}} & E_2 \,\|_{\{a\}}\, \underline{0} \end{array}$$

Now we introduce the notion of strong GSMBF up to $\sim_{\mathbf{GSMBF}}$ which allows us to avoid redundancy in strong GSMBFs.

Definition 13.20 *A strong GSMBF up to* $\sim_{\mathbf{GSMBF}}$ *is a* Ψ*-indexed family* $\mathbf{B} = \{\mathcal{B}_\psi \subseteq \mathit{LabS} \times \mathit{LabS} \mid \psi \in \Psi\}$ *of relations over* LabS *such that:*

- $I_{\{\langle E, Exec\rangle \mid E \in \mathcal{G}\}} \subseteq \mathcal{B}_{I_{Exec}} \qquad \forall\, Exec \subseteq AVId$
- $s_1\ \mathcal{B}_\psi\ s_2 \Rightarrow s_2\ \mathcal{B}_{\psi^{-1}}\ s_1 \qquad \forall\, \psi \in \Psi$
- $s_1\ \mathcal{B}_{\psi'}\ s_2 \wedge s_2\ \mathcal{B}_{\psi''}\ s_3 \Rightarrow s_1\ \mathcal{B}_{\psi' \circ \psi''}\ s_3 \qquad \forall\, \psi', \psi'' \in \Psi$
- $(s_1, s_2) \in \mathcal{B}_\psi \Rightarrow$
  - $s_1 \equiv \langle E_1, \mathit{dom}(\psi)\rangle \wedge s_2 \equiv \langle E_2, \mathit{range}(\psi)\rangle$
  - $\forall\, (C_1, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi).\ \mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$
  - $\forall\, (C_1, C_2) \in \mathit{terel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi).\ \mathit{ter}(s_1, C_1) = \mathit{ter}(s_2, C_2)$ ■

Proposition 13.3 *If* $\mathbf{B} = \{\mathcal{B}_\psi \subseteq \mathit{LabS} \times \mathit{LabS} \mid \psi \in \Psi\}$ *is a strong GSMBF up to* $\sim_{\mathbf{GSMBF}}$*, then* $\mathbf{B} \subseteq\ \sim_{\mathbf{GSMBF}}$.

Proof: Given $\mathbf{B}$ strong GSMBF up to $\sim_{\mathbf{GSMBF}}$, we first prove that $(\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+$ is a strong GSMBF. By proceeding by induction on $n \in \mathbb{N}_+$ we show that, whenever $(s_1, s_2) \in (\mathcal{B} \cup \sim_{GSMBF})^{(n)}_\psi$, then for all $(C_1, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ we have $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$ and for all $(C_1, C_2) \in \mathit{terel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ we have $\mathit{ter}(s_1, C_1) = \mathit{ter}(s_2, C_2)$.

- If $n = 1$, then $(s_1, s_2) \in \mathcal{B}_\psi \cup \sim_{GSMBF,\psi}$. If $(s_1, s_2) \in \mathcal{B}_\psi$ the the result trivially follows from the fact that $\mathbf{B}$ is a strong GSMBF up to $\sim_{\mathbf{GSMBF}}$. If $(s_1, s_2) \in \sim_{GSMBF,\psi}$ we observe that, since $\sim_{\mathbf{GSMBF}} \subseteq (\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+$, each pair $(C_1, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ corresponds to a set of pairs $\{(C_i', C_i'') \in \mathit{strel}(\sim_{\mathbf{GSMBF}}, \psi) \mid i \in I\}$ for some I, such that $C_1 = \bigcup_{i \in I} C_i'$ and $C_2 = \bigcup_{i \in I} C_i''$. As a consequence for all $(C_1, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ we have that $\mathit{tweight}(s_1, C_1) = \sum\{\!| \mathit{tweight}(s_1, C_i') \mid i \in I |\!\} = \sum\{\!| \mathit{tweight}(s_2, C_i'') \mid i \in I |\!\} = \mathit{tweight}(s_2, C_2)$; because $s_1 \sim_{GSMBF,\psi} s_2$. By following a similar argument we have also that for each pair $(C_1, C_2) \in \mathit{terel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$: $\mathit{ter}(s_1, C_1) = \bigvee_{j \in J} \mathit{ter}(s_1, C_j') = \bigvee_{j \in J} \mathit{ter}(s_2, C_j'') = \mathit{ter}(s_2, C_2)$ for some J.

- Let $n > 1$. From $(s_1, s_2) \in (\mathcal{B} \cup \sim_{GSMBF})_\psi^{(n)}$ we derive that there exists $s \in \mathit{LabS}$ and ψ', ψ'' such that $(s_1, s) \in (\mathcal{B} \cup \sim_{GSMBF})_{\psi'}^{(n-1)}$ and $(s, s_2) \in \mathcal{B}_\psi \cup \sim_{GSMBF,\psi}$. For each $(C_1, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ we have two cases depending on whether $\exists s'$ such that $(s_1', s') \in \mathit{stmap}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi')$ with $s_1' \in C_1 \;\Leftrightarrow\; \exists s'$ such that $(s', s_2') \in \mathit{stmap}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi'')$ with $s_2' \in C_2$ or not.

  - In the case the assertion holds, it follows that there exists $C \neq \emptyset$ such that $(C_1, C) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi')$ and $(C, C_2) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi'')$. It turns out: $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s, C)$, by the induction hypothesis; and $\mathit{tweight}(s, C) = \mathit{tweight}(s_2, C_2)$, by applying the same argument as the previous point (case of $n = 1$).

  - In the case the assertion does not hold, it follows that there exist C_1', C_2' with $C_1 \subseteq C_1'$ and $C_2 \subseteq C_2'$ such that $(C_1', \emptyset) \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi')$ and $(\emptyset, C_2') \in \mathit{strel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi'')$. It turns out: $\mathit{tweight}(s_1, C_1) = 0$, by the induction hypothesis; and $\mathit{tweight}(s_2, C_2) = 0$, by applying the same argument as the previous point (case of $n = 1$).

  For each $(C_1, C_2) \in \mathit{terel}((\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+, \psi)$ we follow a similar argument and we obtain: $\mathit{ter}(s_1, C_1) = \mathit{ter}(s, C)$ and $\mathit{ter}(s, C) = \mathit{ter}(s_2, C_2)$ in the first case; and $\mathit{ter}(s_1, C_1) = \mathit{False}$ and $\mathit{ter}(s_2, C_2) = \mathit{False}$ in the second second case.

To complete the proof, we observe that $\mathbf{B} \subseteq (\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+$, and $(\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+ \subseteq \sim_{\mathbf{GSMBF}}$ because $(\mathbf{B} \cup \sim_{\mathbf{GSMBF}})^+$ is a strong GSMBF, hence $\mathbf{B} \subseteq \sim_{\mathbf{GSMBF}}$ by transitivity. □

13.5.3 Congruence Property

Now we show that $\sim_{\mathrm{GSMBE}}$ is a congruence for all the operators of GSMPA and for recursive definitions.

Theorem 13.1 *Let $E_1, E_2 \in \mathcal{G}$. If $E_1 \sim_{\mathrm{GSMBE}} E_2$ then:*

(i) For every $<a_n^v, w> \in \mathit{LAct}$, $<a_n^v, w>.E_1 \sim_{\mathrm{GSMBE}} <a_n^v, w>.E_2$.

(ii) For every $L \subseteq \mathit{AType}$, $E_1/L \sim_{\mathrm{GSMBE}} E_2/L$.

(*iii*) *For every* $\varphi \in ARFun$, $E_1[\varphi] \sim_{\mathrm{GSMBE}} E_2[\varphi]$.

(*iv*) *For every* $F \in \mathcal{G}$, $E_1 + F \sim_{\mathrm{GSMBE}} E_2 + F$ *and* $F + E_1 \sim_{\mathrm{GSMBE}} F + E_2$.

(*v*) *For every* $F \in \mathcal{G}$ *and* $S \subseteq AType$, $E_1 \,\|_S\, F \sim_{\mathrm{GSMBE}} E_2 \,\|_S\, F$ *and* $F \,\|_S\, E_1 \sim_{\mathrm{GSMBE}} F \,\|_S\, E_2$.

Proof: See Appendix D.1. □

Theorem 13.2 $\sim_{\mathrm{GSMBE}}$ *is a congruence w.r.t. recursive constant definitions.*
Proof: See Appendix D.2. □

Part V

Concluding Remarks

Chapter 14

Conclusion

In this chapter we present ongoing work and some ideas about future work.

14.1 Ongoing Work

We are currently working on the following topics:

- The full realization of the approach for correctly dealing with weakly guarded (Zeno) processes in interactive timed systems that we have described in the conclusion of Chapter 9. This can be done by modifying IWMCs, ITA and IGSMPs as there explained.

- The actual inclusion of the generative-reactive mechanism and of priority levels in the algebra GSMPA introduced in Chapter 13 (in such a chapter we have shown that this extension is straightforward), so to develop a new algebra GSMPA_{gr} precisely obtained as an extension EMPA_{gr}.

- The extension of the expressive power of EMPA_{gr} by introducing value passing according to the symbolic treatment of [19]. Since in [19] such an approach is already applied to EMPA, this extension should be straightforward.

- The development of a Markovian congruence for EMPA_{gr} (or for an algebra with an equivalent expressive power) that abstracts from probabilistic internal immediate actions, i.e. those actions which are unobservable and take no time, similarly as we abstract from τ actions in the calculus of IWMCs. Such a weak Markovian congruence could be used to compositionally minimize in an improved way (with respect to the equivalence defined in Chapter 11) the state space of an $\mathrm{EMPA}_{\mathrm{gr}}$ specification. Such a problem is rather involved, because, e.g., even if we consider the simplest case of a process algebra where we may have pure probabilistic choices among internal τ actions, it is not known how to define weak equivalence in such a way that it is a congruence for the operators of the algebra. We believe that a promising starting point for this research is the Markovian testing equivalence introduced in [25].

- The exploitation of the capability of the discrete time algebra of Chapter 10 to represent different time units for the execution of actions belonging to the same specification for approximating generally distributed delays with discrete-time phase type distributions. This because the time unit used in such approximations crucially depends on which general distribution is to be approximated.

14.2 Future Work

We now conclude the thesis by outlining some work left for future research.

- We plan to scale the integrated approach for the specification and analysis of stochastic real-time systems presented in Sect. 1.1 from the calculus of IGSMPs to the algebra GSMPA_{gr}, obtained by importing in GSMPA generative-reactive synchronizations and priority levels. In particular the Markovian mapping should turn a GSMPA_{gr} term into an EMPA_{gr} term by approximating general distributions with phase-type distributions, while the real-time mapping should turn a GSMPA_{gr} term into a net of timed automata (e.g. those of [132, 154]). This would allow us to exploit the expressive features of GSMPA_{gr} for modeling and analyzing real world systems.

- Based on the integrated approach presented in the previous item we plan to build a software tool that: (i) automatizes the generation of semantic models from GSMPA_{gr} system specifications, their minimization, and the derivation of the underlying performance model; and (ii) automatically transforms GSMPA_{gr} specifications into EMPA_{gr} or timed automata specifications, so that it provides an interface with existing tools: TwoTowers [26] for EMPA_{gr} and, e.g., Kronos [154] for timed automata. In this way performance evaluation via Markovian analysis and real-time model checking of systems specified with GSMPA_{gr} could be automatically carried out.

- Once we have gained a sufficient expressive power and we have a software tool like that described in the previous item, it will be possible, and important, to develop a number of case studies taken from real world systems.

- A very important extension to the expressiveness of the process algebras with general distributions presented in this thesis would be the inclusion of a mechanism for interrupting timed activities. This requires the introduction of a special event in semantic models representing "delay interruption" instead of "delay termination". Interruption mechanisms, which are commonly used in the real-time literature (see e.g. [132]), would greatly enhance the expressive power of the algebras since a timed preemption mechanism is needed to model many real systems.

- An important development of the work on probabilistic weak equivalences would be to understand how to develop an equivalence notion which equates different patterns of generally distributed delays as follows. In the definition of [122] the internal computations of processes are standard "τ" actions.

In an algebra with generally distributed delays we can see also a delay as an internal computation (a *timed* "τ"). Therefore the idea is to extend the notion of observational congruence in such a way that it can equate, e.g., a sequence of timed τ with a single timed τ provided that distribution of durations are in the correct relationship. For example a sequence (or a more complex pattern) of exponential timed τ could be equated by a phase-type distributed timed τ. It is worth noting that the possibility of extending the notion of observational congruence in this way strictly depends on the fact that we can express delays with any duration distribution (in languages expressing exponential distributions only there is not such a possibility). This is desirable because it may lead to a tremendous state space reduction of semantic models. For this purpose it seems to be very promising to exploit the third technique for expressing ST semantics we have developed in Chapter 3. It is worth noting that the development of such an equivalence strictly depends on the solution of the problem of defining a weak equivalence in the pure probabilistic case. See [120] for a solution of this problem in the context of semi-Markov processes, i.e. in the absence of a parallel composition operator.

Appendix A

Technical Machinery for Chapter 3

In Sect. A.1 we present the definition of strongly guarded processes for both languages BL and RL (see Chapter 3), while Sect. A.2 and A.3 are entirely dedicated to the proofs of the two central theorems of Chapter 3: the equivalence of ST bisimulation via dynamic names and via the stack technique with ST bisimulation via static names, respectively.

A.1 Definition of Strong Guardedness

In this section we will provide the precise definition of strongly guarded process for both the Basic Language and the Refinement Language.

We start with the definition of strong guardedness for Basic Language process. If we exclude static operators from the language the definition that we present coincides with the definition given in [122].

Definition A.1 *A process P of BL_C is strongly guarded if for every subterm $recX.Q$ of P the following holds: for each free occurrence of X in Q we have that there exists $a \in A$ and a subterm $a.Q'$ of Q such that the same occurrence of X lies inside Q' and $a.Q'$ occurs in P not in the scope of an hiding operator "$/L$" with $a \in L$.* ■

As far as the definition of strong guardedness for the Refinement Language is concerned, the presence of the ACP sequential composition operator ";" (instead of the CCS prefix operator) and of the action refinement operator make the definition of strongly guarded processes slightly more complex with respect to the case of the basic language, due to technical reasons.

Similarly as for the basic language, in order for a process P of RL_C to be strongly guarded we must have that every occurrence of a variable X in a subterm $recX.Q$ of P is guarded, i.e. there exists a subterm $R; Q'$ of Q such that X lies inside Q' and we are guaranteed that R executes a visible action before terminating. First of all note that, due the adoption of ACP sequential composition, we can have that a variable X in Q can be guarded by several visible actions (e.g. in $(a + b); X$ the variable X is guarded by the set of actions $\{a, b\}$), while in the case of prefix a strongly guarded variable X is always guarded by a single action. Secondly, we note that, differently from the case of the Basic Language, the execution of X in Q may be preceded by the occurrence of a free variable $Y \neq X$ as, e.g., in process "$recY.a; recX.Y; X$". Therefore in order to understand if X is strongly guarded in $recX.Q$, we have to evaluate the closure of subterm Q by replacing each free variable with the corresponding definition. Thirdly, due to the presence of the action refinement operator we have to account for the combined effect of the refinement and hiding operators which include term R preceding variable X in their scope, e.g. X is not strongly guarded in $((a; X)[a \rightsquigarrow b])/b$.

In the following we present three auxiliary definitions which are necessary for the definition of strong guardedness in the Refinement Language.

Definition A.2 *Given $P \in RL_C$ and a finite set $M \subset A$, we define the predicate $G_M(P)$, stating that the successful termination of P is strongly guarded by the set of visible actions M, by structural induction as follows:*

$G_M(X),\ G_M(\delta)$	$\forall\, M \subset_{fin} A$
$G_M(a)$	if $a \in M$
$G_M(Q'; Q''),\ G_M(Q' \parallel_S Q'')$	if $G_M(Q') \vee G_M(Q'')$
$G_M(Q' + Q'')$	if $G_M(Q') \wedge G_M(Q'')$
$G_M(Q/L)$	if $G_{M-L}(Q)$
$G_M(recX.Q)$	if $G_M(Q)$
$G_M(Q'[a \rightsquigarrow Q''])$	if $(G_M(Q') \wedge a \notin M) \vee (G_{M\cup\{a\}}(Q') \wedge G_M(Q''))$

■

Note that it is correct to consider the successful termination of a subterm X of P to be strongly guarded because P is a closed term, hence a variable X occurs in P always in the scope of its definition $recX.Q$. Moreover the successful termination of $recX.Q$ is not strongly guarded only if in $recX.Q$ the successful termination may be reached without encountering neither visible actions nor X occurrences.

Definition A.3 *Given $P \in RL_C$ and a subterm Q of P, we define the closure of Q to be the closed term obtained, starting from Q, by subsequently replacing any occurrence of any free variable X with the subterm $RecX.R$ binding such variable in P.* ■

Note that, since variable occurences which are bound by inner recursive definitions including Q in their scope are not free inside outer recursive definitions including Q in their scope, the sequence of substitutions considered in Definition A.3 is always finite.

Definition A.4 *Given $P \in RL$ and a variable $X \in Var$ such that P includes one free occurrence of X, we define $RefHi_X(P)$, the term obtained from P by considering only the action refinement and hiding operators* *including X in their scope, as follows:*

$$
\begin{array}{ll}
\mathit{RefHi}_X(X) = X & \\
\mathit{RefHi}_X(Y) = \bot & \text{if } Y \neq X \\
\mathit{RefHi}_X(\mathit{rec}X.P) = \mathit{RefHi}_X(\delta) = \mathit{RefHi}_X(\varepsilon) = \mathit{RefHi}_X(\mu) = \bot & \\
\mathit{RefHi}_X(\mathit{rec}Y.P) = \mathit{RefHi}_X(P) & \text{if } Y \neq X \\
\mathit{RefHi}_X(P+Q) = \mathit{RefHi}_X(P \,\|_S\, Q) = \mathit{RefHi}_X(P;Q) \ = & \\
\qquad\qquad\qquad\qquad \max(\mathit{RefHi}_X(P), \mathit{RefHi}_X(Q)) & \\
\mathit{RefHi}_X(P/L) = \mathit{RefHi}_X(P)/L & \\
\mathit{RefHi}_X(P[a \rightsquigarrow Q]) = \mathit{RefHi}_X(P)[a \rightsquigarrow Q] &
\end{array}
$$

where we assume $\bot \leq P \ \ \forall P \in RL$ and $\bot/L = \bot[a \rightsquigarrow P] = \bot$. ■

We are now in a position to present the definition of strongly guarded processes of RL_C.

Definition A.5 *A process P of RL_C is strongly guarded if for every subterm $\mathit{rec}X.Q$ of P the following holds: for each free occurrence of X in Q we have that there exists $M \subset_{\mathit{fin}} A$ and a subterm $R;Q'$ of Q such that the same occurrence of X lies inside Q' and, accounting for the refinement and hiding operators which include R in their scope, the successful termination of term $\mathit{Cl}(R)$, representing the closure of R as a subterm of P, is strongly guarded by the set of actions M. More precisely, defined R^c to be the context of R in P expressed by means of a fresh free variable Y, i.e. $P \equiv R^c\{R/Y\}$, we must have that the successful termination of term $\mathit{RefHi}_Y(R^c)\{\mathit{Cl}(R)/Y\}$ is strongly guarded by M, i.e. $G_M(\mathit{RefHi}_Y(R^c)\{\mathit{Cl}(R)/Y\})$ holds true.* ■

A.2 Equivalence of the Dynamic Name and Static Name Techniques

In this section we will provide the proof of Theorem 3.5, i.e. the equivalence of ST bisimulation expressed through dynamic names with the definition of ST bisimulation based on static names. In the following we will denote by $\longrightarrow_{sn}$ the state transitions of the static name operational semantics in order to distinguish them from those of the dynamic name operational semantics (simply denoted by $\longrightarrow$).

We denote by $BL_{dn,CR}$ the subset of $BL_{dn,C}$ of the state terms reachable from a process of BL_C, i.e. $BL_{dn,CR} = \{T \in BL_{dn,C} \mid \exists P \in BL_C, k \in \mathbb{N}, \{T_i\}_{i<k}, \{\theta_i\}_{i\leq k}. P \xrightarrow{\theta_1} T_1 \xrightarrow{\theta_2} \ldots \xrightarrow{\theta_{k-1}} T_{k-1} \xrightarrow{\theta_k} T\}$.

The proof of Theorem 3.5 is based on a mapping *dsmap* between the dynamic and static names assumed by the actions of a process according to the dynamic name and static name ST semantics, respectively. An association between the dynamic name $i \in \mathbb{N}^+$ and the static name $@ \in \mathit{GLoc}$ assumed by an action a according to the two semantics is represented inside *dsmap* by the triple $(a, i, @)$. In the following we use $a : (i, @)$ to stand for $(a, i, @) \in \mathit{dsmap}$.

We denote by dsa, a *dynamic-static association*, whose elements are associations $(i, @)$. dsa ranges over the set DSA of partial bijections from $\mathbb{N}^+$ to $GLoc$. A *dynamic-static mapping dsmap* is a relation from A to $\mathbb{N}^+ \times GLoc$ such that $\forall a \in A.\, dsmap_a \in DSA$, i.e. $dsmap$ is a set including an independent dynamic-static association for each different action type. We denote by $DSMAP$ the set of all dynamic-static mappings.

Definition A.6 *The function dsmap* : $BL_{dn,CR} \longrightarrow DSMAP$*, which evaluates a dynamic-static mapping from a state term of a dynamic ST semantic model, is defined as follows:*

- $dsmap(\underline{0}) = dsmap(\mu.P) = dsmap(\dot{\tau}.P) = dsmap(P + Q) = dsmap(recX.P) = \emptyset$
- $dsmap(a_1^-.P) = \{a : (1, \bullet)\}$
- $dsmap(T \,\|_{S,M}\, U) =$
  $\{a : (j, l@) \mid a \notin S \wedge a : (i, @) \in dsmap(T) \wedge a : (j, l_i) \in M\} \cup$
  $\{a : (j, r@) \mid a \notin S \wedge a : (i, @) \in dsmap(U) \wedge a : (j, r_i) \in M\} \cup$
  $\{a : (i, <@'|@''>) \mid a \in S \wedge a : (i, @') \in dsmap(T) \wedge a : (i, @'') \in dsmap(U)\}$
- $dsmap(T/L) = \{a : (i, @) \in dsmap(T) \mid a \notin L\}$ ■

The following proposition determine how the dynamic-static mapping $dsmap(T)$ changes during the evolution of a dynamic state T in the three cases of an action start, an action termination and a silent transition.

Proposition A.1 *Let* T *be a dynamic state of* $BL_{dn,CR}$*. We have:*

- *If* $T \xrightarrow{a_i^+} T'$ *then* $i = \min\{k \mid k \notin dom(dsmap(T)_a)\}$ *and there exists* $@ \in GLoc$ *such that* $@ \notin range(dsmap(T)_a)$ *and* $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$*. Moreover* $dsmap(T')_b = dsmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$.
- *If* $T \xrightarrow{a_i^-} T'$ *then there exists* $@ \in GLoc$ *such that* $(i, @) \in dsmap(T)_a$ *and* $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$*. Moreover* $dsmap(T')_b = dsmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$.
- *If* $T \xrightarrow{\dot{\tau}} T'$ *then* $dsmap(T') = dsmap(T)$.

Proof: By induction on the derivation of a transition $T \xrightarrow{\theta} T'$, by showing that whenever all the premises of a dynamic name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T, the conclusion satisfies it. □

Now we will show how to derive the state of the static name semantics corresponding to a given state of the dynamic name semantics.

Definition A.7 *The function* $CL : BL_{dn,CR} \longrightarrow BL_{sn}$*, where* CL *stands for "clean", is defined as follows. Given* $T \in BL_{dn,CR}$*,* $CL(T)$ *is obtained by syntactically replacing each ST action* a_1^- *occurring in* T *with* a^- *and by discarding the sets* M *in the operators* $\|_{S,M}$. ■

Definition A.8 *The function* $sn : BL_{dn,CR} \longrightarrow \Sigma_R$*, which evaluates the state* $sn(T)$ *of the static name ST semantics corresponding to a state* T *of the dynamic name ST semantics, is defined as follows:*

$$sn(T) = \langle CL(T), \bigcup_{a \in A} \{a : @ \mid @ \in range(dsmap(T)_a)\}\rangle$$

■

Another preliminary result which is needed before proving the equivalence of the dynamic name and static name approaches is to establish a correspondence between the dynamic name and static name transitions.

Proposition A.2 *Let* T *be a dynamic state of* $BL_{dn,CR}$*. We have:*

- $T \xrightarrow{a_i^+} T'$ *implies* $\exists @ \in GLoc.\ dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\} \wedge$ $sn(T) \xrightarrow{a_@^+}_{sn} sn(T')$
- $T \xrightarrow{a_i^-} T'$ *implies* $\exists @ \in GLoc.\ dsmap(T')_a = dsmap(T)_a - \{(i, @)\} \wedge$ $sn(T) \xrightarrow{a_@^-}_{sn} sn(T')$
- $T \xrightarrow{\dot{\tau}} T'$ *implies* $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$

Proof: By induction on the derivation of a dynamic name transition from T by showing that whenever all the premises of a dynamic name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T and static name transitions are defined by structural semantics, the conclusion satisfies it. □

Proposition A.3 *Let* T *be a dynamic state of* $BL_{dn,CR}$*. We have:*

- $sn(T) \xrightarrow{a_@^+}_{sn} s$ *implies* $\exists i \in \mathbb{N}^+, T' \in BL_{dn,CR}.\ T \xrightarrow{a_i^+} T' \wedge s = sn(T') \wedge$ $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$
- $sn(T) \xrightarrow{a_@^-}_{sn} s$ *implies* $\exists i \in \mathbb{N}^+, T' \in BL_{dn,CR}.\ T \xrightarrow{a_i^-} T' \wedge s = sn(T') \wedge$ $dsmap(T')_a = dsmap(T)_a - \{(@, i)\}$
- $sn(T) \xrightarrow{\dot{\tau}}_{sn} s$ *implies* $\exists T' \in BL_{dn,CR}.\ T \xrightarrow{\dot{\tau}} T' \wedge s = sn(T')$

Proof: By induction on the derivation of a static name transition from $sn(T)$ by showing that whenever all the premises of a static name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T and dynamic name transitions are defined by structural semantics, the conclusion satisfies it. □

Finally, we introduce function $H(T, U)$ which computes a (static name) association history H from two dynamic name states T and U which are put in correspondence by a dynamic name equivalence.

Definition A.9 *The function* $H : BL_{dn,CR} \times BL_{dn,CR} \longrightarrow \mathcal{H}$, *which evaluates an association history* H *between the global locations of the actions in execution of a pair of dynamic name state terms* (T, U) *such that* $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, *i.e. the set of dynamic names of the actions in execution in* T *and* U *is the same, is defined as follows:*

$$H(T, U) = \{a : (@', @'') \mid a \in A \wedge dsmap(T)_a^{-1}(@') = dsmap(U)_a^{-1}(@'')\}$$

■

Now we are in a position to present the two main lemmas needed for the proof of Theorem 3.5. The first lemma shows that from a dynamic name ST bisimulation we can derive a static name ST bisimulation, the second lemma shows the converse.

Definition A.10 *Let* $\mathcal{B}$ *be a weak bisimulation [122] over dynamic name state terms of* $BL_{dn,C}$. *The corresponding* $\mathcal{H}$*-indexed family of binary relations* $\mathbf{sn}(\mathcal{B}) = \{sn(\mathcal{B})_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ *among states of a static name ST semantic model is defined as follows:*

$$sn(\mathcal{B})_H = \{(sn(T), sn(U)) \mid T, U \in BL_{dn,CR} \wedge (\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)) \wedge T\,\mathcal{B}\,U \wedge H(T, U) = H\}$$

■

Lemma A.1 *Given a weak bisimulation* $\mathcal{B}$ *[122] over dynamic name state terms of* $BL_{dn,C}$, *we have that* $\mathbf{sn}(\mathcal{B})$ *is a weak ST bisimulation family.*

Proof: First of all let us observe that $\mathbf{sn}(\mathcal{B})$ is a a symmetric $\mathcal{H}$-indexed family of binary relations over Σ_R because $\mathcal{B}$ is a symmetric binary relation.

Supposed that $H \in \mathcal{H}$ and $(s, t) \in sn(\mathcal{B})_H$ we show that:

1. H is compatible with (s, t)
2. $s \xrightarrow{a^+_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ t \stackrel{a^+_{@'}}{\Longrightarrow} t' \wedge (s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$
3. $s \xrightarrow{a^-_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@, @') \in H \wedge t \stackrel{a^-_{@'}}{\Longrightarrow} t' \wedge (s', t') \in sn(\mathcal{B})_{H - \{a:(@,@')\}}$
4. $s \xrightarrow{\dot{\tau}} s'$ implies $\exists t' \in \Sigma_R.\ t \stackrel{\varepsilon}{\Longrightarrow} t' \wedge (s', t') \in sn(\mathcal{B})_H$

In the following we will separately prove each item. Note that, by definition of $\mathbf{sn}(\mathcal{B})$, we have that there exist $T, U \in BL_{dn,CR}$ such that $s = sn(T)$ and $t = sn(U)$.

1. Since, by definition of $\mathbf{sn}(\mathcal{B})$, we have that $H(T, U) = H$, then $dom(H_a) = range(dsmap(T)_a)$ and $range(H_a) = range(dsmap(U)_a)$, hence H is compatible with $(sn(T), sn(U))$.

2. If $sn(T) \xrightarrow{a^+_{@}}_{sn} s'$ then, from Prop. A.3 and A.1, we have that there exist $i \in \mathbb{N}^+$ and $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{a^+_i} T'$. Moreover, we have $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$, with $(i, @) \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\ dsmap(T')_b = dsmap(T)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \overset{a^+_i}{\Longrightarrow} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.2 and A.1, we derive that there exists $@' \in GLoc$ such that $sn(U) \overset{a^+_{@'}}{\Longrightarrow}_{sn} sn(U')$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a \cup \{(i, @')\}$, with $(i, @') \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\ dsmap(U')_b = dsmap(U)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\ dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) \cup \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$.

3. If $sn(T) \xrightarrow{a^-_{@}}_{sn} s'$ then, from Prop. A.3 and A.1, we have that there exist $i \in \mathbb{N}^+$ and $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $(i, @) \in dsmap(T)_a$ and $T \xrightarrow{a^-_i} T'$. Moreover, we have $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$ and $\forall b \in A, b \neq a.\ dsmap(T')_b = dsmap(T)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \overset{a^-_i}{\Longrightarrow} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.2 and A.1, we derive that there exists $@' \in GLoc$ such that $(j, @') \in dsmap(U)_a$ and $sn(U) \overset{a^-_{@'}}{\Longrightarrow}_{sn} sn(U')$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a - \{(i, @')\}$ and $\forall b \in A, b \neq a.\ dsmap(U')_b = dsmap(U)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\ dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) - \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H - \{a:(@,@')\}}$.

4. If $sn(T) \xrightarrow{\dot{\tau}}_{sn} s'$ then, from Prop. A.3 and A.1, we have that there exists $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{\dot{\tau}} T'$ and $dsmap(T') = dsmap(T)$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \overset{\varepsilon}{\Longrightarrow} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.2 and A.1, we derive that $sn(U) \overset{\varepsilon}{\Longrightarrow}_{sn} sn(U')$ and $dsmap(U') = dsmap(U)$.

Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, we have $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$. Moreover $H(T', U') = H(T, U)$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_H$.

□

Definition A.11 *Let* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ *be a weak ST bisimulation family. The corresponding binary relation* $dn(\mathbf{B}) \subseteq BL_{dn,CR} \times BL_{dn,CR}$ *among states of a dynamic name ST semantic model is defined as follows:*

$$dn(\mathbf{B}) = \{(T, U) \in BL_{dn,CR} \times BL_{dn,CR} \mid (\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)) \wedge sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)\}$$

■

Lemma A.2 *Given a weak ST bisimulation family* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$*, we have that* $dn(\mathbf{B})$ *is a weak bisimulation [122].*

Proof: First of all let us observe that $dn(\mathbf{B})$ is a symmetric binary relation because $\mathbf{B}$ is a symmetric $\mathcal{H}$-indexed family of binary relations.

Supposed that $(T, U) \in dn(\mathbf{B})$ we show that:

$$T \xrightarrow{\theta} T' \text{ implies } \exists U' \in BL_{dn,CR}.\, U \stackrel{\hat{\theta}}{\Longrightarrow} U' \wedge (T', U') \in dn(\mathbf{B})$$

where $\hat{\theta} = \theta$ if $\theta \in STA_{dn}$ and $\hat{\theta} = \varepsilon$ if $\theta = \dot{\tau}$ [122].

We have three cases depending on whether θ is an action start, an action termination or a silent transition:

- If $T \xrightarrow{a_i^+} T'$ then, from Prop. A.2 and A.1, we have that there exists $@ \in GLoc$ such that $sn(T) \xrightarrow{a_@^+}_{sn} sn(T')$. Moreover, we have $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$, with $(i, @) \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\, dsmap(T')_b = dsmap(T)_b$.

  Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $sn(U) \stackrel{a_{@'}^+}{\Longrightarrow}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U) \cup \{a:(@,@')\}}\, s$.

  Then, from repeated use of Prop. A.3 and A.1, we derive that there exist $j \in \mathbb{N}^+$ and $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $U \stackrel{a_j^+}{\Longrightarrow} U'$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a \cup \{(j, @')\}$, with $(j, @') \notin dsmap(U)_a$, and $\forall b \in A, b \neq a.\, dsmap(U')_b = dsmap(U)_b$.

  Since, from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from repeated use of Prop. A.1, we derive that $i = j$. Moreover, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) \cup \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

- If $T \xrightarrow{a_i^-} T'$ then, from Prop. A.2 and A.1, we have that there exists $@ \in GLoc$ such that $(i, @) \in dsmap(T)_a$ and $sn(T) \xrightarrow{a_@^-}_{sn} sn(T')$. Moreover, we have $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$ and $\forall b \in A, b \neq a.\, dsmap(T')_b = dsmap(T)_b$.

  Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ and $@' \in GLoc$, with $a : (@, @') \in H(T,U)$, such that $sn(U) \xRightarrow{a_{@'}^-}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)-\{a:(@,@')\}}\, s$.

  Then, from repeated use of Prop. A.3 and A.1, we derive that there exist $j \in \mathbb{N}^+$ and $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $(j, @') \in dsmap(U)_a$ and $U \xRightarrow{a_j^-} U'$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a - \{(j, @')\}$ and $\forall b \in A, b \neq a.\, dsmap(U')_b = dsmap(U)_b$.

  Since $a : (@, @') \in H(T,U)$, from $(i, @) \in dsmap(T)_a$ and $(j, @') \in dsmap(U)_a$ we derive $i = j$. Moreover, since from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T',U') = H(T,U) - \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

- If $T \xrightarrow{\dot{\tau}} T'$ then, from Prop. A.2 and A.1, we have that $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$ and $dsmap(T') = dsmap(T)$.

  Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ such that $sn(U) \xRightarrow{\varepsilon}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)}\, s$.

  Then, from repeated use of Prop. A.3 and A.1, we derive that there exists $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $U \xRightarrow{\varepsilon} U'$ and $dsmap(U') = dsmap(U)$.

  Since from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, we have $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$. Moreover $H(T',U') = H(T,U)$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

□

The proof of Theorem 3.5, i.e. given $P, Q \in BL_C$ we have $P \simeq_{dn} Q$ if and only if $P \simeq_{sn} Q$, is now straightforward.

Proof:[of Theorem 3.5]

Proof of "$\Longrightarrow$" Let us suppose that $\langle P, \emptyset \rangle \xrightarrow{a_@^+}_{sn} s$. We have to show that $\exists t \in \Sigma_R, @' \in GLoc.$ $\langle Q, \emptyset \rangle \xRightarrow{a_{@'}^+} t \wedge (s,t) \in \approx_{sn,\{a:(@,@')\}}$.

From Prop. A.3 and A.1, we have that there exist $i \in \mathbb{N}^+$ and $T \in BL_{dn,CR}$, with $s = sn(T)$, such that $P \xrightarrow{a_i^+} T$. Moreover, we have $dsmap(T)_a = \{(i, @)\}$ and $\forall b \in A, b \neq a.\, dsmap(T)_b = \emptyset$.

From the definition of observational congruence [122] we derive that there $\exists U \in BL_{dn,CR}$ such that $Q \overset{a_i^+}{\Longrightarrow} U$ and $(T,U) \in \approx_{dn}$, where $\approx_{dn}$ denotes weak bisimulation equivalence [122], hence there exists a weak bisimulation $\mathcal{B}$ such that $T\,\mathcal{B}\,U$.

Then, from repeated use of Prop. A.2 and A.1, we derive that there exists $@' \in GLoc$ such that $Q \overset{a_{@'}^+}{\Longrightarrow}_{sn} sn(U)$. Moreover, we derive $dsmap(U)_a = \{(i,@')\}$ and $\forall b \in A, b \neq a.\, dsmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$ and $H(T,U) = \{a : (@,@')\}$, $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family such that, defined $t = sn(U)$, $(s,t) \in sn(\mathcal{B})_{\{a:(@,@')\}}$, hence $(s,t) \in \approx_{sn,\{a:(@,@')\}}$.

In the case of a start move of $\langle Q, \emptyset\rangle$ or a silent move of $\langle P, \emptyset\rangle$ or $\langle Q, \emptyset\rangle$ the proof is similar.

Proof of "$\Longleftarrow$" Let us suppose that $P \xrightarrow{a_i^+} T$. We have to show that $\exists U \in BL_{dn,CR}.\, Q \overset{a_i^+}{\Longrightarrow} U \wedge (T,U) \in \approx_{dn}$, where $\approx_{dn}$ denotes weak bisimulation equivalence [122].

From Prop. A.2 and A.1, we have that there exists $@ \in GLoc$ such that $\langle P, \emptyset\rangle \xrightarrow{a_@^+}_{sn} sn(T)$. Moreover, we have $dsmap(T)_a = \{(i,@)\}$ and $\forall b \in A, b \neq a.\, dsmap(T)_b = \emptyset$.

From the definition of (static name) observational ST congruence we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $\langle Q, \emptyset\rangle \overset{a_{@'}^+}{\Longrightarrow}_{sn} s$ and $(sn(T), s) \in \approx_{sn,\{a:(@,@')\}}$, hence there exists a weak ST bisimulation family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ such that $(sn(T), s) \in \mathcal{B}_{\{a:(@,@')\}}$.

Then, from repeated use of Prop. A.3 and A.1, we derive that there exists $j \in \mathbb{N}^+$ and $U \in BL_{dn,CR}$, with $s = sn(U)$, such that $Q \overset{a_j^+}{\Longrightarrow} U$. Moreover, we derive $dsmap(U)_a = \{(j,@')\}$ and $\forall b \in A, b \neq a.\, dsmap(U)_b = \emptyset$.

Since $\forall a \in A.\, dom(dsmap(P)_a) = dom(dsmap(Q)_a) = \emptyset$, from repeated use of Prop. A.1, we derive that $i = j = 1$. Finally, since we have $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$ and $H(T,U) = \{a : (@,@')\}$, $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$ and $dn(\mathbf{B})$ is a weak bisimulation [122] such that $(T,U) \in dn(\mathbf{B})$, hence $(T,U) \in \approx_{dn}$.

In the case of a start move of Q or a silent move of P or Q the proof is similar.

□

A.3 Equivalence of the Stack and Static Name Techniques

In this section we will provide the proof of Theorem 3.10, i.e. the equivalence of ST bisimulation expressed via the stack technique with the definition of ST bisimulation based on static names. Similarly as in Appendix B, we will denote by $\longrightarrow_{sn}$ the state transitions of the static name operational semantics in order to distinguish them from those of the stack technique operational semantics (simply denoted by $\longrightarrow$). Since Theorem 3.10 compares the stack and the static name approaches over processes of BL_C, in this appendix we will just consider the stack technique applied to a simple language with prefixing instead of general sequential composition. Therefore we will consider a set of stack technique state terms BL_s generated by a syntax with prefixing like that of state terms of the dynamic name technique ($BL_{s,C}$ as usual denotes the set of processes of BL_s). Moreover we will consider the stack technique transition system to be generated by the set of operational rules in Table 3.7, where we perform the obvious translation of the rules for sequential composition into those for prefixing and we do not consider the elimination rules for the parallel operator (we just treat processes which, after performing an a_i^-, become terminated processes in the same way as we do for those which do not terminate).

We denote by $BL_{s,CR}$ the subset of $BL_{s,C}$ of the state terms reachable from a process of BL_C, i.e. $BL_{s,CR} = \{T \in BL_{s,C} \mid \exists P \in BL_C, k \in \mathbb{N}, \{T_i\}_{i<k}, \{\theta_i\}_{i\leq k}. P \xrightarrow{\theta_1} T_1 \xrightarrow{\theta_2} \ldots \xrightarrow{\theta_{k-1}} T_{k-1} \xrightarrow{\theta_k} T\}$.

The proof of Theorem 3.10 is based on a mapping *ssmap* between the stack positions and the static names assumed by the actions of a process according to the ST semantics via stack technique and static name ST semantics, respectively. An association between the stack position $i \in \mathbb{N}^+$ and the static name $@ \in GLoc$ assumed by an action a according to the two semantics is represented inside *ssmap* by the triple $(a, i, @)$. In the following we use $a : (i, @)$ to stand for $(a, i, @) \in ssmap$.

We denote by W a *stack-static association string*. W ranges over the set $SSAS$ of partial bijections from $\mathbb{N}^+$ to $GLoc$ such that $\exists k \geq 1.\, dom(W) = \{1 \ldots k\}$, i.e. non-empty strings over the alphabet of global locations $GLoc$ without multiple occurrences of the same global location. A stack-static association string W can be seen as a set whose elements are associations $(i, @)$. A *stack-static mapping ssmap* is a relation from A to $\mathbb{N}^+ \times GLoc$ such that $\forall a \in A.\, ssmap_a \in SSAS$, i.e. *ssmap* is a set including an independent stack-static association string for each different action type. We denote by $SSMAP$ the set of all stack-static mappings.

Definition A.12 *The function* $ssmap : BL_{s,CR} \longrightarrow SSMAP$*, which evaluates a stack-static mapping from a state term of stack technique ST semantic model, is defined as follows:*

- $ssmap(\underline{0}) = ssmap(\mu.P) = ssmap(\dot{\tau}.P) = ssmap(P + Q) = ssmap(recX.P) = \emptyset$
- $ssmap(a_1^-.P) = \{a : (1, \bullet)\}$
- $ssmap(T \,\|_{S,M}\, U) =$
  $\{a : (j, l@) \mid a \notin S \wedge a : (i, @) \in ssmap(T) \wedge j = \#_{i,l}(M_a)\} \cup$
  $\{a : (j, r@) \mid a \notin S \wedge a : (i, @) \in ssmap(U) \wedge j = \#_{i,r}(M_a)\} \cup$
  $\{a : (i, <@'|@''>) \mid a \in S \wedge a : (i, @') \in ssmap(T) \wedge a : (i, @'') \in ssmap(U)\}$
- $ssmap(T/L) = \{a : (i, @) \in ssmap(T) \mid a \notin L\}$ ■

The following proposition determine how the stack-static mapping $ssmap(T)$ changes during the evolution of a stack technique state T in the three cases of an action start, an action termination and a silent transition.

Proposition A.4 *Let T be a state of $BL_{s,CR}$. We have:*

- *If* $T \xrightarrow{a^+} T'$ *there exists* $@ \in GLoc$ *such that* $@ \notin range(ssmap(T)_a)$ *and* $ssmap(T')_a = @ssmap(T)_a$*. Moreover* $ssmap(T')_b = ssmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$*.*

- *If* $T \xrightarrow{a_i^-} T'$ *then there exists* $@ \in GLoc$ *such that* $(i, @) \in ssmap(T)_a$ *and* $ssmap(T')_a = ssmap(T)_a \ominus i$. *Moreover* $ssmap(T')_b = ssmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$.
- *If* $T \xrightarrow{\dot{\tau}} T'$ *then* $ssmap(T') = ssmap(T)$.

Proof: By induction on the derivation of a transition $T \xrightarrow{\theta} T'$, by showing that whenever all the premises of a stack technique operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T, the conclusion satisfies it. □

The state of the static name semantics corresponding to a given state of the semantics via the stack technique is derived, as in Appendix B, as follows.

Definition A.13 *The function* $CL : BL_{s,CR} \longrightarrow BL_{sn}$, *where* CL *stands for "clean", is defined as follows. Given* $T \in BL_{s,CR}$, $CL(T)$ *is obtained by syntactically replacing each ST action* a_1^- *occurring in* T *with* a^- *and by discarding the sets* M *in the operators* $\|_{S,M}$. ■

Definition A.14 *The function* $sn : BL_{s,CR} \longrightarrow \Sigma_R$, *which evaluates the state* $sn(T)$ *of the static name ST semantics corresponding to a state* T *of the ST semantics via the stack technique, is defined as follows:*

$$sn(T) = \langle CL(T), \bigcup_{a \in A} \{a : @ \mid @ \in range(ssmap(T)_a)\}\rangle$$

■

As in Appendix B, another preliminary result which is needed before proving the equivalence of the stack and static name approaches is to establish a correspondence between the stack and static name transitions.

Proposition A.5 *Let* T *be a state of* $BL_{s,CR}$. *We have:*

- $T \xrightarrow{a^+} T'$ *implies* $\exists @ \in GLoc.\ ssmap(T')_a = @\, ssmap(T)_a \wedge sn(T) \xrightarrow{a^+_@}_{sn} sn(T')$
- $T \xrightarrow{a_i^-} T'$ *implies* $\exists @ \in GLoc.\ ssmap(T')_a = ssmap(T)_a \ominus i \wedge sn(T) \xrightarrow{a^-_@}_{sn} sn(T')$
- $T \xrightarrow{\dot{\tau}} T'$ *implies* $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$

Proof: By induction on the derivation of a stack technique transition from T by showing that whenever all the premises of a stack technique operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T and static name transitions are defined by structural semantics, the conclusion satisfies it. □

Proposition A.6 *Let* T *be a state of* $BL_{s,CR}$. *We have:*

- $sn(T) \xrightarrow{a^+_@}_{sn} s$ *implies* $\exists T' \in BL_{s,CR}.\ T \xrightarrow{a^+} T' \wedge s = sn(T') \wedge$
  $ssmap(T')_a = @\, ssmap(T)_a$
- $sn(T) \xrightarrow{a^-_@}_{sn} s$ *implies* $\exists i \in \mathbb{N}^+, T' \in BL_{s,CR}.\ T \xrightarrow{a^-_i} T' \wedge s = sn(T') \wedge$
  $ssmap(T')_a = ssmap(T)_a \ominus i$
- $sn(T) \xrightarrow{\dot{\tau}}_{sn} s$ *implies* $\exists T' \in BL_{s,CR}.\ T \xrightarrow{\dot{\tau}} T' \wedge s = sn(T')$

Proof: By induction on the derivation of a static name transition from $sn(T)$ by showing that whenever all the premises of a static name operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T and stack technique transitions are defined by structural semantics, the conclusion satisfies it. □

Finally, we introduce function $H(T, U)$ which computes a (static name) association history H from two stack technique states T and U which are put in correspondence by a stack technique equivalence.

Definition A.15 *The function $H : BL_{s,CR} \times BL_{s,CR} \longrightarrow \mathcal{H}$, which evaluates an association history H between the global locations of the actions in execution of a pair of stack technique state terms (T, U) such that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, i.e. the number of stack positions of the actions in execution in T and U is the same, is defined as follows:*

$$H(T, U) = \{a : (@', @'') \mid a \in A \wedge ssmap(T)^{-1}_a(@') = ssmap(U)^{-1}_a(@'')\}$$

■

Now we are in a position to present the two main lemmas needed for the proof of Theorem 3.10. The first lemma shows that from a stack technique ST bisimulation we can derive a static name ST bisimulation, the second lemma shows the converse.

Definition A.16 *Let $\mathcal{B}$ be a weak bisimulation [122] over stack technique state terms of $BL_{s,C}$. The corresponding $\mathcal{H}$-indexed family of binary relations $\mathbf{sn}(\mathcal{B}) = \{sn(\mathcal{B})_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ among states of a static name ST semantic model is defined as follows:*

$$sn(\mathcal{B})_H = \{(sn(T), sn(U)) \mid T, U \in BL_{s,CR} \wedge (\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)) \wedge T\,\mathcal{B}\,U \wedge H(T, U) = H\}$$

■

Lemma A.3 *Given a weak bisimulation $\mathcal{B}$ [122] over stack technique state terms of $BL_{s,C}$, we have that $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family.*

Proof: First of all let us observe that $\mathbf{sn}(\mathcal{B})$ is a a symmetric $\mathcal{H}$-indexed family of binary relations over Σ_R because $\mathcal{B}$ is a symmetric binary relation.

Supposed that $H \in \mathcal{H}$ and $(s, t) \in sn(\mathcal{B})_H$ we show that:

1. H is compatible with (s, t)

2. $s \xrightarrow{a^+_{@}} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ t \xRightarrow{a^+_{@'}} t' \wedge (s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$

3. $s \xrightarrow{a^-_{@}} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@, @') \in H \wedge t \xRightarrow{a^-_{@'}} t' \wedge$
$$(s', t') \in sn(\mathcal{B})_{H - \{a:(@,@')\}}$$

4. $s \xrightarrow{\dot{\tau}} s'$ implies $\exists t' \in \Sigma_R.\ t \xRightarrow{\varepsilon} t' \wedge (s', t') \in sn(\mathcal{B})_H$

In the following we will separately prove each item. Note that, by definition of $\mathbf{sn}(\mathcal{B})$, we have that there exist $T, U \in BL_{s,CR}$ such that $s = sn(T)$ and $t = sn(U)$.

1. Since, by definition of $\mathbf{sn}(\mathcal{B})$, we have that $H(T, U) = H$, then $dom(H_a) = range(ssmap(T)_a)$ and $range(H_a) = range(ssmap(U)_a)$, hence H is compatible with $(sn(T), sn(U))$.

2. If $sn(T) \xrightarrow{a^+_{@}}_{sn} s'$ then, from Prop. A.6 and A.4, we have that there exists $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{a^+} T'$. Moreover, we have $ssmap(T')_a = @\, ssmap(T)_a$, with $@ \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\ ssmap(T')_b = ssmap(T)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{s,CR}$ such that $U \xRightarrow{a^+} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.5 and A.4, we derive that there exists $@' \in GLoc$ such that $sn(U) \xRightarrow{a^+_{@'}}_{sn} sn(U')$. Moreover, we derive $ssmap(U')_a = @\, ssmap(U)_a\}$, with $@ \notin range(ssmap(U)_a)$, and $\forall b \in A, b \neq a.\ ssmap(U')_b = ssmap(U)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\ dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T', U') = H(T, U) \cup \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$.

3. If $sn(T) \xrightarrow{a^-_{@}}_{sn} s'$ then, from Prop. A.6 and A.4, we have that there exist $i \in \mathbb{N}^+$ and $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $(i, @) \in ssmap(T)_a$ and $T \xrightarrow{a^-_i} T'$. Moreover, we have $ssmap(T')_a = ssmap(T)_a \ominus i$ and $\forall b \in A, b \neq a.\ ssmap(T')_b = ssmap(T)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{s,CR}$ such that $U \xRightarrow{a^-_i} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.5 and A.4, we derive that there exists $@' \in GLoc$ such that $(j, @') \in ssmap(U)_a$ and $sn(U) \xRightarrow{a^-_{@'}}_{sn} sn(U')$. Moreover, we derive $ssmap(U')_a = ssmap(U)_a \ominus i$ and $\forall b \in A, b \neq a.\ ssmap(U')_b = ssmap(U)_b$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and

$ssmap(U)$ that we derived, we have: $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T', U') = H(T, U) - \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H-\{a:(@,@')\}}$.

4. If $sn(T) \xrightarrow{\dot{\tau}}_{sn} s'$ then, from Prop. A.6 and A.4, we have that there exists $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{\dot{\tau}} T'$ and $ssmap(T') = ssmap(T)$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [122] we derive that there exists $U' \in BL_{s,CR}$ such that $U \stackrel{\varepsilon}{\Longrightarrow} U'$ and $T'\,\mathcal{B}\,U'$.

   Then, from repeated use of Prop. A.5 and A.4, we derive that $sn(U) \stackrel{\varepsilon}{\Longrightarrow}_{sn} sn(U')$ and $ssmap(U') = ssmap(U)$.

   Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, we have $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$. Moreover $H(T', U') = H(T, U)$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_H$.

$\square$

Definition A.17 *Let* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ *be a weak ST bisimulation family. The corresponding binary relation* $s(\mathbf{B}) \subseteq BL_{s,CR} \times BL_{s,CR}$ *among states of a stack technique ST semantic model is defined as follows:*

$$s(\mathbf{B}) = \{(T, U) \in BL_{s,CR} \times BL_{s,CR} \mid (\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)) \wedge sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)\}$$

$\blacksquare$

Lemma A.4 *Given a weak ST bisimulation family* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$*, we have that* $s(\mathbf{B})$ *is a weak bisimulation [122].*

Proof: First of all let us observe that $s(\mathbf{B})$ is a symmetric binary relation because $\mathbf{B}$ is a symmetric $\mathcal{H}$-indexed family of binary relations.

Supposed that $(T, U) \in s(\mathbf{B})$ we show that:

$$T \xrightarrow{\theta} T' \text{ implies } \exists U' \in BL_{s,CR}.\, U \stackrel{\hat{\theta}}{\Longrightarrow} U' \wedge (T', U') \in s(\mathbf{B})$$

where $\hat{\theta} = \theta$ if $\theta \in STA_s$ and $\hat{\theta} = \varepsilon$ if $\theta = \dot{\tau}$ [122].

We have three cases depending on whether θ is an action start, an action termination or a silent transition:

- If $T \xrightarrow{a^+} T'$ then, from Prop. A.5 and A.4, we have that there exists $@ \in GLoc$ such that $sn(T) \xrightarrow{a^+_@}_{sn} sn(T')$. Moreover, we have $ssmap(T')_a = @\, ssmap(T)_a$, with $@ \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\, ssmap(T')_b = ssmap(T)_b$.

  Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $sn(U) \stackrel{a^+_{@'}}{\Longrightarrow}_{sn} s$ and $sn(T')\,\mathcal{B}_{H(T,U)\cup\{a:(@,@')\}}\,s$.

Then, from repeated use of Prop. A.6 and A.4, we derive that there exists $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $U \stackrel{a^+}{\Longrightarrow} U'$. Moreover, we derive $ssmap(U')_a = @'\ ssmap(U)_a$, with $@' \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\ ssmap(U')_b = ssmap(U)_b$.

Since, from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\ dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T',U') = H(T,U) \cup \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

- If $T \xrightarrow{a_i^-} T'$ then, from Prop. A.5 and A.4, we have that there exists $@ \in GLoc$ such that $(i, @) \in ssmap(T)_a$ and $sn(T) \xrightarrow{a_@^-}_{sn} sn(T')$. Moreover, we have $ssmap(T')_a = ssmap(T)_a \ominus i$ and $\forall b \in A, b \neq a.\ ssmap(T')_b = ssmap(T)_b$.

  Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$, with $a : (@, @') \in H(T,U)$, such that $sn(U) \stackrel{a_{@'}^-}{\Longrightarrow}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)-\{a:(@,@')\}}\, s$.

  Then, from repeated use of Prop. A.6 and A.4, we derive that there exist $j \in \mathbb{N}^+$ and $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $(j, @') \in ssmap(U)_a$ and $U \stackrel{a_j^-}{\Longrightarrow} U'$. Moreover, we derive $ssmap(U')_a = ssmap(U)_a \ominus j$ and $\forall b \in A, b \neq a.\ ssmap(U')_b = ssmap(U)_b$.

  Since $a : (@, @') \in H(T,U)$, from $(i, @) \in ssmap(T)_a$ and $(j, @') \in ssmap(U)_a$ we derive $i = j$. Moreover, since from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\ dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T',U') = H(T,U) - \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

- If $T \xrightarrow{\dot{\tau}} T'$ then, from Prop. A.5 and A.4, we have that $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$ and $ssmap(T') = ssmap(T)$.

  Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ such that $sn(U) \stackrel{\varepsilon}{\Longrightarrow}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)}\, s$.

  Then, from repeated use of Prop. A.6 and A.4, we derive that there exists $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $U \stackrel{\varepsilon}{\Longrightarrow} U'$ and $ssmap(U') = ssmap(U)$.

  Since from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, we have $\forall a \in A.\ dom(ssmap(T')_a) = dom(ssmap(U')_a)$. Moreover $H(T',U') = H(T,U)$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

$\square$

The proof of Theorem 3.10, i.e. given $P, Q \in BL_C$ we have $P \simeq_s Q$ if and only if $P \simeq_{sn} Q$, is now straightforward.

Proof:[of Theorem 3.10]

Proof of "$\Longrightarrow$" Let us suppose that $\langle P, \emptyset\rangle \xrightarrow{a^+_{@}}_{sn} s$. We have to show that $\exists t \in \Sigma_R, @' \in GLoc$. $\langle Q, \emptyset\rangle \xRightarrow{a^+_{@'}} t \wedge (s,t) \in \approx_{sn,\{a:(@,@')\}}$.

From Prop. A.6 and A.4, we have that there exists $T \in BL_{s,CR}$, with $s = sn(T)$, such that $P \xrightarrow{a^+} T$. Moreover, we have $ssmap(T)_a = \{(1,@)\}$ and $\forall b \in A, b \neq a.\, ssmap(T)_b = \emptyset$.

From the definition of observational congruence [122] we derive that there $\exists U \in BL_{s,CR}$ such that $Q \xRightarrow{a^+} U$ and $(T,U) \in\approx_s$, where $\approx_s$ denotes weak bisimulation equivalence [122], hence there exists a weak bisimulation $\mathcal{B}$ such that $T\,\mathcal{B}\,U$.

Then, from repeated use of Prop. A.5 and A.4, we derive that there exists $@' \in GLoc$ such that $Q \xRightarrow{a^+_{@'}}_{sn} sn(U)$. Moreover, we derive $ssmap(U)_a = \{(1,@')\}$ and $\forall b \in A, b \neq a.\, ssmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$ and $H(T,U) = \{a : (@,@')\}$, $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family such that, defined $t = sn(U)$, $(s,t) \in sn(\mathcal{B})_{\{a:(@,@')\}}$, hence $(s,t) \in \approx_{sn,\{a:(@,@')\}}$.

In the case of a start move of $\langle Q, \emptyset\rangle$ or a silent move of $\langle P, \emptyset\rangle$ or $\langle Q, \emptyset\rangle$ the proof is similar.

Proof of "$\Longleftarrow$" Let us suppose that $P \xrightarrow{a^+} T$. We have to show that $\exists U \in BL_{s,CR}.\, Q \xRightarrow{a^+} U \wedge (T,U) \in \approx_s$, where $\approx_s$ denotes weak bisimulation equivalence [122].

From Prop. A.5 and A.4, we have that there exists $@ \in GLoc$ such that $\langle P, \emptyset\rangle \xrightarrow{a^+_{@}}_{sn} sn(T)$. Moreover, we have $ssmap(T)_a = \{(1,@)\}$ and $\forall b \in A, b \neq a.\, ssmap(T)_b = \emptyset$.

From the definition of (static name) observational ST congruence we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $\langle Q, \emptyset\rangle \xRightarrow{a^+_{@'}}_{sn} s$ and $(sn(T), s) \in\approx_{sn,\{a:(@,@')\}}$, hence there exists a weak ST bisimulation family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ such that $(sn(T), s) \in \mathcal{B}_{\{a:(@,@')\}}$.

Then, from repeated use of Prop. A.6 and A.4, we derive that there exists $U \in BL_{s,CR}$, with $s = sn(U)$, such that $Q \xRightarrow{a^+} U$. Moreover, we derive $ssmap(U)_a = \{(1,@')\}$ and $\forall b \in A, b \neq a.\, ssmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$ and $H(T,U) = \{a : (@,@')\}$, $sn(T)\,\mathcal{B}_{H(T,U)}\, sn(U)$ and $s(\mathbf{B})$ is a weak bisimulation [122] such that $(T,U) \in s(\mathbf{B})$, hence $(T,U) \in \approx_s$.

In the case of a start move of Q or a silent move of P or Q the proof is similar.

$\square$

Appendix B

Technical Machinery for Chapter 11

In this appendix we present the proofs of the three main results of Chapter 11: the consistency of the $\mathrm{EMPA_{gr}}$ synchronization discipline with the cooperation structure model, the congruence property of $\sim_{\mathrm{MB}}$, and the soundness and completeness of the axiomatization of $\sim_{\mathrm{MB}}$ for nonrecursive process terms.

B.1 Consistency with the Cooperation Structure Model

In this section we will provide the proof of Theorem 11.2, showing that the synchronization discipline of $\mathrm{EMPA_{gr}}$ is consistent with the cooperation structure model. Before presenting the proof we introduce some preliminary definitions and we prove some auxiliary lemmas.

Definition B.1 *We define for all $E \in \mathcal{G}$ and $a \in AType$*

$$\begin{aligned} AA(E, a) &= \{\!| <a, \tilde{\lambda}> \in Act \mid \exists E' \in \mathcal{G}.\, E \xrightarrow{a, \tilde{\lambda}} E' \wedge PL(\tilde{\lambda}) \geq 0 \,|\!\} \\ PA(E, a) &= \{\!| <a, \tilde{\lambda}> \in Act \mid \exists E' \in \mathcal{G}.\, E \xrightarrow{a, \tilde{\lambda}} E' \wedge PL(\tilde{\lambda}) < 0 \,|\!\} \end{aligned}$$

which are the multisets of active resp. passive actions of type a executable by a given $EMPA_{\mathrm{gr}}$ term E. We then define for all multisets PA of passive actions of the same type

$$\begin{aligned} PASelect(PA) &= \{\!| <a, \tilde{\lambda}> \in PA \mid \forall <a, \tilde{\mu}> \in PA.\, |PL(\tilde{\lambda})| \geq |PL(\tilde{\mu})| \,|\!\} \\ TPAW(PA) &= \sum \{\!| w \mid \exists l \in \mathbb{N}.\, <a, *_{l,w}> \in PA \,|\!\} \end{aligned}$$

which are the multiset of the passive actions of PA at the highest priority level and the total weight of the passive actions of PA, respectively. ■

Definition B.2 *We define functions*

$$\begin{aligned} AMap &: \mathcal{P}(AType \times (CStructs \cup (\{\perp\} \times CTrees) \cup (MAct \times \{\perp\}))) \times AType \longrightarrow \mathcal{M}(Act) \\ PMap &: \mathcal{P}(AType \times (CStructs \cup (\{\perp\} \times CTrees) \cup (MAct \times \{\perp\}))) \times AType \longrightarrow \mathcal{M}(Act) \end{aligned}$$

by letting

$$
\begin{array}{rcl}
AMap(tcstructs, a) & = & \{\mid <a, \tilde{\lambda}> \in Act \mid \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT} \rangle) \in tcstructs. \\
& & (\overline{CT} = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\
& & \exists \mathcal{S} \in Dom(\llbracket \overline{CT} \rrbracket_{\mathrm{awd}}). \, \tilde{\lambda} = \tilde{\mu} \cdot \frac{\llbracket \overline{CT} \rrbracket_{\mathrm{awd}}(\mathcal{S})}{TW(\llbracket \overline{CT} \rrbracket_{\mathrm{awd}})} \mid\} \\
PMap(tcstructs, a) & = & \{\mid <a, \tilde{\lambda}> \in Act \mid \exists (a, \langle \bot, CT \rangle) \in tcstructs. \\
& & \exists \mathcal{S} \in Dom(\llbracket CT \rrbracket_{\mathrm{awd}}). \, \tilde{\lambda} = *_{\llbracket CT \rrbracket_{\mathrm{pri}}, \llbracket CT \rrbracket_{\mathrm{awd}}(\mathcal{S})} \mid\}
\end{array}
$$

which are mappings from a set of typed (possibly incomplete) cooperation structures to the multiset of active resp. passive actions of type a executable according to such a set. ■

Lemma B.1 *For all $E \in \mathcal{G}$ and $a \in AType$ the following holds:*

1. *If there exists $CT \in CTrees$ such that $(a, \langle \bot, CT \rangle) \in \llbracket E \rrbracket_{\mathrm{csm}}$, then CT is unique.*
2. *For all $m \in MAct$ such that there exists $\overline{CT} \in CTrees \cup \{\bot\}$ such that $(a, \langle m, \overline{CT} \rangle) \in \llbracket E \rrbracket_{\mathrm{csm}}$, $\overline{CT}$ is unique.*

Proof: An easy induction on the syntactical structure of E, in which the definition of $\llbracket\,\rrbracket_{\mathrm{csm}}$ of Table 11.4 is considered and the use of locations is exploited.

□

Lemma B.2 *For all $CT_1, CT_2 \in CTrees$*

$$
\llbracket CT_1 \vee CT_2 \rrbracket_{\mathrm{awd}} = \begin{cases} \llbracket CT_1 \rrbracket_{\mathrm{awd}} \cup \llbracket CT_2 \rrbracket_{\mathrm{awd}} & \text{if } \llbracket CT_1 \rrbracket_{\mathrm{pri}} = \llbracket CT_2 \rrbracket_{\mathrm{pri}} \\ \llbracket CT_1 \rrbracket_{\mathrm{awd}} & \text{if } \llbracket CT_1 \rrbracket_{\mathrm{pri}} > \llbracket CT_2 \rrbracket_{\mathrm{pri}} \\ \llbracket CT_2 \rrbracket_{\mathrm{awd}} & \text{if } \llbracket CT_2 \rrbracket_{\mathrm{pri}} > \llbracket CT_1 \rrbracket_{\mathrm{pri}} \end{cases}
$$

Proof: It follows by applying Def. 11.11 to $\llbracket CT_1 \vee CT_2 \rrbracket_{\mathrm{awd}}$, $\llbracket CT_1 \rrbracket_{\mathrm{awd}}$, and $\llbracket CT_2 \rrbracket_{\mathrm{awd}}$ above and by applying the definition of $\llbracket CT_1 \vee CT_2 \rrbracket_{\mathrm{pwd}}$. In particular, in the first case above we exploit the fact that the processes associated with the leaves of $CT_1 \vee CT_2$ are all distinct to observe that

$$
\begin{array}{l}
(\llbracket CT_1 \rrbracket_{\mathrm{pwd}} \cup \llbracket CT_2 \rrbracket_{\mathrm{pwd}})(\{proc(\mathcal{S}_i) \mid 1 \le i \le n\}) = \\
\quad = \begin{cases} \llbracket CT_1 \rrbracket_{\mathrm{pwd}}(\{proc(\mathcal{S}_i) \mid 1 \le i \le n\}) & \text{if } \forall i \in \{1, \dots, n\}. \, \mathcal{S}_i \, subt \, CT_1 \\ \llbracket CT_2 \rrbracket_{\mathrm{pwd}}(\{proc(\mathcal{S}_i) \mid 1 \le i \le n\}) & \text{if } \forall i \in \{1, \dots, n\}. \, \mathcal{S}_i \, subt \, CT_2 \\ \text{undefined} & \text{otherwise} \end{cases}
\end{array}
$$

□

Lemma B.3 *For all $CT_1, CT_2 \in CTrees$*

$$
\begin{array}{rcl}
\llbracket CT_1 \wedge CT_2 \rrbracket_{\mathrm{awd}} & = & \{(sacts_1 \cup sacts_2, \frac{w_1}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})} \cdot \frac{w_2}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})} \cdot N_{CT_1, CT_2}) \mid \\
& & (sacts_1, w_1) \in \llbracket CT_1 \rrbracket_{\mathrm{awd}} \wedge (sacts_2, w_2) \in \llbracket CT_2 \rrbracket_{\mathrm{awd}}\}
\end{array}
$$

Proof: Let us preliminarily observe that for all $CT \in CTrees$ we have

$$
TW(\llbracket CT \rrbracket_{\mathrm{pwd}}) = TW(\llbracket CT \rrbracket_{\mathrm{awd}})
$$

and $(\{proc(\mathcal{S}_i) \mid 1 \leq i \leq n\}, w) \in [\![CT]\!]_{\mathrm{pwd}} \iff (\{<\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \mid 1 \leq i \leq n \wedge l_i = \max\{l' \mid <\alpha, *_{l',w'}> \in \mathcal{S}_i\}\}, w \cdot \prod_{1 \leq i \leq n} \frac{w_i}{\sum\{\mid w' \mid <\alpha, *_{l',w'}> \in \mathcal{S}_i \wedge l' = l_i \mid\}}) \in [\![CT]\!]_{\mathrm{awd}}$ whenever $\mathcal{S}_i, 1 \leq i \leq n$, are distinct subterms of CT. By applying Def. 11.11 and the definition of $[\![CT_1 \wedge CT_2]\!]_{\mathrm{pwd}}$, we get

$$
\begin{aligned}
&[\![CT_1 \wedge CT_2]\!]_{\mathrm{awd}} = \\
&\quad = \{(\{<\alpha_i, *_{l_i,w_i}> \mid 1 \leq i \leq n\}, w) \mid \exists \mathcal{S}_1, \ldots, \mathcal{S}_n \; subt \; CT_1 \wedge CT_2. \\
&\qquad (\forall i, j \in \{1, \ldots, n\}. \, i \neq j \Longrightarrow \mathcal{S}_i \neq \mathcal{S}_j) \wedge \\
&\qquad (\forall i \in \{1, \ldots, n\}. <\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \wedge l_i = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}_i\}) \wedge \\
&\qquad w = [\![CT_1 \wedge CT_2]\!]_{\mathrm{pwd}}(\{proc(\mathcal{S}_i) \mid 1 \leq i \leq n\}) \cdot \prod_{1 \leq i \leq n} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}\} \\
&\quad = \{(\{<\alpha_i, *_{l_i,w_i}> \mid 1 \leq i \leq n\}, w) \mid \exists \mathcal{S}_1, \ldots, \mathcal{S}_n \; subt \; CT_1 \wedge CT_2. \, \exists w'_1, w'_2 \in \mathbb{R}_+. \\
&\qquad (\forall i, j \in \{1, \ldots, n\}. \, i \neq j \Longrightarrow \mathcal{S}_i \neq \mathcal{S}_j) \wedge \\
&\qquad (\forall i \in \{1, \ldots, n\}. <\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \wedge l_i = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}_i\}) \wedge \\
&\qquad (\{proc(\mathcal{S}_i) \mid \mathcal{S}_i \; subt \; CT_1\}, w'_1) \in [\![CT_1]\!]_{\mathrm{pwd}} \wedge \\
&\qquad (\{proc(\mathcal{S}_i) \mid \mathcal{S}_i \; subt \; CT_2\}, w'_2) \in [\![CT_2]\!]_{\mathrm{pwd}} \wedge \\
&\qquad w = \frac{w'_1}{TW([\![CT_1]\!]_{\mathrm{pwd}})} \cdot \frac{w'_2}{TW([\![CT_2]\!]_{\mathrm{pwd}})} \cdot N_{CT_1,CT_2} \cdot \\
&\qquad\qquad \prod_{\mathcal{S}_i \; subt \; CT_1} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}} \cdot \prod_{\mathcal{S}_i \; subt \; CT_2} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}\} \\
&\quad = \{(\{<\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \mid 1 \leq i \leq n_1 \wedge \mathcal{S}_i \; subt \; CT_1 \wedge \\
&\qquad\qquad\qquad (\forall i, j \in \{1, \ldots, n_1\}. \, i \neq j \Longrightarrow \mathcal{S}_i \neq \mathcal{S}_j) \wedge \\
&\qquad\qquad\qquad l_i = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}_i\})\} \cup \\
&\qquad \{<\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \mid n_1 + 1 \leq i \leq n_2 \wedge \mathcal{S}_i \; subt \; CT_2 \wedge \\
&\qquad\qquad\qquad (\forall i, j \in \{n_1 + 1, \ldots, n_2\}. \, i \neq j \Longrightarrow \mathcal{S}_i \neq \mathcal{S}_j) \wedge \\
&\qquad\qquad\qquad l_i = \max\{l \mid <\alpha, *_{l,w}> \in \mathcal{S}_i\})\}, \\
&\qquad \frac{w'_1 \cdot \prod_{i=1}^{n_1} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}}{TW([\![CT_1]\!]_{\mathrm{pwd}})} \cdot \frac{w'_2 \cdot \prod_{i=n_1+1}^{n_2} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}}{TW([\![CT_2]\!]_{\mathrm{pwd}})} \cdot N_{CT_1,CT_2}) \mid \\
&\qquad (\{<\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \mid 1 \leq i \leq n_1\}, w'_1 \cdot \prod_{i=1}^{n_1} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}) \in [\![CT_1]\!]_{\mathrm{awd}} \wedge \\
&\qquad (\{<\alpha_i, *_{l_i,w_i}> \in \mathcal{S}_i \mid n_1 + 1 \leq i \leq n_2\}, w'_2 \cdot \prod_{i=n_1+1}^{n_2} \frac{w_i}{\sum\{\mid w \mid <\alpha, *_{l,w}> \in \mathcal{S}_i \wedge l = l_i \mid\}}) \in [\![CT_2]\!]_{\mathrm{awd}}\}
\end{aligned}
$$

$\square$

We are now in a position to present the proof of Theorem 11.2.

Proof of Theorem 11.2 The definitions above allow us to simply rephrase the theorem as follows. For all $E \in \mathcal{G}$ and $a \in AType$ the two following equations must hold:

$$
\begin{aligned}
AA(E, a) &= AMap([\![E]\!]_{\mathrm{csm}}, a) \\
PA(E, a) &= PMap([\![E]\!]_{\mathrm{csm}}, a)
\end{aligned}
$$

We prove the correctness of both equations by induction on the structure of terms $E \in \mathcal{G}$. We have two base steps.

- If $E \equiv \underline{0}$, then we have $AA(\underline{0}, a) = \emptyset$ and also $AMap([\![\underline{0}]\!]_{\mathrm{csm}}, a) = \emptyset$, since $[\![\underline{0}]\!]_{\mathrm{csm}} = \emptyset$. The same holds for the second equation.

- If $E \equiv <b, \tilde{\lambda}'>.E_1$, then we have two cases.

  - If $PL(\tilde{\lambda}') \geq 0$, then the first equation has the following structure.

    - If $a \equiv b$ then $AA(<b, \tilde{\lambda}'>.E_1, b) = \{\!| <b, \tilde{\lambda}'> |\!\}$ and also $AMap([\![]\!]_{\mathrm{csm}}, a) = \{\!| <b, \tilde{\lambda}'> |\!\}$, since $[\![<b, \tilde{\lambda}'>.E_1]\!]_{\mathrm{csm}} = \{(b, \langle <\bullet, \tilde{\lambda}'>, \bot \rangle)\}$.
    - If $a \not\equiv b$ then $AA(<b, \tilde{\lambda}'>.E_1, a) = \emptyset$ and also the righthand multiset of the first equation turns out to be empty.

    As far as the second equation is concerned, we have that $PA(<b, \tilde{\lambda}'>.E_1, a) = \emptyset$ and also the righthand multiset of such an equation turns out to be empty.

  - If $\tilde{\lambda}' = *_{l,w}$, then for the first equation we have that $AA(<b, *_{l,w}>.E_1, a) = \emptyset$ and, since $[\![<b, *_{l,w}>.E_1]\!]_{\mathrm{csm}} = \{(b, \langle \bot, \{<\bullet, *_{l,w}>\} \rangle)\}$, also the righthand multiset of such equation turns out to be empty.

    The second equation has the following structure.

    - If $a \equiv b$ then $PA(<b, *_{l,w}>.E_1, b) = \{\!| <b, *_{l,w}>.E_1 |\!\}$ and also the righthand multiset of the second equation turns out to be $\{\!| <b, *_{l,w}> |\!\}$.
    - If $a \not\equiv b$ then $PA(<b, \tilde{\lambda}'>.E_1, a) = \emptyset$ and also the righthand multiset of the second equation turns out to be empty.

The inductive steps are the following ones.

- If $E \equiv E_1/L$, then the first equation has the following structure.

  - If $a \notin L \cup \{\tau\}$ then we have $AA(E_1/L, a) = AA(E_1, a)$. From the definition of $[\![E_1/L]\!]_{\mathrm{csm}}$ it immediately derives that $AMap([\![E_1/L]\!]_{\mathrm{csm}}, a) = AMap([\![E_1]\!]_{\mathrm{csm}}, a)$, hence the first equation derives from the induction hypothesis.
  - If $a \in L$ then we have $AA(E_1/L, a) = \emptyset$ and also $AMap([\![E_1/L]\!]_{\mathrm{csm}}, a) = \emptyset$.
  - If $a \equiv \tau$ then we have:

    $$AA(E_1/L, \tau) = \{\!| <\tau, \tilde{\lambda}> \in Act \mid \exists b \in L \cup \{\tau\}. <b, \tilde{\lambda}> \in AA(E_1, b) |\!\}$$

    By applying the induction hypothesis we have that:

    $$\begin{aligned} AA(E_1/L, \tau) = \{\!| <\tau, \tilde{\lambda}> \in Act \mid \exists b \in L \cup \{\tau\}. & \\ & \exists (b, \langle <\sigma, \tilde{\mu}>, \overline{CT} \rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ & \quad (\overline{CT} = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ & \exists \mathcal{S} \in Dom([\![\overline{CT}]\!]_{\mathrm{awd}}). \\ & \quad \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![\overline{CT}]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![\overline{CT}]\!]_{\mathrm{awd}})} |\!\} \end{aligned}$$

On the other hand, from the definition of $[\![E_1/L]\!]_{\mathrm{csm}}$, we have that:

$$\begin{aligned} AMap([\![E_1/L]\!]_{\mathrm{csm}}, \tau) = \{\!|\, <\tau, \tilde{\lambda}> \in Act \mid \exists b \in L \cup \{\tau\}. \\ \exists (b, \langle <\sigma, \tilde{\mu}>, \overline{CT}\rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ (\overline{CT} = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ \exists \mathcal{S} \in Dom([\![\overline{CT}]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![\overline{CT}]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![\overline{CT}]\!]_{\mathrm{awd}})} \,|\!\} \end{aligned}$$

This multiset is the same multiset we derived for $AA(E_1/L, \tau)$.

The second equation has the following structure.

– If $a \notin L \cup \{\tau\}$ then we have $PA(E_1/L, a) = PA(E_1, a)$. From the definition of $[\![E_1/L]\!]_{\mathrm{csm}}$ it immediately derives that $PMap([\![E_1/L]\!]_{\mathrm{csm}}, a) = PMap([\![E_1]\!]_{\mathrm{csm}}, a)$, hence the first equation derives from the induction hypothesis.

– If $a \in L$ then we have $PA(E_1/L, a) = \emptyset$ and also $PMap([\![E_1/L]\!]_{\mathrm{csm}}, a) = \emptyset$.

– If $a \equiv \tau$ then we have, similarly to the case of active actions:

$$PA(E_1/L, \tau) = \{\!|\, <\tau, \tilde{\lambda}> \in Act \mid \exists b \in L \cup \{\tau\}. <b, \tilde{\lambda}> \in PA(E_1, b) \,|\!\}$$

By applying the induction hypothesis we have that:

$$\begin{aligned} PA(E_1/L, \tau) = \{\!|\, <\tau, \tilde{\lambda}> \in Act \mid \exists b \in L \cup \{\tau\}. \\ \exists (b, \langle \bot, CT\rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ \exists \mathcal{S} \in Dom([\![CT]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = *_{[\![CT]\!]_{\mathrm{pri}}, [\![CT]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

On the other hand, from the definition of $[\![E_1/L]\!]_{\mathrm{csm}}$, we have that:

$$\begin{aligned} PMap([\![E_1/L]\!]_{\mathrm{csm}}, a) = \{\!|\, <\tau, \tilde{\lambda}> \in Act \mid (\exists b \in L \cup \{\tau\} \wedge \exists (b, \langle \bot, CT\rangle) \in [\![E_1]\!]_{\mathrm{csm}}) \wedge \\ \exists \mathcal{S} \in \textstyle\bigcup_{\exists b \in L \cup \{\tau\}. (b, \langle \bot, CT\rangle) \in [\![E_1]\!]_{\mathrm{csm}}} Dom([\![CT]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = *_{[\![CT]\!]_{\mathrm{pri}}, [\![CT]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

It is trivial to see that this multiset is equal to the multiset we derived for $PA(E_1/L, \tau)$.

- If $E \equiv E_1[\varphi]$, then for the first equation we have that, for all $a \in AType$:

$$AA(E_1[\varphi], a) = \{\!|\, <a, \tilde{\lambda}> \in Act \mid \exists b \in AType. \varphi(b) = a \wedge <b, \tilde{\lambda}> \in AA(E_1, b) \,|\!\}$$

By applying the induction hypothesis we have that, for all $a \in AType$:

$$\begin{aligned} AA(E_1[\varphi], a) = \{\!|\, <a, \tilde{\lambda}> \in Act \mid \exists b \in AType. \varphi(b) = a \wedge \\ \exists (b, \langle <\sigma, \tilde{\mu}>, \overline{CT}\rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ (\overline{CT} = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ \exists \mathcal{S} \in Dom([\![\overline{CT}]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![\overline{CT}]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![\overline{CT}]\!]_{\mathrm{awd}})} \,|\!\} \end{aligned}$$

On the other hand, from the definition of $[\![E_1[\varphi]]\!]_{\mathrm{csm}}$, we have that, for all $a \in AType$:

$$\begin{aligned} AMap([\![E_1[\varphi]]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists b \in AType.\, \varphi(b) = a \wedge \\ & \exists (b, \langle <\sigma, \tilde{\mu}>, \overline{CT} \rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ & (\overline{CT} = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ & \exists \mathcal{S} \in Dom([\![\overline{CT}]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![\overline{CT}]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![\overline{CT}]\!]_{\mathrm{awd}})} \,|\!\} \end{aligned}$$

This multiset is the same multiset we derived for $AA(E_1[\varphi], a)$.

As far as the second equation is concerned we have that, similarly to the case of active actions, for all $a \in AType$:

$$PA(E_1[\varphi], a) = \{\!| <a, \tilde{\lambda}> \in Act \mid \exists b \in AType.\, \varphi(b) = a \wedge <b, \tilde{\lambda}> \in PA(E_1, b) \,|\!\}$$

By applying the induction hypothesis we have that, for all $a \in AType$:

$$\begin{aligned} PA(E_1[\varphi], a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists b \in AType.\, \varphi(b) = a \wedge \\ & \exists (b, \langle \bot, CT \rangle) \in [\![E]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in Dom([\![CT]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![CT]\!]_{\mathrm{pri}}, [\![CT]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

On the other hand, from the definition of $[\![E_1[\varphi]]\!]_{\mathrm{csm}}$, we have that, for all $a \in AType$:

$$\begin{aligned} PMap([\![E_1[\varphi]]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & (\exists b \in AType.\, \varphi(b) = a \wedge \exists (b, \langle \bot, CT \rangle) \in [\![E_1]\!]_{\mathrm{csm}}) \wedge \\ & \exists \mathcal{S} \in \textstyle\bigcup_{\exists b \in AType.\varphi(b)=a \wedge (b, \langle \bot, CT \rangle) \in [\![E_1]\!]_{\mathrm{csm}}} Dom([\![CT]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![CT]\!]_{\mathrm{pri}}, [\![CT]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

It is trivial to see that this multiset is equal to the multiset we derived for $PA(E_1[\varphi], a)$.

- If $E \equiv E_1 + E_2$, then for the first equation we have that, for all $a \in AType$:

$$AA(E_1 + E_2, a) = AA(E_1, a) \oplus AA(E_2, a)$$

By applying the induction hypothesis we have that, for all $a \in AType$:

$$AA(E_1 + E_2, a) = AMap(E_1, a) \oplus AMap(E_2, a)$$

On the other hand, from the definition of $[\![E_1 + E_2]\!]_{\mathrm{csm}}$, we have that, for all $a \in AType$:

$$\begin{aligned} AMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT} \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ & (Upd(\overline{CT}, +_\swarrow) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ & \exists \mathcal{S} \in Dom([\![Upd(\overline{CT}, +_\swarrow)]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![Upd(\overline{CT}, +_\swarrow)]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![Upd(\overline{CT}, +_\swarrow)]\!]_{\mathrm{awd}})} \,|\!\} \oplus \\ \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT} \rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ & (Upd(\overline{CT}, +_\searrow) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ & \exists \mathcal{S} \in Dom([\![Upd(\overline{CT}, +_\searrow)]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![Upd(\overline{CT}, +_\searrow)]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![Upd(\overline{CT}, +_\swarrow)]\!]_{\mathrm{awd}})} \,|\!\} \end{aligned}$$

Since function *Upd* modifies a tree $\overline{CT}$ only by relabeling the names of the activities in its leaves, the weights returned by $[\![Upd(\overline{CT},+_{\swarrow})]\!]_{\text{awd}}$ are the same as those returned by $[\![\overline{CT}]\!]_{\text{awd}}$. Therefore this multiset is equal to the multiset we derived for $AA(E_1 + E_2)$.

As far as the second equation is concerned we have that, for all $a \in AType$:

$$PA(E_1 + E_2, a) = PASelect(PA(E_1, a) \oplus PA(E_2, a))$$

By applying the induction hypothesis we have that, for all $a \in AType$:

$$PA(E_1 + E_2, a) = PASelect(PMap(E_1, a) \oplus PMap(E_2, a))$$

On the other hand, we have that this multiset is equal to that obtained from $PMap([\![E_1 + E_2]\!]_{\text{csm}}, a)$ by applying the definition of $[\![E_1 + E_2]\!]_{\text{csm}}$. We preliminarily recall that the previous Lemma B.1 guarantees that there exists at most one tree CT such that $(a, \langle \bot, CT \rangle) \in [\![E_i]\!]_{\text{csm}}$ for $i \in \{1, 2\}$. We have three cases:

- If $\nexists CT_2 \in CTrees.(a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\text{csm}}$ then we have that $PA(E_1 + E_2, a) = PMap(E_1, a)$, since if $PMap(E_1, a)$ is not empty, then all its actions have the priority level $[\![CT_1]\!]_{\text{pri}}$ of the unique tree CT_1 such that $(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\text{csm}}$.

  On the other hand, from the definition of $[\![E_1 + E_2]\!]_{\text{csm}}$, we have that:

$$\begin{aligned} PMap([\![E_1 + E_2]\!]_{\text{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid {} & \exists (a, \langle \bot, Upd(CT, +_{\swarrow}) \rangle) \in [\![E_1]\!]_{\text{csm}}. \\ & \exists \mathcal{S} \in Dom([\![Upd(CT, +_{\swarrow})]\!]_{\text{awd}}). \\ & \tilde{\lambda} = *_{[\![Upd(CT,+_{\swarrow})]\!]_{\text{pri}}, [\![Upd(CT,+_{\swarrow})]\!]_{\text{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

  Since function *Upd* only relabels the names of the activities in the leaves of a tree CT, we have that $[\![Upd(CT, +_{\swarrow})]\!]_{\text{pri}} = [\![CT]\!]_{\text{pri}}$, and that the weights returned by $[\![Upd(CT, +_{\swarrow})]\!]_{\text{awd}}$ are the same as those returned by $[\![CT]\!]_{\text{awd}}$. Therefore this multiset is equal to the multiset we derived for $PA(E_1 + E_2)$.

- If $\nexists CT_1 \in CTrees.(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\text{csm}}$ then the proof is completely similar to the previous case.

- If $\exists CT_1 \in CTrees.(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\text{csm}}$ and $\exists CT_2 \in CTrees.(a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\text{csm}}$ then we have two subcases for the structure of the second equation:

  * If $\nexists \mathcal{S}_1, \mathcal{S}_2.\ CT_1 = \mathcal{S}_1 \wedge CT_2 = \mathcal{S}_2$ then we have that:

$$\begin{aligned} PMap([\![E_1 + E_2]\!]_{\text{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid {} & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\text{csm}}. \\ & \exists (a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\text{csm}}. \\ \exists \mathcal{S} \in Dom([\![Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})]\!]_{\text{awd}}). & \\ \tilde{\lambda} = *_{[\![Upd(CT_1,+_{\swarrow}) \vee Upd(CT_2,+_{\searrow})]\!]_{\text{pri}}, [\![Upd(CT_1,+_{\swarrow}) \vee Upd(CT_2,+_{\searrow})]\!]_{\text{awd}}(\mathcal{S})} \,|\!\} & \end{aligned}$$

    Three subcases arise for the structure of the second equation depending on the priority levels

$[\![CT_1]\!]_{\mathrm{pri}}$ and $[\![CT_2]\!]_{\mathrm{pri}}$ of the two unique subtrees CT_1 and CT_2 satisfying $(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}$ and $(a, \langle \bot, CT_2 \rangle)$.

- If $[\![CT_1]\!]_{\mathrm{pri}} > [\![CT_2]\!]_{\mathrm{pri}}$ then $PA(E_1 + E_2, a) = PMap(E_1, a)$, since the actions of $PMap(E_1, a)$ have higher priority level than those of $PMap(E_2, a)$.

  As far as $PMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a)$ is concerned, from Lemma B.2 we derive that $[\![Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})]\!]_{\mathrm{pri}} = [\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{pri}}$ and that $Dom([\![Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})]\!]_{\mathrm{awd}}) = Dom([\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}})$. Therefore we obtain:

  $$\begin{aligned} PMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid\ & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in Dom([\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{pri}}, [\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}}(\mathcal{S})} |\!\} \end{aligned}$$

  Since relabeling does not change priorities and weights, this multiset is equal to the multiset we derived for $PA(E_1 + E_2, a)$.

- If $[\![CT_1]\!]_{\mathrm{pri}} < [\![CT_2]\!]_{\mathrm{pri}}$ then the proof is completely similar to the previous subcase.

- If $[\![CT_1]\!]_{\mathrm{pri}} = [\![CT_2]\!]_{\mathrm{pri}}$ then $PA(E_1 + E_2, a) = PMap(E_1, a) \oplus PMap(E_2, a)$, since all the actions of $PMap(E_1, a)$ have the same priority level as those of $PMap(E_2, a)$.

  As far as $PMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a)$ is concerned, from Lemma B.2 we derive that $[\![Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})]\!]_{\mathrm{pri}} = [\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{pri}} = [\![Upd(CT_2, +_{\searrow})]\!]_{\mathrm{pri}}$ and $Dom([\![Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})]\!]_{\mathrm{awd}}) = Dom([\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}}) \cup Dom([\![Upd(CT_2, +_{\searrow})]\!]_{\mathrm{awd}})$. Therefore we obtain:

  $$\begin{aligned} PMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid\ & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in Dom([\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{pri}}, [\![Upd(CT_1, +_{\swarrow})]\!]_{\mathrm{awd}}(\mathcal{S})} |\!\} \oplus \\ \{\!| <a, \tilde{\lambda}> \in Act \mid\ & \exists (a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in Dom([\![Upd(CT_2, +_{\searrow})]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![Upd(CT_2, +_{\searrow})]\!]_{\mathrm{pri}}, [\![Upd(CT_2, +_{\searrow})]\!]_{\mathrm{awd}}(\mathcal{S})} |\!\} \end{aligned}$$

  Since relabeling does not change priorities and weights, this multiset is equal to the multiset we derived for $PA(E_1 + E_2, a)$.

– If $\exists \mathcal{S}_1, \mathcal{S}_2.\, CT_1 = \mathcal{S}_1 \wedge CT_2 = \mathcal{S}_2$ then we have that:

$$\begin{aligned} PMap([\![E_1 + E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid\ & \exists (a, \langle \bot, \mathcal{S}_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \exists (a, \langle \bot, \mathcal{S}_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in Dom([\![Upd(\mathcal{S}_1, +_{\swarrow}) \cup Upd(\mathcal{S}_2, +_{\searrow})]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![Upd(\mathcal{S}_1, +_{\swarrow}) \cup Upd(\mathcal{S}_2, +_{\searrow})]\!]_{\mathrm{pri}}, [\![Upd(\mathcal{S}_1, +_{\swarrow}) \cup Upd(\mathcal{S}_2, +_{\searrow})]\!]_{\mathrm{awd}}(\mathcal{S})} |\!\} \end{aligned}$$

Three subcases arise for the structure of the second equation depending on the priority levels $[\![\mathcal{S}_1]\!]_{\mathrm{pri}}$ and $[\![\mathcal{S}_2]\!]_{\mathrm{pri}}$, which are completely similar to the three subcases of the previous case, with the only difference that $Upd(\mathcal{S}_1, +_{\swarrow}) \cup Upd(\mathcal{S}_2, +_{\searrow})$ must be considered instead of $Upd(CT_1, +_{\swarrow}) \vee Upd(CT_2, +_{\searrow})$.

- If $E \equiv E_1 \,\|_S\, E_2$, then the first equation has the following structure.

  – If $a \notin S$ then we have:

$$AA(E_1 \,\|_S\, E_2, a) = AA(E_1, a) \oplus AA(E_2, a)$$

By applying the induction hypothesis we have that:

$$AA(E_1 \,\|_S\, E_2, a) = AMap(E_1, a) \oplus AMap(E_2, a)$$

On the other hand, from the definition of $[\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}$, we have that:

$$\begin{aligned} AMap([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a) = \{\!|\, <a, \tilde{\lambda}> \in Act \mid \exists(a, \langle <\sigma, \tilde{\mu}>, \overline{CT}\rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ (Upd(\overline{CT}, \|_{\swarrow}) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ \exists \mathcal{S} \in Dom([\![Upd(\overline{CT}, \|_{\swarrow})]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = \tilde{\mu} \cdot \frac{[\![Upd(\overline{CT}, \|_{\swarrow})]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![Upd(\overline{CT}, \|_{\swarrow})]\!]_{\mathrm{awd}})} \,|\!\} \oplus \\ \{\!|\, <a, \tilde{\lambda}> \in Act \mid \exists(a, \langle <\sigma, \tilde{\mu}>, \overline{CT}\rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ (Upd(\overline{CT}, \|_{\searrow}) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\ \exists \mathcal{S} \in Dom([\![Upd(\overline{CT}, \|_{\searrow})]\!]_{\mathrm{awd}}). \\ \tilde{\lambda} = \tilde{\mu} \cdot \frac{[\![Upd(\overline{CT}, \|_{\searrow})]\!]_{\mathrm{awd}}(\mathcal{S})}{TW([\![Upd(\overline{CT}, \|_{\swarrow})]\!]_{\mathrm{awd}})} \,|\!\} \end{aligned}$$

Since function *Upd* modifies a tree $\overline{CT}$ only by relabeling the names of the activities in its leaves, the weights returned by $[\![Upd(\overline{CT}, \|_{\swarrow})]\!]_{\mathrm{awd}}$ are the same as those returned by $[\![\overline{CT}]\!]_{\mathrm{awd}}$. Therefore this multiset is equal to the multiset we derived for $AA(E_1 \,\|_S\, E_2)$.

  – If $a \in S$ then we have:

$$\begin{aligned} AA(E_1 \,\|_S\, E_2, a) = \{\!|\, <a, \tilde{\gamma}> \in Act \mid \exists <a, \tilde{\lambda}_1> \in AA(E_1, a). \\ \exists <a, *_{l_2, w_2}> \in PA(E_2, a). \\ \tilde{\gamma} = \tilde{\lambda}_1 \cdot w_2 / TPAW(PA(E_2, a)) \,|\!\} \oplus \\ \{\!|\, <a, \tilde{\gamma}> \in Act \mid \exists <a, *_{l_1, w_1}> \in PA(E_1, a). \\ \exists <a, \tilde{\lambda}_2> \in AA(E_2, a). \\ \tilde{\gamma} = \tilde{\lambda}_2 \cdot w_1 / TPAW(PA(E_1, a)) \,|\!\} \end{aligned}$$

By applying the induction hypothesis (considering both equations) we have that:

$$
\begin{aligned}
AA(E_1 \,\|_S\, E_2, a) = \{\!|\, & <a, \tilde{\gamma}> \in Act \mid \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT}_1 \rangle) \in \llbracket E_1 \rrbracket_{\mathrm{csm}}. \\
& \exists (a, \langle \bot, CT_2 \rangle) \in \llbracket E_2 \rrbracket_{\mathrm{csm}}. \\
& \exists \mathcal{S}_2 \in Dom(\llbracket CT_2 \rrbracket_{\mathrm{awd}}). \\
& \quad (\overline{CT}_1 = \bot \wedge \tilde{\gamma} = \tilde{\mu} \cdot \tfrac{\llbracket CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})}) \vee \\
& \exists \mathcal{S}_1 \in Dom(\llbracket \overline{CT}_1 \rrbracket_{\mathrm{awd}}). \\
& \quad \tilde{\gamma} = \tilde{\mu} \cdot \tfrac{\llbracket \overline{CT}_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket \overline{CT}_1 \rrbracket_{\mathrm{awd}})} \cdot \tfrac{\llbracket CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})} \,|\!\} \oplus \\
\{\!|\, & <a, \tilde{\gamma}> \in Act \mid \exists (a, \langle \bot, CT_1 \rangle) \in \llbracket E_1 \rrbracket_{\mathrm{csm}}. \\
& \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT}_2 \rangle) \in \llbracket E_2 \rrbracket_{\mathrm{csm}}. \\
& \exists \mathcal{S}_1 \in Dom(\llbracket CT_1 \rrbracket_{\mathrm{awd}}). \\
& \quad (\overline{CT}_2 = \bot \wedge \tilde{\gamma} = \tilde{\mu} \cdot \tfrac{\llbracket CT_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})}) \vee \\
& \exists \mathcal{S}_2 \in Dom(\llbracket \overline{CT}_2 \rrbracket_{\mathrm{awd}}). \\
& \quad \tilde{\gamma} = \tilde{\mu} \cdot \tfrac{\llbracket \overline{CT}_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket \overline{CT}_2 \rrbracket_{\mathrm{awd}})} \cdot \tfrac{\llbracket CT_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})} \,|\!\}
\end{aligned}
$$

On the other hand, from the definition of $\llbracket E_1 \,\|_S\, E_2 \rrbracket_{\mathrm{csm}}$, we have that:

$$
\begin{aligned}
AMap(\llbracket E_1 \,\|_S\, E_2 \rrbracket_{\mathrm{csm}}, a) = \{\!|\, & <a, \tilde{\lambda}> \in Act \mid \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT}_1 \rangle) \in \llbracket E_1 \rrbracket_{\mathrm{csm}}. \\
& \exists (a, \langle \bot, CT_2 \rangle) \in \llbracket E_2 \rrbracket_{\mathrm{csm}}. \\
& \quad (Upd(\overline{CT}_1, \|_{\swarrow}) \,\Delta\, Upd(CT_2, \|_{\searrow}) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\
& \exists \mathcal{S} \in Dom(\llbracket Upd(\overline{CT}_1, \|_{\swarrow}) \,\Delta\, Upd(CT_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}}). \\
& \quad \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{\llbracket Upd(\overline{CT}_1, \|_{\swarrow}) \,\Delta\, Upd(CT_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}}(\mathcal{S})}{TW(\llbracket Upd(\overline{CT}_1, \|_{\swarrow}) \,\Delta\, Upd(CT_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}})} \,|\!\} \oplus \\
\{\!|\, & <a, \tilde{\lambda}> \in Act \mid \exists (a, \langle \bot, CT_1 \rangle) \in \llbracket E_1 \rrbracket_{\mathrm{csm}}. \\
& \exists (a, \langle <\sigma, \tilde{\mu}>, \overline{CT}_2 \rangle) \in \llbracket E_2 \rrbracket_{\mathrm{csm}}. \\
& \quad (Upd(CT_1, \|_{\swarrow}) \,\Delta\, Upd(\overline{CT}_2, \|_{\searrow}) = \bot \wedge \tilde{\lambda} = \tilde{\mu}) \vee \\
& \exists \mathcal{S} \in Dom(\llbracket Upd(CT_1, \|_{\swarrow}) \,\Delta\, Upd(\overline{CT}_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}}). \\
& \quad \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{\llbracket Upd(CT_1, \|_{\swarrow}) \,\Delta\, Upd(\overline{CT}_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}}(\mathcal{S})}{TW(\llbracket Upd(CT_1, \|_{\swarrow}) \,\Delta\, Upd(\overline{CT}_2, \|_{\searrow}) \rrbracket_{\mathrm{awd}})} \,|\!\}
\end{aligned}
$$

We observe that it is not possible that $Upd(\overline{CT}_1, \|_{\swarrow}) \,\Delta\, Upd(CT_2, \|_{\searrow}) = \bot$ since $Upd(CT_2, \|_{\searrow}) \neq \bot$ and similarly $Upd(CT_1, \|_{\swarrow}) \,\Delta\, Upd(\overline{CT}_2, \|_{\searrow}) \neq \bot$. Moreover from Lemma B.3 we have that for $\mathcal{S}_1 \in Dom(\llbracket CT_1 \rrbracket_{\mathrm{awd}})$ and $\mathcal{S}_2 \in Dom(\llbracket CT_2 \rrbracket_{\mathrm{awd}})$ we have $\llbracket CT_1 \wedge CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1 \cup \mathcal{S}_2) = \frac{\llbracket CT_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})} \cdot \frac{\llbracket CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})} \cdot N_{CT_1, CT_2}$. Besides $TW(\llbracket CT_1 \wedge CT_2 \rrbracket_{\mathrm{awd}}) = N_{CT_1, CT_2}$. We further observe that:

$$
\frac{\frac{\llbracket CT_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})} \cdot \frac{\llbracket CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})} \cdot N_{CT_1, CT_2}}{N_{CT_1, CT_2}} = \frac{\llbracket CT_1 \rrbracket_{\mathrm{awd}}(\mathcal{S}_1)}{TW(\llbracket CT_1 \rrbracket_{\mathrm{awd}})} \cdot \frac{\llbracket CT_2 \rrbracket_{\mathrm{awd}}(\mathcal{S}_2)}{TW(\llbracket CT_2 \rrbracket_{\mathrm{awd}})}
$$

We now rewrite the multiset above by dividing the two cases $\overline{CT}_1 = \bot$ and $\overline{CT}_1 \neq \bot$ (and similarly in the second part of the multiset for $\overline{CT}_2$). We have:

$$
\begin{aligned}
AMap([\![E_1 \,\|_S\, E_2]\!]_{\rm csm}, a) = \{\!|\, <a,\tilde{\lambda}> \in Act \mid\ & \exists(a,\langle <\sigma,\tilde{\mu}>, \overline{CT}_1\rangle) \in [\![E_1]\!]_{\rm csm}.\\
& \exists(a,\langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\rm csm}.\\
& \exists \mathcal{S}_2 \in Dom([\![CT_2]\!]_{\rm awd}).\\
& \quad (\overline{CT}_1 = \perp \wedge \lambda = \tilde{\mu} \cdot \tfrac{[\![CT_2]\!]_{\rm awd}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\rm awd})}) \vee\\
& \exists \mathcal{S}_1 \in Dom([\![\overline{CT}_1]\!]_{\rm awd}).\\
& \quad \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![CT_1]\!]_{\rm awd}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\rm awd})} \cdot \tfrac{[\![CT_2]\!]_{\rm awd}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\rm awd})} \,|\!\} \oplus\\
\{\!|\, <a,\tilde{\lambda}> \in Act \mid\ & \exists(a,\langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\rm csm}.\\
& \exists(a,\langle <\sigma,\tilde{\mu}>, \overline{CT}_2\rangle) \in [\![E_2]\!]_{\rm csm}.\\
& \exists \mathcal{S}_1 \in Dom([\![CT_1]\!]_{\rm awd}).\\
& \quad (\overline{CT}_2 = \perp \wedge \lambda = \tilde{\mu} \cdot \tfrac{[\![CT_1]\!]_{\rm awd}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\rm awd})}) \vee\\
& \exists \mathcal{S}_2 \in Dom([\![\overline{CT}_2]\!]_{\rm awd}).\\
& \quad \tilde{\lambda} = \tilde{\mu} \cdot \tfrac{[\![CT_1]\!]_{\rm awd}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\rm awd})} \cdot \tfrac{[\![CT_2]\!]_{awd}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\rm awd})} \,|\!\}
\end{aligned}
$$

This multiset is equal to that we obtained for $AA(E_1 \,\|_S\, E_2, a)$.

As far as the second equation is concerned, it has the following structure:

– If $a \notin S$ then we have:

$$PA(E_1 \,\|_S\, E_2, a) = PASelect(PA(E_1, a) \oplus PA(E_2, a))$$

By applying the induction hypothesis we have that:

$$PA(E_1 \,\|_S\, E_2, a) = PASelect(PMap(E_1, a) \oplus PMap(E_2, a))$$

On the other hand, we have that this multiset is equal to that obtained from $PMap([\![E_1 \,\|_S\, E_2]\!]_{\rm csm}, a)$ by applying the definition of $[\![E_1 \,\|_S\, E_2]\!]_{\rm csm}$. We preliminarily recall that the previous Lemma B.1 guarantees that there exists at most one tree CT such that $(a, \langle \perp, CT\rangle) \in [\![E_i]\!]_{\rm csm}$ for $i \in \{1,2\}$. We have three cases:

∗ If $\nexists CT_2 \in CTrees.(a, \langle \perp, CT_2\rangle) \in [\![E_2]\!]_{\rm csm}$ then we have that $PA(E_1 \,\|_S\, E_2, a) = PMap(E_1, a)$, since if $PMap(E_1, a)$ is not empty, then all its actions have the priority level $[\![CT_1]\!]_{\rm pri}$ of the unique tree CT_1 such that $(a, \langle \perp, CT_1\rangle) \in [\![E_1]\!]_{\rm csm}$.

On the other hand, from the definition of $[\![E_1 \,\|_S\, E_2]\!]_{\rm csm}$, we have that:

$$
\begin{aligned}
PMap([\![E_1 \,\|_S\, E_2]\!]_{\rm csm}, a) = \{\!|\, <a,\tilde{\lambda}> \in Act \mid\ & \exists(a,\langle \perp, Upd(CT, \|_{\swarrow})\rangle) \in [\![E_1]\!]_{\rm csm}.\\
& \exists \mathcal{S} \in Dom([\![Upd(CT, \|_{\swarrow})]\!]_{\rm awd}).\\
& \quad \tilde{\lambda} = *_{[\![Upd(CT,\|_{\swarrow})]\!]_{\rm pri}, [\![Upd(CT,\|_{\swarrow})]\!]_{\rm awd}(\mathcal{S})} \,|\!\}
\end{aligned}
$$

Since function *Upd* only relabels the names of the activities in the leaves of a tree CT, we have that $[\![Upd(CT, \|_{\swarrow})]\!]_{\rm pri} = [\![CT]\!]_{\rm pri}$, and that the weights returned by $[\![Upd(CT, \|_{\swarrow})]\!]_{\rm awd}$ are the same as those returned by $[\![CT]\!]_{\rm awd}$. Therefore this multiset is equal to the multiset we derived for $PA(E_1 \,\|_S\, E_2)$.

* If $\not\exists CT_1 \in \mathit{CTrees}.(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}$ then the proof is completely similar to the previous case.
* If $\exists CT_1 \in \mathit{CTrees}.(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}$ and $\exists CT_2 \in \mathit{CTrees}.(a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}$ then we have that:

$$\begin{aligned} \mathit{PMap}([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in \mathit{Act} \mid {} & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ & \exists (a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ \exists \mathcal{S} \in \mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow}) \vee \mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{awd}}). & \\ \tilde{\lambda} = *_{[\![\mathit{Upd}(CT_1,\|_{\swarrow}) \vee \mathit{Upd}(CT_2,\|_{\searrow})]\!]_{\mathrm{pri}}, [\![\mathit{Upd}(CT_1,\|_{\swarrow}) \vee \mathit{Upd}(CT_2,\|_{\searrow})]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} & \end{aligned}$$

Three subcases arise for the structure of the second equation depending on the priority levels $[\![CT_1]\!]_{\mathrm{pri}}$ and $[\![CT_2]\!]_{\mathrm{pri}}$ of the two unique subtrees CT_1 and CT_2 satisfying $(a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}$ and $(a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}$.

  · If $[\![CT_1]\!]_{\mathrm{pri}} > [\![CT_2]\!]_{\mathrm{pri}}$ then $\mathit{PA}(E_1 \,\|_S\, E_2, a) = \mathit{PMap}(E_1, a)$, since the actions of $\mathit{PMap}(E_1, a)$ have higher priority level than those of $\mathit{PMap}(E_2, a)$.
  As far as $\mathit{PMap}([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a)$ is concerned, from Lemma B.2 we derive that $[\![\mathit{Upd}(CT_1, \|_{\swarrow}) \vee \mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{pri}} = [\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{pri}}$ and that $\mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow}) \vee \mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{awd}}) = \mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{awd}})$. Therefore we obtain:

$$\begin{aligned} \mathit{PMap}([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in \mathit{Act} \mid {} & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ & \exists \mathcal{S} \in \mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{awd}}). \\ & \tilde{\lambda} = *_{[\![\mathit{Upd}(CT_1,\|_{\swarrow})]\!]_{\mathrm{pri}}, [\![\mathit{Upd}(CT_1,\|_{\swarrow})]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \end{aligned}$$

  Since relabeling does not change priorities and weights, this multiset is equal to the multiset we derived for $\mathit{PA}(E_1 \,\|_S\, E_2, a)$.

  · If $[\![CT_1]\!]_{\mathrm{pri}} < [\![CT_2]\!]_{\mathrm{pri}}$ then the proof is completely similar to the previous subcase.

  · If $[\![CT_1]\!]_{\mathrm{pri}} = [\![CT_2]\!]_{\mathrm{pri}}$ then $\mathit{PA}(E_1 \,\|_S\, E_2, a) = \mathit{PMap}(E_1, a) \oplus \mathit{PMap}(E_2, a)$, since all the actions of $\mathit{PMap}(E_1, a)$ have the same priority level as those of $\mathit{PMap}(E_2, a)$.
  As far as $\mathit{PMap}([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a)$ is concerned, from Lemma B.2 we derive that $[\![\mathit{Upd}(CT_1, \|_{\swarrow}) \vee \mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{pri}} = [\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{pri}} = [\![\mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{pri}}$ and that $\mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow}) \vee \mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{awd}}) = \mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{awd}}) \cup \mathit{Dom}([\![\mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{awd}})$. Therefore we obtain:

$$\begin{aligned} \mathit{PMap}([\![E_1 \,\|_S\, E_2]\!]_{\mathrm{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in \mathit{Act} \mid {} & \exists (a, \langle \bot, CT_1 \rangle) \in [\![E_1]\!]_{\mathrm{csm}}. \\ \exists \mathcal{S} \in \mathit{Dom}([\![\mathit{Upd}(CT_1, \|_{\swarrow})]\!]_{\mathrm{awd}}). & \\ \tilde{\lambda} = *_{[\![\mathit{Upd}(CT_1,\|_{\swarrow})]\!]_{\mathrm{pri}}, [\![\mathit{Upd}(CT_1,\|_{\swarrow})]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} \oplus & \\ \{\!| <a, \tilde{\lambda}> \in \mathit{Act} \mid {} & \exists (a, \langle \bot, CT_2 \rangle) \in [\![E_2]\!]_{\mathrm{csm}}. \\ \exists \mathcal{S} \in \mathit{Dom}([\![\mathit{Upd}(CT_2, \|_{\searrow})]\!]_{\mathrm{awd}}). & \\ \tilde{\lambda} = *_{[\![\mathit{Upd}(CT_2,\|_{\searrow})]\!]_{\mathrm{pri}}, [\![\mathit{Upd}(CT_2,\|_{\searrow})]\!]_{\mathrm{awd}}(\mathcal{S})} \,|\!\} & \end{aligned}$$

  Since relabeling does not change priorities and weights, this multiset is equal to the multiset we derived for $\mathit{PA}(E_1 \,\|_S\, E_2, a)$.

– If $a \in S$ then we have:

$$\begin{aligned} PA(E_1 \parallel_S E_2, a) = \{\!| <a, \tilde{\gamma}> \in Act \mid & \exists <a, *_{l_1,w_1}> \in PA(E_1, a). \\ & \exists <a, *_{l_2,w_2}> \in PA(E_2, a). \\ \tilde{\gamma} = *_{\max\{l_1,l_2\}, \frac{w_1}{TPAW(PA(E_1,a))} \cdot \frac{w_2}{TPAW(PA(E_2,a))} \cdot N} \wedge & \\ N = \begin{cases} TPAW(PA(E_1,a)) + TPAW(PA(E_2,a)) & \text{if } l_1 = l_2 \\ TPAW(PA(E_1,a)) & \text{if } l_1 > l_2 \\ TPAW(PA(E_2,a)) & \text{if } l_2 > l_1 \end{cases} & |\!\} \end{aligned}$$

By applying the induction hypothesis we have that:

$$\begin{aligned} PA(E_1 \parallel_S E_2, a) = \{\!| <a, \tilde{\gamma}> \in Act \mid & \exists (a, \langle \perp, CT_1 \rangle) \in [\![E]\!]_{\text{csm}} \\ & \exists (a, \langle \perp, CT_2 \rangle) \in [\![E]\!]_{\text{csm}}. \\ & \exists \mathcal{S}_1 \in Dom([\![CT_1]\!]_{\text{awd}}). \\ & \exists \mathcal{S}_2 \in Dom([\![CT_2]\!]_{\text{awd}}). \\ \tilde{\gamma} = *_{\max\{[\![CT_1]\!]_{\text{pri}}, [\![CT_2]\!]_{\text{pri}}\}, \frac{[\![CT_1]\!]_{\text{awd}}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\text{awd}})} \cdot \frac{[\![CT_2]\!]_{\text{awd}}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\text{awd}})} \cdot N_{CT_1,CT_2}} & \ |\!\} \end{aligned}$$

On the other hand, from the definition of $[\![E_1 \parallel_S E_2]\!]_{\text{csm}}$, we have that:

$$\begin{aligned} PMap([\![E_1 \parallel_S E_2]\!]_{\text{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists (a, \langle \perp, CT_1 \rangle) \in [\![E_1]\!]_{\text{csm}}. \\ & \exists (a, \langle \perp, CT_2 \rangle) \in [\![E_2]\!]_{\text{csm}}. \\ & \exists \mathcal{S} \in Dom([\![CT_1 \wedge CT_2]\!]_{\text{awd}}). \\ & \tilde{\lambda} = *_{[\![CT_1 \wedge CT_2]\!]_{\text{pri}}, [\![CT_1 \wedge CT_2]\!]_{\text{awd}}(\mathcal{S})} \ |\!\} \end{aligned}$$

From Lemma B.3 we have that $[\![CT_1 \wedge CT_2]\!]_{\text{pri}} = \max [\![CT_1]\!]_{\text{pri}}, [\![CT_2]\!]_{\text{pri}}$. Besides we have that for $\mathcal{S}_1 \in Dom([\![CT_1]\!]_{\text{awd}})$ and $\mathcal{S}_2 \in Dom([\![CT_2]\!]_{\text{awd}})$ we have $[\![CT_1 \wedge CT_2]\!]_{\text{awd}}(\mathcal{S}_1 \cup \mathcal{S}_2) = \frac{[\![CT_1]\!]_{\text{awd}}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\text{awd}})} \cdot \frac{[\![CT_2]\!]_{\text{awd}}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\text{awd}})} \cdot N_{CT_1,CT_2}$. Therefore we have that:

$$\begin{aligned} PMap([\![E_1 \parallel_S E_2]\!]_{\text{csm}}, a) = \{\!| <a, \tilde{\lambda}> \in Act \mid & \exists (a, \langle \perp, CT_1 \rangle) \in [\![E]\!]_{\text{csm}}. \\ & \exists (a, \langle \perp, CT_2 \rangle) \in [\![E]\!]_{\text{csm}}. \\ & \exists \mathcal{S}_1 \in Dom([\![CT_1]\!]_{\text{awd}}). \\ & \exists \mathcal{S}_2 \in Dom([\![CT_2]\!]_{\text{awd}}). \\ \tilde{\lambda} = *_{\max\{[\![CT_1]\!]_{\text{pri}}, [\![CT_2]\!]_{\text{pri}}\}, \frac{[\![CT_1]\!]_{\text{awd}}(\mathcal{S}_1)}{TW([\![CT_1]\!]_{\text{awd}})} \cdot \frac{[\![CT_2]\!]_{\text{awd}}(\mathcal{S}_2)}{TW([\![CT_2]\!]_{\text{awd}})} \cdot N_{CT_1,CT_2}} & \ |\!\} \end{aligned}$$

This multiset is equal to the multiset we derived for $PA(E_1 \parallel_S E_2, a)$. ∎

B.2 Congruence Property of $\sim_{\text{MB}}$

In this appendix we provide the proof of the two results (Theorems 11.3 and 11.4) concerned with the congruence property of $\sim_{\text{MB}}$. First, we demonstrate Theorem 11.3 that says that $\sim_{\text{MB}}$ is a congruence w.r.t. all the operators of EMPA_{gr}.

Proof of Theorem 11.3 Let $E_1, E_2 \in \mathcal{G}$.

(i) Let $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ be a Markovian bisimulation such that $(E_1, E_2) \in \mathcal{B}$. Given $<a, \tilde{\lambda}> \in Act$, we prove that

$$\mathcal{B}' = (\mathcal{B} \cup \{(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2), (<a, \tilde{\lambda}>.E_2, <a, \tilde{\lambda}>.E_1)\})^+$$

is a Markovian bisimulation. Observed that $\mathcal{B}'$ is an equivalence relation, we have two cases.

- If $(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2) \in \mathcal{B}$, then $\mathcal{B}' = \mathcal{B}$ and the result trivially follows.
- Assume that $(<a, \tilde{\lambda}>.E_1, <a, \tilde{\lambda}>.E_2) \notin \mathcal{B}$. Observed that

  $$\mathcal{G}/\mathcal{B}' = (\mathcal{G}/\mathcal{B} - \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}) \cup \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}} \cup [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}$$

  let $(F_1, F_2) \in \mathcal{B}'$, $b \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}'$.

  * If $(F_1, F_2) \in \mathcal{B}$ and $C \in \mathcal{G}/\mathcal{B} - \{[<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}\}$, then trivially $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}$ and $C = [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}} \cup [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

    $$Rate(F_j, b, l, C) = Rate(F_j, b, l, [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}) + Rate(F_j, b, l, [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}})$$

    so $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}' - \mathcal{B}$, i.e. $F_1 \in [<a, \tilde{\lambda}>.E_1]_{\mathcal{B}}$ and $F_2 \in [<a, \tilde{\lambda}>.E_2]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

    $$Rate(F_j, b, l, C) = \begin{cases} \tilde{\lambda} & \text{if } b = a \wedge l = PL(\tilde{\lambda}) \wedge C = [E_j]_{\mathcal{B}'} \\ \perp & \text{otherwise} \end{cases}$$

    Since $[E_1]_{\mathcal{B}'} = [E_2]_{\mathcal{B}'}$, it turns out that $Rate(F_1, b, l, C) = Rate(F_2, b, l, C)$.

(ii) Given $L \subseteq AType - \{\tau\}$, we prove that

$$\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}} \text{ where } \mathcal{B} = \{(E_1/L, E_2/L) \mid E_1 \sim_{\mathrm{MB}} E_2\}$$

is a Markovian bisimulation, with $Id_{\mathcal{G}}$ being the identity relation over $\mathcal{G}$. Observed that $\mathcal{B}'$ is an equivalence relation and that either each of the terms of an equivalence class has "$\_/L$" as outermost operator or none of them has, let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}'$.

- If $(F_1, F_2) \in Id_{\mathcal{G}}$, then trivially $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.
- If $(F_1, F_2) \in \mathcal{B}$, then $F_1 \equiv E_1/L$ and $F_2 \equiv E_2/L$ where $E_1 \sim_{\mathrm{MB}} E_2$.
  * If none of the terms in C has "$\_/L$" as outermost operator, then trivially $Rate(F_1, a, l, C) = \perp = Rate(F_2, a, l, C)$.
  * If each of the terms in C has "$\_/L$" as outermost operator, given $E/L \in C$ it turns out that $C = \{E'/L \mid E' \in [E]_{\sim_{\mathrm{MB}}}\}$. Thus for $j \in \{1, 2\}$ we have

    $$Rate(F_j, a, l, C) = \begin{cases} Rate(E_j, a, l, [E]_{\sim_{\mathrm{MB}}}) \\ Rate(E_j, \tau, l, [E]_{\sim_{\mathrm{MB}}}) + \sum_{b \in L} Rate(E_j, b, l, [E]_{\sim_{\mathrm{MB}}}) \end{cases}$$

    depending on whether $a \notin L \cup \{\tau\}$ or $a = \tau$. From $E_1 \sim_{\mathrm{MB}} E_2$ it follows $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

(*iii*) Given $\varphi \in ATRFun$, the proof that

$$\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}} \text{ where } \mathcal{B} = \{(E_1[\varphi], E_2[\varphi]) \mid E_1 \sim_{\mathrm{MB}} E_2\}$$

is a Markovian bisimulation is similar to the one developed in (*ii*). The main difference is that in the last subcase the result follows from the fact that for $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C) = \sum_{b \in AType, \varphi(b) = a} Rate(E_j, b, l, [E]_{\sim_{\mathrm{MB}}})$$

(*iv*) Let $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ be a Markovian bisimulation such that $(E_1, E_2) \in \mathcal{B}$. Given $F \in \mathcal{G}$, we prove that

$$\mathcal{B}' = (\mathcal{B} \cup \{(E_1 + F, E_2 + F), (E_2 + F, E_1 + F)\})^+$$

is a Markovian bisimulation. Observed that $\mathcal{B}'$ is an equivalence relation, we have two cases:

- If $(E_1 + F, E_2 + F) \in \mathcal{B}$, then $\mathcal{B}' = \mathcal{B}$ and the result trivially follows.
- Assume that $(E_1 + F, E_2 + F) \notin \mathcal{B}$. Observed that

  $$\mathcal{G}/\mathcal{B}' = (\mathcal{G}/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}) \cup \{[E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}\}$$

  let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}'$.

  * If $(F_1, F_2) \in \mathcal{B}$ and $C \in \mathcal{G}/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}$, then trivially $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}$ and $C = [E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have $Rate(F_j, a, l, C) = Rate(F_j, a, l, [E_1 + F]_{\mathcal{B}}) + Rate(F_j, a, l, [E_2 + F]_{\mathcal{B}})$ so $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.
  * If $(F_1, F_2) \in \mathcal{B}' - \mathcal{B}$, i.e. $F_1 \in [E_1 + F]_{\mathcal{B}}$ and $F_2 \in [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

    $$Rate(F_j, a, l, C) = \begin{cases} Rate(E_j, a, l, C) + Rate(F, a, l, C) \\ Rate(E_j, a, l, C) \\ Rate(F, a, l, C) \\ \bot \end{cases}$$

    depending on whether $Rate(E_j, a, l, C) \neq \bot \wedge Rate(F, a, l, C) \neq \bot$ or $Rate(E_j, a, l, C) \neq \bot \wedge ((l \geq 0 \wedge Rate(F, a, l, C) = \bot) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{-}. l' \leq l \Longrightarrow Rate(F, a, l', \mathcal{G}) = \bot))$ or $Rate(F, a, l, C) \neq \bot \wedge ((l \geq 0 \wedge Rate(E_j, a, l, C) = \bot) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{-}. l' \leq l \Longrightarrow Rate(E_j, a, l', \mathcal{G}) = \bot))$ or none of the previous clauses holds.

    · If $C \in \mathcal{G}/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}$, then from $(E_1, E_2) \in \mathcal{B}$ we derive $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$ so $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

    · If $C = [E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have $Rate(E_j, a, l, C) = Rate(E_j, a, l, [E_1 + F]_{\mathcal{B}}) + Rate(E_j, a, l, [E_2 + F]_{\mathcal{B}})$. Since $(E_1, E_2) \in \mathcal{B}$, it turns out that $Rate(E_1, a, l, C) = Rate(E_2, a, l, C)$ so $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

(v) Given $F \in \mathcal{G}$ and $S \subseteq AType - \{\tau\}$, we prove that

$$\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}} \text{ where } \mathcal{B} = \{(E_1 \parallel_S F, E_2 \parallel_S F) \mid E_1 \sim_{\mathrm{MB}} E_2\}$$

is a Markovian bisimulation. Observed that $\mathcal{B}'$ is an equivalence relation and that either each of the terms of an equivalence class has "$\_ \parallel_S F$" as outermost operator or none of them has, let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/\mathcal{B}'$.

– If $(F_1, F_2) \in Id_{\mathcal{G}}$, then trivially $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

– If $(F_1, F_2) \in \mathcal{B}$, then $F_1 \equiv E_1 \parallel_S F$ and $F_2 \equiv E_2 \parallel_S F$ where $E_1 \sim_{\mathrm{MB}} E_2$.

  * If none of the terms in C has "$\_ \parallel_S F$" as outermost operator, then trivially $Rate(F_1, a, l, C) = \perp = Rate(F_2, a, l, C)$.

  * If each of the terms in C has "$\_ \parallel_S F$" as outermost operator, given $E \parallel_S G \in C$ it turns out that $C = \{E' \parallel_S G \mid E' \in [E]_{\sim_{\mathrm{MB}}}\}$.

    · If $a \notin S$, then for $j \in \{1, 2\}$ we have that

$$Rate(F_j, a, l, C) = \begin{cases} Rate(E_j, a, l, [E]_{\sim_{\mathrm{MB}}}) + Rate(F, a, l, \{G\}) \\ Rate(E_j, a, l, [E]_{\sim_{\mathrm{MB}}}) \\ Rate(F, a, l, \{G\}) \\ \perp \end{cases}$$

depending on whether $E_j \in [E]_{\sim_{\mathrm{MB}}} \wedge F \equiv G \wedge Rate(E_j, a, l, [E]_{\sim_{\mathrm{MB}}}) \neq \perp \wedge Rate(F, a, l, \{G\}) \neq \perp$ or $E_j \notin [E]_{\sim_{\mathrm{MB}}} \wedge F \equiv G \wedge Rate(E_j, a, l, [E]_{\sim_{\mathrm{MB}}}) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow Rate(F, a, l', \mathcal{G}) = \perp))$ or $E_j \in [E]_{\sim_{\mathrm{MB}}} \wedge F \not\equiv G \wedge Rate(F, a, l, \{G\}) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow Rate(E_j, a, l', [E]_{\sim_{\mathrm{MB}}}) = \perp$ or none of the previous clauses holds. Since $E_1 \sim_{\mathrm{MB}} E_2$, it follows that $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

    · If $a \in S$ and we pose for $l' \in \mathbb{Z}_-$

$$\begin{aligned} *_{-l', w_{E_j, l'}} &= Rate(E_j, a, l', [E]_{\sim_{\mathrm{MB}}}) \\ *_{-l', w_{E_j, l', \mathrm{tot}}} &= Rate(E_j, a, l', \mathcal{G}) \\ *_{-l', w_{F, l'}} &= Rate(F, a, l', \{G\}) \\ *_{-l', w_{F, l', \mathrm{tot}}} &= Rate(F, a, l', \mathcal{G}) \end{aligned}$$

and

$$N = \begin{cases} w_{E_j, l_1, \mathrm{tot}} + w_{F, l_2, \mathrm{tot}} & \text{if } l_1 = l_2 \\ w_{E_j, l_1, \mathrm{tot}} & \text{if } l_1 > l_2 \\ w_{F, l_2, \mathrm{tot}} & \text{if } l_2 > l_1 \end{cases}$$

then for $j \in \{1, 2\}$ we have that

$$Rate(F_j, a, l, C) = \begin{cases} Rate(E_j, a, l, [E]_{\sim_{\rm MB}}) \cdot w_{F,l'} / w_{F,l',\rm tot} + \\ \qquad Rate(F, a, l, \{G\}) \cdot w_{E_j,l''} / w_{E_j,l'',\rm tot} \\ Rate(E_j, a, l, [E]_{\sim_{\rm MB}}) \cdot w_{F,l'} / w_{F,l',\rm tot} \\ Rate(F, a, l, \{G\}) \cdot w_{E_j,l''} / w_{E_j,l'',\rm tot} \\ *_{-l,(w_{E_j,l_1}/w_{E_j,l_1,\rm tot})\cdot(w_{F,l_2}/w_{F,l_2,\rm tot})\cdot N} \\ \perp \end{cases}$$

depending on whether $l \geq 0 \wedge Rate(E_j, a, l, [E]_{\sim_{\rm MB}}) \neq \perp \wedge \exists l' \in \mathbb{Z}_-. Rate(F, a, l', \{G\}) \neq \perp \wedge Rate(F, a, l, \{G\}) \neq \perp \wedge \exists l'' \in \mathbb{Z}_-. Rate(E_j, a, l'', [E]_{\sim_{\rm MB}}) \neq \perp$ or $l \geq 0 \wedge Rate(E_j, a, l, [E]_{\sim_{\rm MB}}) \neq \perp \wedge \exists l' \in \mathbb{Z}_-. Rate(F, a, l', \{G\}) \neq \perp \wedge (Rate(F, a, l, \{G\}) = \perp \vee \forall l''' \in \mathbb{Z}_-. Rate(E_j, a, l''', [E]_{\sim_{\rm MB}}) = \perp)$ or $l \geq 0 \wedge Rate(F, a, l, \{G\}) \neq \perp \wedge \exists l'' \in \mathbb{Z}_-. Rate(E_j, a, l'', [E]_{\sim_{\rm MB}}) \neq \perp \wedge (Rate(E_j, a, l, [E]_{\sim_{\rm MB}}) = \perp \vee \forall l''' \in \mathbb{Z}_-. Rate(F, a, l''', \{G\}) = \perp)$ or $l < 0 \wedge \exists l_1, l_2 \in \mathbb{Z}_-. Rate(E_j, a, l_1, [E]_{\sim_{\rm MB}}) \neq \perp \wedge Rate(F, a, l_2, \{G\}) \neq \perp \wedge -l = \max(-l_1, -l_2)$ or none of the previous clauses holds. From $E_1 \sim_{\rm MB} E_2$, it follows that $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$. ■

Second, we demonstrate Theorem 11.4 that says that $\sim_{\rm MB}$ is a congruence w.r.t. recursive definitions. To do that, we have to preliminarily extend the definition of $\sim_{\rm MB}$ to open terms, i.e. to terms containing occurrences of undefined constants. Similarly to [122], the proof will be based on the notion of bisimulation up to. In the following, we denote by $Def : Const \longrightarrow \mathcal{L}$ a set of given constant defining equations and by "$\_\,st\,\_$" the relation subterm-of. We also let $\perp \leq \perp$ and $\perp < \tilde{\lambda}$ for all $\tilde{\lambda} \in ARate$.

Definition B.3 *The term $E\langle A := E' \rangle$ obtained from $E \in \mathcal{L}$ by replacing each occurrence of A with E', where $A \triangleq E' \in Def$, is defined by induction on the syntactical structure of E as follows:*

$$\begin{array}{rcl} \underline{0}\langle A := E' \rangle & \equiv & \underline{0} \\ (<a, \tilde{\lambda}>.E)\langle A := E' \rangle & \equiv & <a, \tilde{\lambda}>.E\langle A := E' \rangle \\ E/L\langle A := E' \rangle & \equiv & E\langle A := E' \rangle/L \\ E[\varphi]\langle A := E' \rangle & \equiv & E\langle A := E' \rangle[\varphi] \\ (E_1 + E_2)\langle A := E' \rangle & \equiv & E_1\langle A := E' \rangle + E_2\langle A := E' \rangle \\ (E_1 \,\|_S\, E_2)\langle A := E' \rangle & \equiv & E_1\langle A := E' \rangle \,\|_S\, E_2\langle A := E' \rangle \\ B\langle A := E' \rangle & \equiv & \begin{cases} E' & \text{if } B \equiv A \\ B & \text{if } B \not\equiv A \end{cases} \end{array}$$

■

Definition B.4 *The set of terms obtained from $E \in \mathcal{L}$ by repeatedly replacing constants by the right hand side terms of their defining equations in Def is defined by*

$$Subst_{Def}(E) = \bigcup_{n \in \mathbb{N}} Subst^n_{Def}(E)$$

where

$$Subst^n_{Def}(E) = \begin{cases} \{E\} \\ \{F \in \mathcal{L} \mid F \equiv G\langle A := E'\rangle \wedge G \in Subst^{n-1}_{Def}(E) \wedge A\, st\, G \wedge A \triangleq E' \in Def\} \end{cases}$$

depending on whether $n = 0$ *or* $n > 0$. ■

Definition B.5 *The set of constants occurring in* $E \in \mathcal{L}$ *w.r.t. Def is defined by*

$$Const_{Def}(E) = \{A \in Const \mid \exists F \in Subst_{Def}(E).\, A\, st\, F\}$$

■

Definition B.6 $A \in Const$ *is free w.r.t. Def if and only if, for all* $E \in \mathcal{L}$, $A \triangleq E \notin Def$. *We denote by* $Free_{Def}$ *the set of constants that are free w.r.t. Def.* ■

Definition B.7 $E \in \mathcal{L}$ *is partially closed and guarded (pcg) w.r.t. Def if and only if for each* $A \in Const_{Def}(E)$ *either* $A \in Free_{Def}$ *or*

- A *is equipped in Def with a defining equation, say* $A \triangleq E'$, *and*
- *there exists* $F \in Subst_{Def}(E')$ *such that, whenever an instance of* $B \notin Free_{Def}$ *satisfies* $B\, st\, F$, *then the same instance satisfies* $B\, st <a, \tilde{\lambda}>.G\, st\, F$. ■

Definition B.8 *The term* $E\langle\langle A := B\rangle\rangle$ *obtained from* $E \in \mathcal{L}$ *by replacing each occurrence of* $A \in Free_{Def}$ *with* $B \in \mathcal{G}$ *is defined by induction on the syntactical structure of* E *as follows:*

$$\begin{array}{rcl}
\underline{0}\langle\langle A := B\rangle\rangle & \equiv & \underline{0} \\
(<a, \tilde{\lambda}>.E)\langle\langle A := B\rangle\rangle & \equiv & <a, \tilde{\lambda}>.E\langle\langle A := B\rangle\rangle \\
E/L\langle\langle A := B\rangle\rangle & \equiv & E\langle\langle A := B\rangle\rangle/L \\
E[\varphi]\langle\langle A := B\rangle\rangle & \equiv & E\langle\langle A := B\rangle\rangle[\varphi] \\
(E_1 + E_2)\langle\langle A := B\rangle\rangle & \equiv & E_1\langle\langle A := B\rangle\rangle + E_2\langle\langle A := B\rangle\rangle \\
(E_1 \,\|_S\, E_2)\langle\langle A := B\rangle\rangle & \equiv & E_1\langle\langle A := B\rangle\rangle \,\|_S\, E_2\langle\langle A := B\rangle\rangle \\
A'\langle\langle A := B\rangle\rangle & \equiv & \begin{cases} B & \text{if } A' \equiv A \\ A' & \text{if } A' \not\equiv A \wedge A' \in Free_{Def} \\ A'' & \text{if } A' \not\equiv A \wedge A' \triangleq E \in Def \wedge A'' \triangleq E\langle\langle A := B\rangle\rangle \in Def \end{cases}
\end{array}$$

■

Definition B.9 *Let* $E_1, E_2 \in \mathcal{L}$ *be pcg and suppose that* $Const_{Def}(E_1) \cup Const_{Def}(E_2)$ *contains* $\{A_i \in Const \mid i \in I\}$ *as free constants w.r.t. Def. We pose* $E_1 \sim_{\mathrm{MB}} E_2$ *if and only if, for all sets* $\{B_i \in \mathcal{G} \mid i \in I\}$ *such that* $E_1\langle\langle A_i := B_i\rangle\rangle_{i\in I}, E_2\langle\langle A_i := B_i\rangle\rangle_{i\in I} \in \mathcal{G}$, *it turns out that*

$$E_1\langle\langle A_i := B_i\rangle\rangle_{i\in I} \sim_{\mathrm{MB}} E_2\langle\langle A_i := B_i\rangle\rangle_{i\in I}$$

■

We are now in a position to prove Theorem 11.4.

Proof of Theorem 11.4 Let $E_1, E_2 \in \mathcal{L}$ be pcg and suppose that $Const_{Def}(E_1) \cup Const_{Def}(E_2)$ contains only $A \in Const$ as a free constant w.r.t. Def. Let $A_1 \stackrel{\Delta}{=} E_1\langle\langle A := A_1\rangle\rangle$, $A_2 \stackrel{\Delta}{=} E_2\langle\langle A := A_2\rangle\rangle \in Def$ with $A_1, A_2 \in \mathcal{G}$. We have to prove that $A_1 \sim_{\rm MB} A_2$ whenever $E_1 \sim_{\rm MB} E_2$. To achieve this, it suffices to prove that

$$\begin{aligned}\mathcal{B} = \{(F_1, F_2) \in \mathcal{G} \times \mathcal{G} \mid F_1 \equiv F\langle\!\langle B := A_1\rangle\!\rangle \wedge F_2 \equiv F\langle\!\langle B := A_2\rangle\!\rangle \wedge \\ F \in \mathcal{L} \text{ pcg with at most } B \in Free_{Def}\}\end{aligned}$$

is a Markovian bisimulation up to $\sim_{\rm MB}$: the result will follow by taking $F \equiv B$. Let $\mathcal{B}' = \mathcal{B} \cup \mathcal{B}^{-1}$. Given $(F_1, F_2) \in \mathcal{B}$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\rm MB})^+$, we must prove that $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$.

We start by showing that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$ for all $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}/(\mathcal{B}' \cup \sim_{\rm MB})^+$. If $Rate(F_1, a, l, C) = \perp$ there is nothing to prove, otherwise we proceed by induction on the maximum depth d of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C.

- If $d = 1$ then only the rule for the action prefix operator has been used to deduce the only existing transition. Therefore $F \equiv <a, \tilde{\lambda}>.F'$ with $PL(\tilde{\lambda}) = l$ and for $j \in \{1, 2\}$ we have $F_j \equiv <a, \tilde{\lambda}>.F'\langle\!\langle B := A_j\rangle\!\rangle$. Since $(F'\langle\!\langle B := A_1\rangle\!\rangle, F'\langle\!\langle B := A_2\rangle\!\rangle) \in \mathcal{B}$, it turns out that $C = [F'\langle\!\langle B := A_1\rangle\!\rangle]_{(\mathcal{B}' \cup \sim_{\rm MB})^+} = [F'\langle\!\langle B := A_2\rangle\!\rangle]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ hence $Rate(F_1, a, l, C) = \tilde{\lambda} = Rate(F_2, a, l, C)$.

- If $d > 1$ then several subcases arise depending on the syntactical structure of F.

  - If $F \equiv F'/L$ then for $j \in \{1, 2\}$ we have $F_j \equiv F'\langle\!\langle B := A_j\rangle\!\rangle/L$. Since F_1 has a transition having derivative term in C, there exists $G \in \mathcal{G}$ such that G/L is the derivative term and $C = [G/L]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$. Because of the congruence property of $\sim_{\rm MB}$ and $\mathcal{B}$ w.r.t. "$\_/L$", we have that $H/L \in [G/L]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ implies $[H]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}/L \subseteq [G/L]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$. [1] As a consequence $[G/L]_{(\mathcal{B}' \cup \sim_{\rm MB})^+} = (\cup_i C_i/L) \cup D$ where $C_i = [G_i]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ are distinguished equivalence classes with $G_i/L \in [G/L]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ and D is a set of terms not having "$\_/L$" as outermost operator, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, for any i either (i) $F'\langle\!\langle B := A_1\rangle\!\rangle$ has no transitions having type a, priority level l, and derivative term in C_i, or (ii) $d - 1$ is the maximum depth of the inferences of the transitions for $F'\langle\!\langle B := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C_i. For $j \in \{1, 2\}$ we have

  $$Rate(F_j, a, l, C) = \sum_i Rate(F_j, a, l, C_i/L)$$

  where for any i

[1] Given a set of terms $\mathcal{T}$, we denote by $\mathcal{T}/L$ the set of terms $\{E/L \mid E \in \mathcal{T}\}$.

$$Rate(F_j, a, l, C_i/L) = \begin{cases} Rate(F'\langle\!\langle B{:=}A_j\rangle\!\rangle, a, l, C_i) \\ Rate(F'\langle\!\langle B{:=}A_j\rangle\!\rangle, \tau, l, C_i) + \sum_{b \in L} Rate(F'\langle\!\langle B{:=}A_j\rangle\!\rangle, b, l, C_i) \end{cases}$$

depending on whether $a \notin L \cup \{\tau\}$ or $a = \tau$. By applying the induction hypothesis to each $F'\langle\!\langle B := A_1\rangle\!\rangle$, b (with $b \in L \cup \{\tau\}$ if $a = \tau$ and $b = a$ otherwise), l, and C_i such that $Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, b, l, C_i) \neq \bot$, we have that $Rate(F_1, a, l, C_i/L) \leq Rate(F_2, a, l, C_i/L)$ for any i. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

– If $F \equiv F'[\varphi]$ then the proof is similar to the one developed in the previous case. The result follows by applying the induction hypothesis to the fact that for $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C_i[\varphi]) = \sum_{b \in AType, \varphi(b) = a} Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, b, l, C_i)$$

– If $F \equiv F' + F''$ then for $j \in \{1, 2\}$ we have $F_j \equiv F'\langle\!\langle B := A_j\rangle\!\rangle + F''\langle\!\langle B := A_j\rangle\!\rangle$. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, for any $G \in \{F'\langle\!\langle B := A_1\rangle\!\rangle, F''\langle\!\langle B := A_1\rangle\!\rangle\}$ either (i) G has no transitions having type a, priority level l, and derivative term in C, or (ii) $d - 1$ is the maximum depth of the inferences of the transitions for G having type a, priority level l, and derivative term in C. For $j \in \{1, 2\}$ we have:

$$Rate(F_j, a, l, C) = \begin{cases} Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) + Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \\ Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \\ Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \\ \bot \end{cases}$$

depending on whether $Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \neq \bot \wedge Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \neq \bot$ or $Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \neq \bot \wedge ((l \geq 0 \wedge Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) = \bot) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{-}. l' \leq l \Longrightarrow Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l', \mathcal{G}) = \bot))$ or $Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) \neq \bot \wedge ((l \geq 0 \wedge Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C) = \bot) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{-}. l' \leq l \Longrightarrow Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l', \mathcal{G}) = \bot))$ or none of the previous clauses holds. By applying the induction hypothesis to $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C if $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \neq \bot$, and to $F''\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C if $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \neq \bot$, we have that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \leq Rate(F'\langle\!\langle B := A_2\rangle\!\rangle, a, l, C)$ and $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \leq Rate(F''\langle\!\langle B := A_2\rangle\!\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

– If $F \equiv F' \,\|_S\, F''$ then for $j \in \{1, 2\}$ we have $F_j \equiv F'\langle\!\langle B := A_j\rangle\!\rangle \,\|_S\, F''\langle\!\langle B := A_j\rangle\!\rangle$. Since F_1 has a transition having derivative term in C, there exist $G', G'' \in \mathcal{G}$ such that $G' \,\|_S\, G''$ is the derivative term and $C = [G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$.

  * If $a \notin S$ then either $G'' \equiv F''\langle\!\langle B := A_1\rangle\!\rangle$ or $G' \equiv F'\langle\!\langle B := A_1\rangle\!\rangle$. Because of the congruence property of $\sim_{\rm MB}$ and $\mathcal{B}$ w.r.t. "$\_ \,\|_S\, \_$", we have $H \,\|_S\, F''\langle\!\langle B := A_1\rangle\!\rangle \in [G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+} \Longrightarrow$

$[H]_{(\mathcal{B}'\cup\sim_{\rm MB})^+} \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ and $F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S H \in [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+} \Longrightarrow F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S [H]_{(\mathcal{B}'\cup\sim_{\rm MB})^+} \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$. Moreover, due to the congruence property of $\mathcal{B}$ w.r.t. "$\_ \parallel_S \_$" we have that, for all $\mathcal{T} \subseteq \mathcal{G}$, $\mathcal{T} \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ if and only if $\mathcal{T} \parallel_S F''\langle\!\langle B := A_2\rangle\!\rangle \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ and $F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S \mathcal{T} \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ if and only if $F'\langle\!\langle B := A_2\rangle\!\rangle \parallel_S \mathcal{T} \subseteq [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$. As a consequence $[G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+} = (\cup_i C'_i \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle) \cup (\cup_i C'_i \parallel_S F''\langle\!\langle B := A_2\rangle\!\rangle) \cup (\cup_j F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S C''_j) \cup (\cup_j F'\langle\!\langle B := A_2\rangle\!\rangle \parallel_S C''_j) \cup D$ where $C'_i = [H'_i]_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ are distinguished equivalence classes with $H'_i \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle \in [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ and $C''_j = [H''_j]_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ are distinct equivalence classes with $F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S H''_j \in [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$. Besides, classes C'_i and C''_j are such that if $F_1 \equiv F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle \in [G' \parallel_S G'']_{(\mathcal{B}'\cup\sim_{\rm MB})^+}$ then there exist i such that $F'\langle\!\langle B := A_1\rangle\!\rangle \in C'_i$ and j such that $F''\langle\!\langle B := A_1\rangle\!\rangle \in C''_j$. Moreover D is a set of terms not of the form $H \parallel_S F''\langle\!\langle B := A_j\rangle\!\rangle$ or $F'\langle\!\langle B := A_j\rangle\!\rangle \parallel_S H$ for any term H, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, considered term $F'\langle\!\langle B := A_1\rangle\!\rangle$ ($F''\langle\!\langle B := A_1\rangle\!\rangle$) we have that for any i (for any j) either (i) $F'\langle\!\langle B := A_1\rangle\!\rangle$ ($F''\langle\!\langle B := A_1\rangle\!\rangle$) has no transitions having type a, priority level l, and derivative term in C'_i (C''_j), or (ii) $d-1$ is the maximum depth of the inferences of the transitions for $F'\langle\!\langle B := A_1\rangle\!\rangle$ ($F''\langle\!\langle B := A_1\rangle\!\rangle$) having type a, priority level l, and derivative term in C'_i (C''_j). Observed that for $k \in \{1,2\}$ we have for any i, j

$$\begin{array}{rcl} Rate(F_k, a, l, C'_i \parallel_S F''\langle\!\langle B := A_k\rangle\!\rangle) & = & Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) \\ Rate(F_k, a, l, F'\langle\!\langle B := A_k\rangle\!\rangle \parallel_S C''_j) & = & Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) \end{array}$$

for $k \in \{1,2\}$ we have

$$Rate(F_k, a, l, C) = \begin{cases} \sum_i Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) + \\ \qquad\qquad \sum_j Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) \\ \sum_i Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) \\ \sum_j Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) \\ \perp \end{cases}$$

depending on whether $\exists i.\, Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) \neq \perp \wedge \exists j.\, Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) \neq \perp$ or $\exists i.\, Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall j. \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) = \perp))$ or $\exists j.\, Rate(F''\langle\!\langle B := A_k\rangle\!\rangle, a, l, C''_j) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall i. \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow Rate(F'\langle\!\langle B := A_k\rangle\!\rangle, a, l, C'_i) = \perp$ or none of the previous clauses holds. By applying the induction hypothesis to each $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C'_i such that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C'_i) \neq \perp$ and to each $F''\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C''_j such that $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l, C''_j) \neq \perp$, we have that $Rate(F_1, a, l, C'_i \parallel_S F''\langle\!\langle B := A_1\rangle\!\rangle) \leq Rate(F_2, a, l, C'_i \parallel_S F''\langle\!\langle B := A_2\rangle\!\rangle)$ for any i and that $Rate(F_1, a, l, F'\langle\!\langle B := A_1\rangle\!\rangle \parallel_S C''_j) \leq$

$Rate(F_2, a, l, F'\langle\!\langle B := A_2\rangle\!\rangle \,\|_S\, C''_j)$ for any j. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

* Let $a \in S$. Because of the congruence property of $\sim_{\rm MB}$ and $\mathcal{B}$ w.r.t. "$\_ \,\|_S\, \_$", we have that $H' \,\|_S\, H'' \in [G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+} \implies [H']_{(\mathcal{B}' \cup \sim_{\rm MB})^+} \,\|_S [H'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+} \subseteq [G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$.[2] As a consequence $[G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+} = (\cup_i C'_i \,\|_S\, C''_i) \cup D$ where $C'_i = [G'_i]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ and $C''_i = [G''_i]_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ are distinguished pairs of equivalence classes with $G'_i \,\|_S\, G''_i \in [G' \,\|_S\, G'']_{(\mathcal{B}' \cup \sim_{\rm MB})^+}$ and D is a set of terms not having "$\_ \,\|_S\, \_$" as outermost operator, hence not reachable from F_1 or F_2. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, for any i if there exists $l' \in \mathbb{Z}_-$ such that $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l', C''_i) \neq \bot$ ($Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l', C'_i) \neq \bot$) then either (i) $F'\langle\!\langle B := A_1\rangle\!\rangle$ ($F''\langle\!\langle B := A_1\rangle\!\rangle$) has no transitions having type a, priority level l, and derivative term in C'_i (C''_i), or (ii) $d-1$ is the maximum depth of the inferences of the transitions for $F'\langle\!\langle B := A_1\rangle\!\rangle$ ($F''\langle\!\langle B := A_1\rangle\!\rangle$) having type a, priority level l, and derivative term in C'_i (C''_i). Observed that for $j \in \{1,2\}$

$$Rate(F_j, a, l, C) = \sum_i Rate(F_j, a, l, C'_i \,\|_S\, C''_i)$$

if we pose for $l' \in \mathbb{Z}_-$

$$\begin{array}{rcl}
*_{-l', w_{F',l'}} & = & Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l', C'_i) \\
*_{-l', w_{F',l',\rm tot}} & = & Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l', \mathcal{G}) \\
*_{-l', w_{F'',l'}} & = & Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l', C''_i) \\
*_{-l', w_{F'',l',\rm tot}} & = & Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l', \mathcal{G})
\end{array}$$

and

$$N = \begin{cases} w_{F',l_1,\rm tot} + w_{F'',l_2,\rm tot} & \text{if } l_1 = l_2 \\ w_{F',l_1,\rm tot} & \text{if } l_1 > l_2 \\ w_{F'',l_2,\rm tot} & \text{if } l_2 > l_1 \end{cases}$$

then for $j \in \{1,2\}$ we have for any i

$$Rate(F_j, a, l, C'_i \,\|_S\, C''_i) = \begin{cases} Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C'_i) \cdot w_{F'',l'} / w_{F'',l',\rm tot} + \\ \quad Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C''_i) \cdot w_{F',l''} / w_{F',l'',\rm tot} \\ Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C'_i) \cdot w_{F'',l'} / w_{F'',l',\rm tot} \\ Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C''_i) \cdot w_{F',l''} / w_{F',l'',\rm tot} \\ *_{-l, (w_{F',l_1}/w_{F',l_1,\rm tot}) \cdot (w_{F'',l_2}/w_{F'',l_2,\rm tot}) \cdot N} \\ \bot \end{cases}$$

depending on whether $l \geq 0 \wedge Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C'_i) \neq \bot \wedge \exists l' \in \mathbb{Z}_-. Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l', C''_i) \neq \bot \wedge Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C''_i) \neq \bot \wedge \exists l'' \in \mathbb{Z}_-. Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l'',$

[2] Given two sets of terms $\mathcal{T}_1$ and $\mathcal{T}_2$, we denote by $\mathcal{T}_1 \,\|_S\, \mathcal{T}_2$ the set of terms $\{E_1 \,\|_S\, E_2 \mid E_1 \in \mathcal{T}_1 \wedge E_2 \in \mathcal{T}_2\}$.

$C_i') \neq \bot$ or $l \geq 0 \wedge Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i') \neq \bot \wedge \exists l' \in \mathbb{Z}_-.\, Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l', C_i'') \neq \bot \wedge (Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i'') = \bot \vee \forall l''' \in \mathbb{Z}_-.\, Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l''', C_i') = \bot)$ or $l \geq 0 \wedge Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i'') \neq \bot \wedge \exists l'' \in \mathbb{Z}_-.\, Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l'', C_i') \neq \bot \wedge (Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C_i') = \bot \vee \forall l''' \in \mathbb{Z}_-.\, Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l''', C_i'') = \bot)$ or $l < 0 \wedge \exists l_1, l_2 \in \mathbb{Z}_-.\, Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l_1, C_i') \neq \bot \wedge Rate(F''\langle\!\langle B := A_j\rangle\!\rangle, a, l_2, C_i'') \neq \bot \wedge -l = \max(-l_1, -l_2)$ or none of the previous clauses holds. By applying the induction hypothesis to each $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C_i' such that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C_i') \neq \bot$ and to each $F''\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C_i'' such that $Rate(F''\langle\!\langle B := A_1\rangle\!\rangle, a, l, C_i'') \neq \bot$, we have that $Rate(F_1, a, l, C_i' \,\|_S\, C_i'') \leq Rate(F_2, a, l, C_i' \,\|_S\, C_i'')$ for any i. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

– If $F \equiv B'$ then for $j \in \{1, 2\}$ we have $F_j \equiv B'\langle\!\langle B := A_j\rangle\!\rangle$.

  ∗ If $B' \equiv B$ then for $j \in \{1, 2\}$ we have $F_j \equiv A_j$. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, then $d - 1$ is the maximum depth of the inferences of the transitions for $E_1\langle\!\langle A := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C. For $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C) = Rate(E_j\langle\!\langle A := A_j\rangle\!\rangle, a, l, C)$$

  By applying the induction hypothesis to $E_1\langle\!\langle A := A_1\rangle\!\rangle$, a, l, and C we have that $Rate(E_1\langle\!\langle A := A_1\rangle\!\rangle, a, l, C) \leq Rate(E_1\langle\!\langle A := A_2\rangle\!\rangle, a, l, C)$. From the fact that $E_1 \sim_{\mathrm{MB}} E_2$ we derive $Rate(E_1\langle\!\langle A := A_2\rangle\!\rangle, a, l, C) = Rate(E_2\langle\!\langle A := A_2\rangle\!\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

  ∗ If $B' \not\equiv B$ then let $B' \triangleq F'$ be the defining equation of B'. Since d is the maximum depth of the inferences of the transitions for F_1 having type a, priority level l, and derivative term in C, then $d - 1$ is the maximum depth of the inferences of the transitions for $F'\langle\!\langle B := A_1\rangle\!\rangle$ having type a, priority level l, and derivative term in C. For $j \in \{1, 2\}$ we have

$$Rate(F_j, a, l, C) = Rate(F'\langle\!\langle B := A_j\rangle\!\rangle, a, l, C)$$

  By applying the induction hypothesis to $F'\langle\!\langle B := A_1\rangle\!\rangle$, a, l, and C we have that $Rate(F'\langle\!\langle B := A_1\rangle\!\rangle, a, l, C) \leq Rate(F'\langle\!\langle B := A_2\rangle\!\rangle, a, l, C)$. It follows that $Rate(F_1, a, l, C) \leq Rate(F_2, a, l, C)$.

By applying a symmetric argument we also have that $Rate(F_2, a, l, C) \leq Rate(F_1, a, l, C)$, hence $Rate(F_1, a, l, C) = Rate(F_2, a, l, C)$. ■

B.3 Soundness and Completeness of the Axiomatization of $\sim_{\mathrm{MB}}$

In this appendix we demonstrate the soundness and the completeness of the axiomatization of $\sim_{\mathrm{MB}}$ shown in Table 11.6 for the set $\mathcal{G}_{\mathrm{nrec}}$ of nonrecursive terms of $\mathcal{G}$. This is achieved by means of the standard technique based on the idea of preliminarily reducing the terms in some suitable normal form.

Definition B.10 *$E \in \mathcal{G}_{\rm nrec}$ is in sum normal form (snf) if and only if $E \equiv \sum_{i \in I} {<} a_i, \tilde{\lambda}_i {>}.E_i$ where:*

- *either $I = \emptyset$ (hence $E \equiv \underline{0}$),*
- *or $I \neq \emptyset$ and for all $i \in I$ E_i is in snf.* ■

Definition B.11 *Function $size : \mathcal{G}_{\rm nrec} \longrightarrow \mathbb{N}_+$ is defined by structural induction as follows:*

- $size(\underline{0}) = 1$;
- $size({<}a, \tilde{\lambda}{>}.E) = 1 + size(E)$;
- $size(E/L) = size(E[\varphi]) = size(E)$;
- $size(E_1 + E_2) = \max(size(E_1), size(E_2))$;
- $size(E_1 \,\|_S\, E_2) = size(E_1) + size(E_2)$. ■

We observe that $size(E) \geq 1$ for all $E \in \mathcal{G}_{\rm nrec}$ and that $size(E_1) \geq size(E_2)$ for all axioms $E_1 = E_2$ in Table 11.6.

Lemma B.4 *For every $E \in \mathcal{G}_{\rm nrec}$ there exixts $F \in \mathcal{G}_{\rm nrec}$ in snf such that $\mathcal{A} \vdash E = F$.*

Proof: We proceed by induction on $size(E)$ as follows:

- Let $size(E) = 1$. The result follows by proving $\mathcal{A} \vdash E = \underline{0}$ by induction on the syntactical structure of E:
  - If $E \equiv \underline{0}$, then we take $F \equiv \underline{0}$ and the result follows by reflexivity.
  - The case $E \equiv {<}a, \tilde{\lambda}{>}.E'$ is not possible because it contradicts the hypothesis $size(E) = 1$.
  - If $E \equiv E'/L$, then E' is a subterm of E such that $size(E') = 1$, hence $\mathcal{A} \vdash E' = \underline{0}$ by structural induction. The result follows by substitutivity, $\mathcal{A}_6$, and transitivity.
  - If $E \equiv E'[\varphi]$, then the result can be proved by proceeding as in the previous case and by exploiting $\mathcal{A}_{10}$.
  - If $E \equiv E_0 + E_1$, then E_0 and E_1 are subterms of E such that $size(E_0) = 1$ and $size(E_1) = 1$, hence $\mathcal{A} \vdash E_0 = \underline{0}$ and $\mathcal{A} \vdash E_1 = \underline{0}$ by structural induction. The result follows by substitutivity, $\mathcal{A}_3$, and transitivity.
  - The case $E \equiv E_0 \,\|_S\, E_1$ is not possible because it contradicts the hypothesis $size(E) = 1$.
- Let the result hold whenever $size(E) \leq n \in \mathbb{N}_+$, and assume $size(E) = n + 1$. The result follows by proceeding by induction on the syntactical structure of E:

– The case $E \equiv \underline{0}$ is not possible because it contradicts the hypothesis $\mathit{size}(E) = n + 1 \geq 2$.

– If $E \equiv \,<a, \tilde{\lambda}>.E'$, then $\mathit{size}(E') = n$ hence by the induction hypothesis there exists $F' \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E' = F'$. The result follows by substitutivity.

– If $E \equiv E'/L$, then E' is a subterm of E hence by structural induction there exists $F' \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E' = F'$. By substitutivity we obtain $\mathcal{A} \vdash E = F'/L$. Assuming $F' \equiv \sum_{i \in I} <a_i, \tilde{\lambda}_i>.F'_i$ where every F'_i is in snf, by exploiting $\mathcal{A}_9$, $\mathcal{A}_8$, $\mathcal{A}_7$, and transitivity we obtain $\mathcal{A} \vdash F'/L = \sum_{i \in I}(<a_i, \tilde{\lambda}_i>.F'_i)/L = \sum_{i \in I, a_i \in L} <\tau, \tilde{\lambda}_i>.(F'_i/L) + \sum_{i \in I, a_i \notin L} <a_i, \tilde{\lambda}_i>.(F'_i/L)$. Since for every $i \in I$ we have $\mathit{size}(F'_i/L) = \mathit{size}(F'_i) < \mathit{size}(F') = \mathit{size}(F'/L) \leq \mathit{size}(E) = n + 1$, by the induction hypothesis it follows that for every $i \in I$ there exists $F''_i \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash F'_i/L = F''_i$. If we take $F \equiv \sum_{i \in I, a_i \in L} <\tau, \tilde{\lambda}_i>.F''_i + \sum_{i \in I, a_i \notin L} <a_i, \tilde{\lambda}_i>.F''_i$, then the result follows by substitutivity.

– If $E \equiv E'[\varphi]$, then the result can be proved by proceeding as in the previous case and by exploiting $\mathcal{A}_{12}$ and $\mathcal{A}_{11}$.

– If $E \equiv E_0 + E_1$, then E_0 and E_1 are subterms of E hence by structural induction there exist $F_0, F_1 \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_0 = F_0$ and $\mathcal{A} \vdash E_1 = F_1$. By substitutivity we obtain $\mathcal{A} \vdash E = F_0 + F_1$ and the result follows after a possible application of $\mathcal{A}_3$.

– If $E \equiv E_0 \,\|_S\, E_1$, then $\mathit{size}(E_0) \leq n$ and $\mathit{size}(E_1) \leq n$ hence by the induction hypothesis there exist $F_0, F_1 \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_0 = F_0$ and $\mathcal{A} \vdash E_1 = F_1$. By substitutivity we obtain $\mathcal{A} \vdash E = F_0 \,\|_S\, F_1$. There are three cases:

  * If $\mathit{size}(E_0) = \mathit{size}(E_1) = 1$, then $F_0 \equiv F_1 \equiv \underline{0}$ hence $\mathcal{A} \vdash E = \underline{0}$ by $\mathcal{A}_{13}$, $\mathcal{A}_3$, and transitivity.
  * If $\mathit{size}(E_0) = 1$ and $\mathit{size}(E_1) > 1$, then $F_0 \equiv \underline{0}$ and $F_1 \equiv \sum_{i \in I_1} <a_i, \tilde{\lambda}_i>.F_{1,i}$ where every $F_{1,i}$ is in snf. By $\mathcal{A}_{13}$, $\mathcal{A}_3$, and transitivity we obtain $\mathcal{A} \vdash F_0 \,\|_S\, F_1 = \sum_{j \in I_1, a_j \notin S} <a_j, \tilde{\lambda}_j>.(\underline{0} \,\|_S\, F_{1,j})$. Since for every $j \in I_1$ such that $a_j \notin S$ we have $\mathit{size}(\underline{0} \,\|_S\, F_{1,j}) < \mathit{size}(\underline{0} \,\|_S\, F_1) \leq \mathit{size}(E) = n + 1$, by the induction hypothesis it follows that for every $j \in I_1$ such that $a_j \notin S$ there exists $F'_{1,j}$ in snf such that $\mathcal{A} \vdash \underline{0} \,\|_S\, F_{1,j} = F'_{1,j}$, hence the result follows by substitutivity. The symmetric case is treated in the same way.
  * If $\mathit{size}(E_0) > 1$ and $\mathit{size}(E_1) > 1$, then $F_0 \equiv \sum_{i \in I_0} <a_i, \tilde{\lambda}_i>.F_{0,i}$ and $F_1 \equiv \sum_{i \in I_1} <a_i, \tilde{\lambda}_i>.F_{1,i}$ where every $F_{0,i}$ and $F_{1,i}$ is in snf. By $\mathcal{A}_{13}$ and transitivity we get $\mathcal{A} \vdash F_0 \,\|_S\, F_1 = \sum_{j \in I_0, a_j \notin S} <a_j, \tilde{\lambda}_j>.(F_{0,j} \,\|_S\, F_1) + \sum_{j \in I_1, a_j \notin S} <a_j, \tilde{\lambda}_j>.(F_0 \,\|_S\, F_{1,j}) + \sum_{k \in K_0, h \in P_{1,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h/W_{1,a_k})>.(F_{0,k} \,\|_S\, F_{1,h}) + \sum_{k \in K_1, h \in P_{0,a_k}} <a_k, \tilde{\lambda}_k \cdot (w_h/W_{0,a_k})>.(F_{0,h} \,\|_S\, F_{1,k}) + \sum_{k \in P'_0, h \in P_{1,a_k}} <a_k, *_{\max(l_k, l_h), (w_k/W_{0,a_k}) \cdot (w_h/W_{1,a_k}) \cdot N_{a_k}}>.(F_{0,k} \,\|_S\, F_{1,h})$. Since the size of every term surrounded by parentheses is at most n, by the induction hypothesis each such term can be proved equal via $\mathcal{A}$ to a term in snf, hence the result follows by substitutivity.

$\square$

Proof of Theorem 11.5 We have to prove that $\mathcal{A} \vdash E_1 = E_2 \iff E_1 \sim_{\mathrm{MB}} E_2$ for all $E_1, E_2 \in \mathcal{G}_{\mathrm{nrec}}$.

($\Longrightarrow$) It is a straightforward consequence of the fact that reflexivity, symmetry, transitivity and substitutivity of $Ded(\mathcal{A})$ are matched by the reflexive, symmetric, transitive and congruence properties of $\sim_{\mathrm{MB}}$, and the fact that every axiom in $\mathcal{A}$ can be restated as a property of $\sim_{\mathrm{MB}}$.

($\Longleftarrow$) Assume $E_1 \sim_{\mathrm{MB}} E_2$. There are two cases:

- If E_1 and E_2 are both in snf, the result follows by proceeding by induction on $size(E_1)$:
  - If $size(E_1) = 1$, then $E_1 \equiv \underline{0} \equiv E_2$ since they are both in snf. The result follows by reflexivity.
  - If $size(E_1) > 1$, then $E_1 \equiv \sum_{i \in I_1} <a_{1,i}, \tilde{\lambda}_{1,i}>.E_{1,i}$ and $E_2 \equiv \sum_{i \in I_2} <a_{2,i}, \tilde{\lambda}_{2,i}>.E_{2,i}$. It is not restrictive to assume that for $k \in \{1, 2\}$ and $i, j \in I_k$ it holds

    $$a_{k,i} = a_{k,j} \wedge PL(\tilde{\lambda}_{k,i}) = PL(\tilde{\lambda}_{k,j}) \wedge E_{k,i} \sim_{\mathrm{MB}} E_{k,j} \Longrightarrow i = j$$
    $$a_{k,i} = a_{k,j} \wedge \tilde{\lambda}_{k,i} = *_{l_{k,i}, w_{k,i}} \wedge \tilde{\lambda}_{k,j} = *_{l_{k,j}, w_{k,j}} \Longrightarrow l_{k,i} = l_{k,j}$$

    because if this were not the case, then it would suffice to resort to finitely many applications of $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_4$, and $\mathcal{A}_5$. Since $E_1 \sim_{\mathrm{MB}} E_2$, we have that $|I_1| = |I_2|$ and for every summand $<a_{1,i}, \tilde{\lambda}_{1,i}>.E_{1,i}$ in E_1 there exists exactly one summand $<a_{2,j}, \tilde{\lambda}_{2,j}>.E_{2,j}$ in E_2 such that $a_{1,i} = a_{2,j} \wedge \tilde{\lambda}_{1,i} = \tilde{\lambda}_{2,j} \wedge E_{1,i} \equiv E_{2,j}$ (hence $E_{1,i} \sim_{\mathrm{MB}} E_{2,j}$), and vice versa. By the induction hypothesis we obtain that $\mathcal{A} \vdash E_{1,i} = E_{2,j}$, and the result follows by substitutivity.
- If E_1 or E_2 is not in snf, then by virtue of Lemma B.4 there exist $F_1, F_2 \in \mathcal{G}_{\mathrm{nrec}}$ in snf such that $\mathcal{A} \vdash E_1 = F_1$ and $\mathcal{A} \vdash E_2 = F_2$. We then obtain $E_1 \sim_{\mathrm{MB}} F_1$ and $E_2 \sim_{\mathrm{MB}} F_2$ by soundness and $F_1 \sim_{\mathrm{MB}} F_2$ by the transitive property, hence $\mathcal{A} \vdash F_1 = F_2$ by the previous point. By transitivity we finally obtain $\mathcal{A} \vdash E_1 = E_2$. ■

Appendix C

Technical Machinery for Chapter 12

In this appendix we present the proofs of Theorems 12.1 and 12.4 of Chapter 12: the congruence property of the RY-Markovian bisimulation equivalence of order 1 $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$, which aggregates yield rewards, and the congruence property of the RYB-Markovian bisimulation equivalence of order 1 $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$ which aggregates yield and bonus rewards.

C.1 Congruence Property of $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$

In this section we provide the proof of Theorem 12.1 which says that $\sim_{\mathrm{MB}_1^{\mathrm{RY}}}$ is a congruence w.r.t. all the operators of $\mathrm{EMPA}_{\mathrm{gr}_1}$ and recursive constant definitions.

Proof of Theorem 12.1 The proof is similar to that of the corresponding Theorems 11.3 and 11.4 of Chapter 11 (presented in Appendix B.2) with some changes in the case of the alternative and parallel composition operators that we now show. In the following, the extended real number summation and multiplication of Definition 12.6 and 12.4, respectively, are also used to express the componentwise, extended real number vector summation and multiplication.

- Let $\mathcal{B} \subseteq \mathcal{G}_1 \times \mathcal{G}_1$ be a RY-Markovian bisimulation of order 1 such that $(E_1, E_2) \in \mathcal{B}$. Given $F \in \mathcal{G}_1$, we prove that $\mathcal{B}' = (\mathcal{B} \cup \{(E_1 + F, E_2 + F), (E_2 + F, E_1 + F)\})^+$ is a RY-Markovian bisimulation of order 1. Observed that $\mathcal{B}'$ is an equivalence relation, we have two cases:
  - If $(E_1 + F, E_2 + F) \in \mathcal{B}$, then $\mathcal{B}' = \mathcal{B}$ and the result trivially follows.
  - Assume that $(E_1 + F, E_2 + F) \notin \mathcal{B}$. Observed that

    $$\mathcal{G}_1/\mathcal{B}' = (\mathcal{G}_1/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}) \cup \{[E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}\}$$

    let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, $\tilde{b} \in ARew$, and $C \in \mathcal{G}_1/\mathcal{B}'$.
    * If $(F_1, F_2) \in \mathcal{B}$ and $C \in \mathcal{G}/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}$, then trivially $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.

- If $(F_1, F_2) \in \mathcal{B}$ and $C = [E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have $RY_1(F_j, a, l, \tilde{b}, C) = RY_1(F_j, a, l, \tilde{b}, [E_1 + F]_{\mathcal{B}}) + RY_1(F_j, a, l, \tilde{b}, [E_2 + F]_{\mathcal{B}})$ so $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.
- If $(F_1, F_2) \in \mathcal{B}' - \mathcal{B}$, i.e. $F_1 \in [E_1 + F]_{\mathcal{B}}$ and $F_2 \in [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

$$RY_1(F_j, a, l, \tilde{b}, C) = \begin{cases} RY_1(E_j, a, l, \tilde{b}, C) + RY_1(F, a, l, \tilde{b}, C) \\ RY_1(E_j, a, l, \tilde{b}, C) \\ RY_1(F, a, l, \tilde{b}, C) \\ \perp \end{cases}$$

depending on whether $RY_1(E_j, a, l, \tilde{b}, C) \neq \perp \wedge RY_1(F, a, l, \tilde{b}, C) \neq \perp$ or $RY_1(E_j, a, l, \tilde{b}, C) \neq \perp \wedge ((l \geq 0 \wedge RY_1(F, a, l, \tilde{b}, C) = \perp) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' \leq l \Longrightarrow RY_1(F, a, l', *, \mathcal{G}_1) = \perp))$ or $RY_1(F, a, l, \tilde{b}, C) \neq \perp \wedge ((l \geq 0 \wedge RY_1(E_j, a, l, \tilde{b}, C) = \perp) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' \leq l \Longrightarrow RY_1(E_j, a, l', *, \mathcal{G}_1) = \perp))$ or none of the previous clauses holds.

  - If $C \in \mathcal{G}_1/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}$, then from $(E_1, E_2) \in \mathcal{B}$ we derive $RY_1(E_1, a, l, \tilde{b}, C) = RY_1(E_2, a, l, \tilde{b}, C)$ so $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.
  - If $C = [E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have $RY_1(E_j, a, l, \tilde{b}, C) = RY_1(E_j, a, l, \tilde{b}, [E_1 + F]_{\mathcal{B}}) + RY_1(E_j, a, l, \tilde{b}, [E_2 + F]_{\mathcal{B}})$. Since $(E_1, E_2) \in \mathcal{B}$, it turns out that $RY_1(E_1, a, l, \tilde{b}, C) = RY_1(E_2, a, l, \tilde{b}, C)$ so $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.

- Given $F \in \mathcal{G}_1$ and $S \subseteq AType - \{\tau\}$, we prove that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_1}$, where $\mathcal{B} = \{(E_1 \parallel_S F, E_2 \parallel_S F) \mid E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2\}$ and $Id_{\mathcal{G}_1}$ is the identity relation over $\mathcal{G}_1$, is a RY-Markovian bisimulation of order 1. Observed that $\mathcal{B}'$ is an equivalence relation and that either each of the terms of an equivalence class has "$\_ \parallel_S F$" as outermost operator or none of them has, let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, $\tilde{b} \in ARew$, and $C \in \mathcal{G}_1/\mathcal{B}'$.

  - If $(F_1, F_2) \in Id_{\mathcal{G}_1}$, then trivially $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.
  - If $(F_1, F_2) \in \mathcal{B}$, then $F_1 \equiv E_1 \parallel_S F$ and $F_2 \equiv E_2 \parallel_S F$ where $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2$.
    - If none of the terms in C has "$\_ \parallel_S F$" as outermost operator, then trivially $RY_1(F_1, a, l, \tilde{b}, C) = \perp = RY_1(F_2, a, l, \tilde{b}, C)$.
    - If each of the terms in C has "$\_ \parallel_S F$" as outermost operator, given $E \parallel_S G \in C$ it turns out that $C = \{E' \parallel_S G \mid E' \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}\}$.
      - If $a \notin S$, then for $j \in \{1, 2\}$ we have that

$$RY_1(F_j, a, l, \tilde{b}, C) = \begin{cases} RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) + RY_1(F, a, l, \tilde{b}, \{G\}) \\ RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \\ RY_1(F, a, l, \tilde{b}, \{G\}) \\ \perp \end{cases}$$

depending on whether $E_j \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}} \wedge F \equiv G \wedge RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge RY_1(F, a, l, \tilde{b}, \{G\}) \neq \perp$ or $E_j \notin [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}} \wedge F \equiv G \wedge RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow RY_1(F, a, l', *, \mathcal{G}_1) = \perp))$ or $E_j \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}} \wedge F \not\equiv G \wedge RY_1(F, a, l, \tilde{b}, \{G\}) \neq \perp \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_-. l' < l \Longrightarrow RY_1(E_j, a, l', *, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) = \perp$ or none of the previous clauses holds. Since $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2$, it follows that $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$.

· If $a \in S$ then, supposed

$$\begin{aligned}
*_{-l, w_{E_j,l}} &= Rate_1(E_j, a, l, *, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \\
*_{-l, w_{E_j,l,\mathrm{tot}}} &= Rate_1(E_j, a, l, *, \mathcal{G}_1) \\
*_{-l, w_{F,l}} &= Rate_1(F, a, l, *, \{G\}) \\
*_{-l, w_{F,l,\mathrm{tot}}} &= Rate_1(F, a, l, *, \mathcal{G}_1)
\end{aligned}$$

for $l \in \mathbb{Z}_-$ and

$$N = \begin{cases} w_{E_j,l_1,\mathrm{tot}} + w_{F,l_2,\mathrm{tot}} & \text{if } l_1 = l_2 \\ w_{E_j,l_1,\mathrm{tot}} & \text{if } l_1 > l_2 \\ w_{F,l_2,\mathrm{tot}} & \text{if } l_2 > l_1 \end{cases}$$

we have that for $j \in \{1, 2\}$

$$RY_1(F_j, a, l, \tilde{b}, C) = \begin{cases} RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \cdot w_{F,l'} / w_{F,l',\mathrm{tot}} + \\ \quad RY_1(F, a, l, \tilde{b}, \{G\}) \cdot w_{E_j,l''} / w_{E_j,l'',\mathrm{tot}} \\ RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \cdot w_{F,l'} / w_{F,l',\mathrm{tot}} \\ RY_1(F, a, l, \tilde{b}, \{G\}) \cdot w_{E_j,l''} / w_{E_j,l'',\mathrm{tot}} \\ *_{-l, w_{E_j,l_1} / w_{E_j,l_1,\mathrm{tot}} \cdot w_{F,l_2} / w_{F,l_2,\mathrm{tot}} \cdot N} \\ \perp \end{cases}$$

depending on whether $l \geq 0 \wedge RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge \exists l' \in \mathbb{Z}_-. RY_1(F, a, l', *, \{G\}) \neq \perp \wedge RY_1(F, a, l, \tilde{b}, \{G\}) \neq \perp \wedge \exists l'' \in \mathbb{Z}_-. RY_1(E_j, a, l'', *, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp$ or $l \geq 0 \wedge RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge \exists l' \in \mathbb{Z}_-. RY_1(F, a, l', *, \{G\}) \neq \perp \wedge (RY_1(F, a, l, \tilde{b}, \{G\}) = \perp \vee \forall l''' \in \mathbb{Z}_-. RY_1(E_j, a, l''', *, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) = \perp)$ or $l \geq 0 \wedge RY_1(F, a, l, \tilde{b}, \{G\}) \neq \perp \wedge \exists l'' \in \mathbb{Z}_-. RY_1(E_j, a, l'', *, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge (RY_1(E_j, a, l, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) = \perp \vee \forall l''' \in \mathbb{Z}_-. RY_1(F, a, l''', *, \{G\}) = \perp)$ or $l < 0 \wedge \exists l_1, l_2 \in \mathbb{Z}_-. RY_1(E_j, a, l_1, \tilde{b}, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RY}}}}) \neq \perp \wedge RY_1(F, a, l_2, \tilde{b}, \{G\}) \neq \perp \wedge -l = \max(-l_1, -l_2)$ or none of the previous clauses holds. From $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RY}}} E_2$, it follows that $RY_1(F_1, a, l, \tilde{b}, C) = RY_1(F_2, a, l, \tilde{b}, C)$. ■

C.2 Congruence Property of $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$

In this section we provide the proof of Theorem 12.4 which says that $\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}$ is a congruence w.r.t. all the operators of $\mathrm{EMPA}_{\mathrm{gr}_1}$ and recursive constant definitions.

Proof of Theorem 12.4 The proof is similar to that of the corresponding Theorems 11.3 and 11.4 of Chapter 11 (presented in Appendix B.2) with some changes in the case of the alternative and parallel composition operators that we now show.

- Let $\mathcal{B} \subseteq \mathcal{G}_1 \times \mathcal{G}_1$ be a RYB-Markovian bisimulation of order 1 such that $(E_1, E_2) \in \mathcal{B}$. Given $F \in \mathcal{G}_1$, we prove that $\mathcal{B}' = (\mathcal{B} \cup \{(E_1 + F, E_2 + F), (E_2 + F, E_1 + F)\})^+$ is a RYB-Markovian bisimulation of order 1. Observed that $\mathcal{B}'$ is an equivalence relation, we have two cases:
  - If $(E_1 + F, E_2 + F) \in \mathcal{B}$, then $\mathcal{B}' = \mathcal{B}$ and the result trivially follows.
  - Assume that $(E_1 + F, E_2 + F) \notin \mathcal{B}$. Observed that

$$\mathcal{G}_1/\mathcal{B}' = (\mathcal{G}_1/\mathcal{B} - \{[E_1 + F]_\mathcal{B}, [E_2 + F]_\mathcal{B}\}) \cup \{[E_1 + F]_\mathcal{B} \cup [E_2 + F]_\mathcal{B}\}$$

  let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}_1/\mathcal{B}'$.

    * If $(F_1, F_2) \in \mathcal{B}$ and $C \in \mathcal{G}/\mathcal{B} - \{[E_1 + F]_\mathcal{B}, [E_2 + F]_\mathcal{B}\}$, then trivially $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.
    * If $(F_1, F_2) \in \mathcal{B}$ and $C = [E_1 + F]_\mathcal{B} \cup [E_2 + F]_\mathcal{B}$, then for $j \in \{1, 2\}$ we have:

$$\begin{array}{rcl}
Rate_1(F_j, a, l, C) & = & Rate_1(F_j, a, l, [E_1 + F]_\mathcal{B}) + Rate_1(F_j, a, l, [E_2 + F]_\mathcal{B}) \\
Yield_1(F_j, a, l, C) & = & Yield_1(F_j, a, l, [E_1 + F]_\mathcal{B}) + Yield_1(F_j, a, l, [E_2 + F]_\mathcal{B}) \\
Bonus_1(F_j, a, l, C) & = & \frac{Rate_1(F_j, a, l, [E_1+F]_\mathcal{B})}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(F_j, a, l, [E_1 + F]_\mathcal{B}) + \\
 & & \frac{Rate_1(F_j, a, l, [E_2+F]_\mathcal{B})}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(F_j, a, l, [E_2 + F]_\mathcal{B})
\end{array}$$

      so $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.
    * If $(F_1, F_2) \in \mathcal{B}' - \mathcal{B}$, i.e. $F_1 \in [E_1 + F]_\mathcal{B}$ and $F_2 \in [E_2 + F]_\mathcal{B}$, then for $j \in \{1, 2\}$ we have

$$\begin{array}{rcl}
Rate_1(F_j, a, l, C) & = & \left\{\begin{array}{l} Rate_1(E_j, a, l, C) + Rate_1(F, a, l, C) \\ Rate_1(E_j, a, l, C) \\ Rate_1(F, a, l, C) \\ \bot \end{array}\right. \\
Yield_1(F_j, a, l, C) & = & \left\{\begin{array}{l} Yield_1(E_j, a, l, C) + Yield_1(F, a, l, C) \\ Yield_1(E_j, a, l, C) \\ Yield_1(F, a, l, C) \\ \bot \end{array}\right. \\
Bonus_1(F_j, a, l, C) & = & \left\{\begin{array}{l} \frac{Rate_1(E_j, a, l, C)}{Rate_1(E_j + F, a, l, C)} \cdot Bonus_1(E_j, a, l, C) + \\ \quad \frac{Rate_1(F, a, l, C)}{Rate_1(E_j + F, a, l, C)} \cdot Bonus_1(F, a, l, C) \\ Bonus_1(E_j, a, l, C) \\ Bonus_1(F, a, l, C) \\ \bot \end{array}\right.
\end{array}$$

      depending on whether $RYB_1(E_j, a, l, C) \neq \bot \wedge RYB_1(F, a, l, C) \neq \bot$ or $RYB_1(E_j, a, l, C) \neq$

$\perp \wedge ((l \geq 0 \wedge RYB_1(F, a, l, C) = \perp) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{\perp}. l' \leq l \Longrightarrow RYB_1(F, a, l', \mathcal{G}_1) = \perp))$ or $RYB_1(F, a, l, C) \neq \perp \wedge ((l \geq 0 \wedge RYB_1(E_j, a, l, C) = \perp) \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{\perp}. l' \leq l \Longrightarrow RYB_1(E_j, a, l', \mathcal{G}_1) = \perp))$ or none of the previous clauses holds. There are two cases:

· If $C \in \mathcal{G}_1/\mathcal{B} - \{[E_1 + F]_{\mathcal{B}}, [E_2 + F]_{\mathcal{B}}\}$, then from $(E_1, E_2) \in \mathcal{B}$ we derive $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.

· If $C = [E_1 + F]_{\mathcal{B}} \cup [E_2 + F]_{\mathcal{B}}$, then for $j \in \{1, 2\}$ we have

$$\begin{array}{rcl} Rate_1(E_j, a, l, C) & = & Rate_1(E_j, a, l, [E_1 + F]_{\mathcal{B}}) + Rate_1(E_j, a, l, [E_2 + F]_{\mathcal{B}}) \\ Yield_1(E_j, a, l, C) & = & Yield_1(E_j, a, l, [E_1 + F]_{\mathcal{B}}) + Yield_1(E_j, a, l, [E_2 + F]_{\mathcal{B}}) \\ Bonus_1(E_j, a, l, C) & = & \frac{Rate_1(E_j, a, l, [E_1 + F]_{\mathcal{B}})}{Rate_1(E_j, a, l, C)} \cdot Bonus_1(E_j, a, l, [E_1 + F]_{\mathcal{B}}) + \\ & & \frac{Rate_1(E_j, a, l, [E_2 + F]_{\mathcal{B}})}{Rate_1(E_j, a, l, C)} \cdot Bonus_1(E_j, a, l, [E_2 + F]_{\mathcal{B}}) \end{array}$$

Since $(E_1, E_2) \in \mathcal{B}$, it turns out that $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.

- Given $F \in \mathcal{G}_1$ and $S \subseteq AType - \{\tau\}$, we prove that $\mathcal{B}' = \mathcal{B} \cup Id_{\mathcal{G}_1}$, where $\mathcal{B} = \{(E_1 \|_S F, E_2 \|_S F) \mid E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2\}$ and $Id_{\mathcal{G}_1}$ is the identity relation over $\mathcal{G}_1$, is a RYB-Markovian bisimulation of order 1. Observed that $\mathcal{B}'$ is an equivalence relation and that either each of the terms of an equivalence class has "$\_ \|_S F$" as outermost operator or none of them has, let $(F_1, F_2) \in \mathcal{B}'$, $a \in AType$, $l \in \mathbb{Z}$, and $C \in \mathcal{G}_1/\mathcal{B}'$.

  – If $(F_1, F_2) \in Id_{\mathcal{G}_1}$, then trivially $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.

  – If $(F_1, F_2) \in \mathcal{B}$, then $F_1 \equiv E_1 \|_S F$ and $F_2 \equiv E_2 \|_S F$ where $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$.

  ∗ If none of the terms in C has "$\_ \|_S F$" as outermost operator, then trivially $RYB_1(F_1, a, l, C) = \perp = RYB_1(F_2, a, l, C)$.

  ∗ If each of the terms in C has "$\_ \|_S F$" as outermost operator, given $E \|_S G \in C$ it turns out that $C = \{E' \|_S G \mid E' \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}\}$.

  · If $a \notin S$, then for $j \in \{1, 2\}$ we have that

$$Rate_1(F_j, a, l, C) = \begin{cases} Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) + Rate_1(F, a, l, \{G\}) \\ Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \\ Rate_1(F, a, l, \{G\}) \\ \bot \end{cases}$$

$$Yield_1(F_j, a, l, C) = \begin{cases} Yield_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) + Yield_1(F, a, l, \{G\}) \\ Yield_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \\ Yield_1(F, a, l, \{G\}) \\ \bot \end{cases}$$

$$Bonus_1(F_j, a, l, C) =$$

$$\begin{cases} \frac{Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}})}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) + \\ \quad \frac{Rate_1(F, a, l, \{G\})}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(F, a, l, \{G\}) \\ Bonus_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \\ Bonus_1(F, a, l, \{G\}) \\ \bot \end{cases}$$

depending on whether $E_j \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}} \wedge F \equiv G \wedge RYB_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge RYB_1(F, a, l, \{G\}) \neq \bot$ or $E_j \notin [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}} \wedge F \equiv G \wedge RYB_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{\perp}. l' < l \Longrightarrow RYB_1(F, a, l', \mathcal{G}_1) = \bot))$ or $E_j \in [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}} \wedge F \not\equiv G \wedge RYB_1(F, a, l, \{G\}) \neq \bot \wedge (l \geq 0 \vee (l < 0 \wedge \forall l' \in \mathbb{Z}_{\perp}. l' < l \Longrightarrow RYB_1(E_j, a, l', [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) = \bot$ or none of the previous clauses holds. Since $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$, it follows that $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$.

· If $a \in S$ then, supposed

$$\begin{aligned} *_{-l, w_{E_j, l}} &= Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \\ *_{-l, w_{E_j, l, \mathrm{tot}}} &= Rate_1(E_j, a, l, \mathcal{G}_1) \\ *_{-l, w_{F, l}} &= Rate_1(F, a, l, \{G\}) \\ *_{-l, w_{F, l, \mathrm{tot}}} &= Rate_1(F, a, l, \mathcal{G}_1) \end{aligned}$$

for $l \in \mathbb{Z}_{\perp}$ and

$$N_{l_1, l_2} = \begin{cases} w_{E_j, l_1, \mathrm{tot}} + w_{F, l_2, \mathrm{tot}} & \text{if } l_1 = l_2 \\ w_{E_j, l_1, \mathrm{tot}} & \text{if } l_1 > l_2 \\ w_{F, l_2, \mathrm{tot}} & \text{if } l_2 > l_1 \end{cases}$$

we have that for $j \in \{1, 2\}$

$$Rate_1(F_j, a, l, C) = \begin{cases} Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \cdot w_{F,l'}/w_{F,l',\mathrm{tot}} + \\ \quad Rate_1(F, a, l, \{G\}) \cdot w_{E_j,l''}/w_{E_j,l'',\mathrm{tot}} \\ Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \cdot w_{F,l'}/w_{F,l',\mathrm{tot}} \\ Rate_1(F, a, l, \{G\}) \cdot w_{E_j,l''}/w_{E_j,l'',\mathrm{tot}} \\ *_{-l, w_{E_j,l_1}/w_{E_j,l_1,\mathrm{tot}} \cdot w_{F,l_2}/w_{F,l_2,\mathrm{tot}} \cdot N_{l_1,l_2}} \\ \bot \end{cases}$$

$$Yield_1(F_j, a, l, C) = \begin{cases} Yield_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \cdot w_{F,l'}/w_{F,l',\mathrm{tot}} + \\ \quad Yield_1(F, a, l, \{G\}) \cdot w_{E_j,l''}/w_{E_j,l'',\mathrm{tot}} \\ Yield_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \cdot w_{F,l'}/w_{F,l',\mathrm{tot}} \\ Yield_1(F, a, l, \{G\}) \cdot w_{E_j,l''}/w_{E_j,l'',\mathrm{tot}} \\ * \\ \bot \end{cases}$$

$$Bonus_1(F_j, a, l, C) =$$

$$\begin{cases} \frac{Rate_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \cdot w_{F,l'}/w_{F,l',\mathrm{tot}}}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) + \\ \quad \frac{Rate_1(F, a, l, \{G\}) \cdot w_{E_j,l''}/w_{E_j,l'',\mathrm{tot}}}{Rate_1(F_j, a, l, C)} \cdot Bonus_1(F, a, l, \{G\}) \\ Bonus_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \\ Bonus_1(F, a, l, \{G\}) \\ * \\ \bot \end{cases}$$

depending on whether $l \geq 0 \wedge RYB_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge \exists l' \in \mathbb{Z}_{\bot}.\, RYB_1(F, a, l', \{G\}) \neq \bot \wedge RYB_1(F, a, l, \{G\}) \neq \bot \wedge \exists l'' \in \mathbb{Z}_{\bot}.\, RYB_1(E_j, a, l'', [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot$ or $l \geq 0 \wedge RYB_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge \exists l' \in \mathbb{Z}_{\bot}.\, RYB_1(F, a, l', \{G\}) \neq \bot \wedge (RYB_1(F, a, l, \{G\}) = \bot \vee \forall l''' \in \mathbb{Z}_{\bot}.\, RYB_1(E_j, a, l''', [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) = \bot)$ or $l \geq 0 \wedge RYB_1(F, a, l, \{G\}) \neq \bot \wedge \exists l'' \in \mathbb{Z}_{\bot}.\, RYB_1(E_j, a, l'', [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge (RYB_1(E_j, a, l, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) = \bot \vee \forall l''' \in \mathbb{Z}_{\bot}.\, RYB_1(F, a, l''', \{G\}) = \bot)$ or $l < 0 \wedge \exists l_1, l_2 \in \mathbb{Z}_{\bot}.\, RYB_1(E_j, a, l_1, [E]_{\sim_{\mathrm{MB}_1^{\mathrm{RYB}}}}) \neq \bot \wedge RYB_1(F, a, l_2, \{G\}) \neq \bot \wedge -l = \max(-l_1, -l_2)$ or none of the previous clauses holds. From $E_1 \sim_{\mathrm{MB}_1^{\mathrm{RYB}}} E_2$, it follows that $RYB_1(F_1, a, l, C) = RYB_1(F_2, a, l, C)$. ■

Appendix D

Technical Machinery for Chapter 13

In this appendix we present the proof of the two main results of Chapter 13: the congruence property of $\sim_{\rm GSMBE}$ with respect to all the operators of GSMPA and with respect to recursive definitions.

D.1 Congruence Property of $\sim_{\rm GSMBE}$ w.r.t. GSMPA Operators

In this section we provide the proof of Theorem 13.1 that says that $\sim_{\rm GSMBE}$ is a congruence w.r.t. all the operators of GSMPA. In order to do that, we introduce, after some preliminary definitions, a lemma from which it derives directly.

We extend the functions *hide*, *relab*, *left* and *right* of Chapter 13 to the consistent location associations $\psi \in \Psi$.

Definition D.1 *Given a consistent location association $\psi \in \Psi$, we define the functions hide* $: \Psi \times \mathcal{P}(AType) \longrightarrow \Psi$ *and relab* $: \Psi \times ARFun \longrightarrow \Psi$ *as follows:*

- $hide(\psi, L) = \{(a^h_{loc'}, a^h_{loc''}) \mid (a^v_{loc'}, a^v_{loc''}) \in \psi \wedge a \in L\} \cup \{(a^v_{loc'}, a^v_{loc''}) \mid (a^v_{loc'}, a^v_{loc''}) \in \psi \wedge a \notin L\}$
- $relab(\psi, \varphi) = \{(\varphi(a)^v_{loc'}, \varphi(a)^v_{loc'}) \mid (a^v_{loc'}, a^v_{loc''}) \in \psi\}$ ■

We denote with ψ_ϵ a relation over $AVId \cup \{\epsilon\}$ such that $\psi_\epsilon \cap AVId \times AVId = \psi \in \Psi$ and $dom(\psi) \cap \psi_\epsilon^{-1}(\epsilon) = range(\psi) \cap \psi_\epsilon(\epsilon) = \emptyset$, where ϵ represents an "empty" location. We denote with Ψ_ϵ the set of all relations ψ_ϵ.

Definition D.2 *Given a consistent location association $\psi \in \Psi$, we define the functions left* $: \Psi \longrightarrow \Psi_\epsilon$ *and right* $: \Psi \longrightarrow \Psi_\epsilon$ *as follows:*

- $left(\psi) = \{(left(a^v_{loc'}), left(a^v_{loc''})) \mid (a^v_{loc'}, a^v_{loc''}) \in \psi\} - \{\epsilon, \epsilon\}$

  where:

  $$left(a^v_{loc}) = \begin{cases} \epsilon & \text{if } left(\{a^v_{loc}\}) = \emptyset \\ a^v_{loc'} & \text{if } left(\{a^v_{loc}\}) = \{a^v_{loc'}\} \end{cases}$$

- $right(\psi) = \{(right(a^v_{loc'}), right(a^v_{loc''})) \mid (a^v_{loc'}, a^v_{loc''}) \in \psi\} - \{\epsilon, \epsilon\}$

  where:
  $$right(a^v_{loc}) = \begin{cases} \epsilon & \text{if } \ right(\{a^v_{loc}\}) = \emptyset \\ a^v_{loc'} & \text{if } \ right(\{a^v_{loc}\}) = \{a^v_{loc'}\} \end{cases}$$
  ■

Lemma D.1 *Let $E_1, E_2 \in \mathcal{G}$ and $Exec_1, Exec_2 \subseteq AVId$. Then:*

(i) For every $<a^v_n, w> \in LAct$,

  – $\langle E_1, \emptyset \rangle \sim_{GSMBF,\emptyset} \langle E_2, \emptyset \rangle \ \Rightarrow \ \langle <a^v_n, w>.E_1, \{a^v_n\} \rangle \sim_{GSMBF,\{(a^v_n,a^v_n)\}} \langle <a^v_n, w>.E_2, \{a^v_n\} \rangle$

  – $\langle <a^v_n, w>.E_1, \{a^v_n\} \rangle \sim_{GSMBF,\{(a^v_n,a^v_n)\}} \langle <a^v_n, w>.E_2, \{a^v_n\} \rangle \ \Rightarrow$
  $$\langle <a^v_n, w>.E_1, \emptyset \rangle \sim_{GSMBF,\emptyset} \langle <a^v_n, w>.E_2, \emptyset \rangle$$

(ii) For every $L \subseteq AType$,
  $\langle E_1, Exec_1 \rangle \sim_{GSMBF,\psi} \langle E_2, Exec_2 \rangle \ \Rightarrow$
  $$\langle E_1/L, hide(Exec_1, L) \rangle \sim_{GSMBF,hide(\psi,L)} \langle E_2/L, hide(Exec_2, L) \rangle$$

(iii) For every $\varphi \in ARFun$,
  $\langle E_1, Exec_1 \rangle \sim_{GSMBF,\psi} \langle E_2, Exec_2 \rangle \ \Rightarrow$
  $$\langle E_1[\varphi], relab(Exec_1, \varphi) \rangle \sim_{GSMBF,relab(\psi,\varphi)} \langle E_2[\varphi], relab(Exec_2, \varphi) \rangle$$

(iv) For every $F \in \mathcal{G}$,
  $\langle E_1, \emptyset \rangle \sim_{GSMBF,\emptyset} \langle E_2, \emptyset \rangle \ \Rightarrow \ \langle E_1 + F, \emptyset \rangle \sim_{GSMBF,\emptyset} \langle E_2 + F, \emptyset \rangle$

(v) For every $F \in \mathcal{G}$ and $S \subseteq AType$,
  $\langle E_1, left(Exec_1) \rangle \sim_{GSMBF,left(\psi)} \langle E_2, left(Exec_2) \rangle \ \Rightarrow \ \langle E_1 \|_S F, Exec_1 \rangle \sim_{GSMBF,\psi} \langle E_2 \|_S F, Exec_2 \rangle$

Proof: The proofs of the 5 items are the following.

(i) **Proof of the first part.** Let $E_1, E_2 \in \mathcal{G}$ be such that $\langle E_1, \emptyset \rangle \sim_{GSMBF,\emptyset} \langle E_2, \emptyset \rangle$.

Let $\mathbf{B}$ be a strong GSMBF such that $(\langle E_1, \emptyset \rangle, \langle E_2, \emptyset \rangle) \in \mathcal{B}_\emptyset$. Given $<a^v_n, w> \in LAct$, we prove that $\mathbf{B}' = \{\mathcal{B}'_\psi \mid \psi \in \Psi\}$ where

  – $\mathcal{B}'_{\{(a^v_n,a^v_n)\}} = (\mathcal{B}_{\{(a^v_n,a^v_n)\}} \cup \{(\langle <a^v_n, w>.E_1, \{a^v_n\} \rangle, \langle <a^v_n, w>.E_2, \{a^v_n\} \rangle),$
  $(\langle <a^v_n, w>.E_2, \{a^v_n\} \rangle, \langle <a^v_n, w>.E_1, \{a^v_n\} \rangle)\})^+$

  – $\mathcal{B}'_\psi = \mathcal{B}_\psi \qquad \forall\, \psi \neq \{(a^v_n, a^v_n)\}$

is a strong GSMBF. Once easily observed that $\mathbf{B}'$ satisfies the first three conditions of the definition of strong GSMBF, we have two cases.

– If $(\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle, \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle) \in \mathcal{B}_{\{(a^v_n, a^v_n)\}}$, then $\mathbf{B}' = \mathbf{B}$ and the result trivially follows.

– Assume that $(\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle, \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle) \notin \mathcal{B}_{\{(a^v_n, a^v_n)\}}$ and let $(s_1, s_2) \in \mathcal{B}'_\psi$. For each $(C_1, C_2) \in \mathit{strel}(\mathbf{B}', \psi)$ we have the following cases.

  * If $(s_1, s_2) \in \mathcal{B}_\psi$ and either $\psi \neq \emptyset$, or $\psi = \emptyset$ and $(C_1, C_2) \in \mathit{strel}(\mathbf{B}, \emptyset) - \{([\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)}, [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)}), ([\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)}, [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)})\}$, then trivially $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$, because $\mathbf{B}$ is a GSMBF.
  * If $(s_1, s_2) \in \mathcal{B}_\emptyset$ and $C_1 = C_2 = [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)} \cup [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)}$, then for $j \in \{1, 2\}$ we have:

    $$\begin{aligned}\mathit{tweight}(s_j, C_j) = {} & \mathit{tweight}(s_j, [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)}) + \\ & \mathit{tweight}(s_j, [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B},\emptyset)})\end{aligned}$$

    so: $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$, because $\mathbf{B}$ is a GSMBF.
  * If $(s_1, s_2) \in \mathcal{B}'_{\{(a^v_n, a^v_n)\}} - \mathcal{B}_{\{(a^v_n, a^v_n)\}}$, i.e. $s_1 \in [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{(a^v_n, a^v_n)\}}}$ and $s_2 \in [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{(a^v_n, a^v_n)\}}}$, then for $j \in \{1, 2\}$ we have trivially $\mathit{tweight}(s_j, C_j) = 0$.

  For each $(C_1, C_2) \in \mathit{terel}(\mathbf{B}', \psi)$ we have the following cases.

  * If $(s_1, s_2) \in \mathcal{B}_\psi$ and either $\{a^v_n, a^v_n\} \not\subset \psi$, or $\{a^v_n, a^v_n\} \subset \psi$ and $(C_1, C_2) \in \mathit{terel}(\mathbf{B}, \psi) - \{([\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}}, [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}}), ([\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}}, [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}})\}$, then trivially $\mathit{ter}(s_1, C_1) = \mathit{ter}(s_2, C_2)$, because $\mathbf{B}$ is a GSMBF.
  * If $(s_1, s_2) \in \mathcal{B}_\psi$ with $\{a^v_n, a^v_n\} \subset \psi$ and $C_1 = C_2 = [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}} \cup [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}}$, then for $j \in \{1, 2\}$ we have:

    $$\mathit{ter}(s_j, C_j) = \mathit{ter}(s_j, [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}}) \vee \mathit{ter}(s_j, [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{a^v_n, a^v_n\}}})$$

    so: $\mathit{ter}(s_1, C_1) = \mathit{ter}(s_2, C_2)$, because $\mathbf{B}$ is a GSMBF.
  * If $(s_1, s_2) \in \mathcal{B}'_{\{(a^v_n, a^v_n)\}} - \mathcal{B}_{\{(a^v_n, a^v_n)\}}$, i.e. $s_1 \in [\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle]_{\mathcal{B}_{\{(a^v_n, a^v_n)\}}}$ and $s_2 \in [\langle <a^v_n, w>.E_2, \{a^v_n\}\rangle]_{\mathcal{B}_{\{(a^v_n, a^v_n)\}}}$, then for $j \in \{1, 2\}$ we have:

    $$\mathit{ter}(s_j, C_j) = \begin{cases} \mathit{True} & \text{if } C_j = [\langle E_j, \emptyset\rangle]_{\mathcal{B}'_\emptyset} \\ \mathit{False} & \text{otherwise} \end{cases}$$

    Since $(\langle E_1, \emptyset\rangle, \langle E_2, \emptyset\rangle) \in \mathcal{B}_\emptyset$ implies $([\langle E_1, \emptyset\rangle]_{\mathcal{B}'_\emptyset}, [\langle E_2, \emptyset\rangle]_{\mathcal{B}'_\emptyset}) \in \mathit{terel}(\mathbf{B}', \{(a^v_n, a^v_n)\})$, it turns out that $\mathit{ter}(s_1, C_1) = \mathit{ter}(s_2, C_2)$.

Proof of the second part. Let $E_1, E_2 \in \mathcal{G}$ and $<a^v_n, w> \in \mathit{LAct}$ be such that $\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle \sim_{GSMBF, \{(a^v_n, a^v_n)\}} \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle$.

Let $\mathbf{B}$ be a strong GSMBF such that $(\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle, \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle) \in \mathcal{B}_{\{(a^v_n, a^v_n)\}}$. The proof that $\mathbf{B}' = \{\mathcal{B}'_\psi \mid \psi \in \Psi\}$ where

– $\mathcal{B}'_{\emptyset} = (\mathcal{B}_{\emptyset} \cup \{(\langle <a^v_n, w>.E_1, \emptyset\rangle, \langle <a^v_n, w>.E_2, \emptyset\rangle), (\langle <a^v_n, w>.E_2, \emptyset\rangle, \langle <a^v_n, w>.E_1, \emptyset\rangle)\})^+$

– $\mathcal{B}'_{\psi} = \mathcal{B}_{\psi} \qquad \forall \psi \neq \emptyset$

is a strong GSMBF is similar to the one just presented. The main differences are the following ones. Let $(s_1, s_2) \in \mathcal{B}'_{\psi}$. Considered $(C_1, C_2) \in \mathit{strel}(\mathbf{B}', \psi)$ in the last subcase (where $(s_1, s_2) \in \mathcal{B}'_{\emptyset} - \mathcal{B}_{\emptyset}$) the result follows from the fact that for $j \in \{1, 2\}$ we have:

$$\mathit{tweight}(s_j, C_j) = \begin{cases} w & \text{if } C_j = [\langle <a^v_n, w>.E_j, \{a^v_n\}\rangle]_{\mathit{stmap}(\mathbf{B}', \emptyset)} \\ 0 & \text{otherwise} \end{cases}$$

Since $(\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle, \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle) \in \mathcal{B}_{\{(a^v_n, a^v_n)\}}$ implies $(\langle <a^v_n, w>.E_1, \{a^v_n\}\rangle, \langle <a^v_n, w>.E_2, \{a^v_n\}\rangle) \in \mathit{stmap}(\mathbf{B}', \emptyset)$, it turns out that $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$.

Considered $(C_1, C_2) \in \mathit{terel}(\mathbf{B}', \psi)$ in the last subcase (where $(s_1, s_2) \in \mathcal{B}'_{\emptyset} - \mathcal{B}_{\emptyset}$) the result follows from the fact that for $j \in \{1, 2\}$ we have trivially $\mathit{ter}(s_j, C_j) = \mathit{False}$.

(ii) Given $L \subseteq \mathit{AType}$, we consider the family of relations:

$$\mathcal{B}_{\psi} = \{(\langle E_1/L, \mathit{hide}(\mathit{Exec}_1, L)\rangle, \langle E_2/L, \mathit{hide}(\mathit{Exec}_2, L)\rangle) \mid \exists \psi'.\, \psi = \mathit{hide}(\psi', L) \wedge \langle E_1, \mathit{Exec}_1\rangle \sim_{GSMBF, \psi'} \langle E_2, \mathit{Exec}_2\rangle\}$$

We prove that $\mathbf{B}' = \{\mathcal{B}'_{\psi} \mid \psi \in \Psi\}$ where

– $\mathcal{B}'_{I_{Exec}} = \mathcal{B}_{I_{Exec}} \cup I_{\{\langle E, Exec\rangle \in LabS \mid E \in \mathcal{G}\}} \qquad \forall\, \mathit{Exec} \subseteq \mathit{AVId}$

– $\mathcal{B}'_{\psi} = \mathcal{B}_{\psi} \qquad \forall\, \psi \neq I_{Exec}$

is a strong GSMBF. Once easily observed that $\mathbf{B}'$ satisfies the first three conditions of the definition of strong GSMBF, let $(s_1, s_2) \in \mathcal{B}'_{\psi}$. For each $(C_1, C_2) \in \mathit{strel}(\mathbf{B}', \psi)$ we have the following cases.

– If $\psi = I_{Exec}$ and $(s_1, s_2) \in I_{\{\langle E, Exec\rangle \in LabS \mid E \in \mathcal{G}\}}$, we have $C_1 = C_2$ because $\mathit{strel}(\mathbf{B}', I_{Exec})$ is an equivalence relation, hence trivially: $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$.

– If $(s_1, s_2) \in \mathcal{B}_{\psi}$, then $s_1 \equiv \langle E_1/L, \mathit{hide}(\mathit{Exec}_1, L)\rangle$ and $s_2 \equiv \langle E_2/L, \mathit{hide}(\mathit{Exec}_2, L)\rangle$ where $\langle E_1, \mathit{Exec}_1\rangle \sim_{GSMBF, \psi'} \langle E_2, \mathit{Exec}_2\rangle$ with $\psi = \mathit{hide}(\psi', L)$. We have two cases.

  * None of the states of C_1 and C_2 has "$\_/L$" as outermost operator, then trivially: $\mathit{tweight}(s_1, C_1) = 0 = \mathit{tweight}(s_2, C_2)$.
  * Each of the states of C_1 and C_2 has "$\_/L$" as outermost operator. For $j \in \{1, 2\}$ we define $C'_j = \{\langle E'_j, \mathit{Exec}'_j\rangle \mid \langle E'_j/L, \mathit{hide}(\mathit{Exec}'_j, L)\rangle \in C_j\}$. From the definition of $\mathbf{B}'$ we derive that there exists $\psi'' \in \Psi$ with $\psi = \mathit{hide}(\psi'', L)$ such that, defined $\mathit{Exec}''_1 = \mathit{dom}(\psi'')$ and $\mathit{Exec}''_2 = \mathit{range}(\psi'')$, for $j \in \{1, 2\}$ it turns out:

    $C_j = \{\langle E''_j/L, \mathit{hide}(\mathit{Exec}''_j \cup \{a^v_{loc}\}, L)\rangle \mid \langle E''_j, \mathit{Exec}''_j \cup \{a^v_{loc}\}\rangle \in C'_j\}$.

    Thus for $j \in \{1, 2\}$ we have

$$tweight(s_j, C_j) = \begin{cases} tweight(\langle E_j, Exec_j \rangle, C'_j) & \text{if } \psi'' = \psi' \\ 0 & \text{otherwise} \end{cases}$$

In the case of $\psi'' = \psi'$ the result derives from the following considerations.

- If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in strel($ $\sim_{\mathbf{GSMBF}}, \psi')$. From $\langle E_1, Exec_1 \rangle \sim_{GSMBF,\psi'} \langle E_2, Exec_2 \rangle$ it follows that $tweight(s_1, C_1)$ $= tweight(s_2, C_2)$.
- If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in strel(\sim_{\mathbf{GSMBF}}, \psi')$. Therefore $tweight(s_2, C'_2) = 0$. It follows $tweight(s_1, C_1) = 0 = tweight(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

For each $(C_1, C_2) \in terel(\mathbf{B}', \psi)$ we have the following cases.

– If $\psi = I_{Exec}$ and $(s_1, s_2) \in I_{\{\langle E, Exec \rangle \in LabS \mid E \in \mathcal{G}\}}$, we have $C_1 = C_2$ because $terel(\mathbf{B}', I_{Exec})$ is an equivalence relation, hence trivially: $ter(s_1, C_1) = ter(s_2, C_2)$.

– If $(s_1, s_2) \in \mathcal{B}_\psi$, then $s_1 \equiv \langle E_1/L, hide(Exec_1, L) \rangle$ and $s_2 \equiv \langle E_2/L, hide(Exec_2, L) \rangle$ where $\langle E_1,$ $Exec_1 \rangle \sim_{GSMBF,\psi'} \langle E_2, Exec_2 \rangle$ with $\psi = hide(\psi', L)$. We have two cases.

  * None of the states of C_1 and C_2 has "$\_/L$" as outermost operator, then trivially: $ter(s_1, C_1) =$ $False = ter(s_2, C_2)$.
  * Each of the states of C_1 and C_2 has "$\_/L$" as outermost operator. For $j \in \{1, 2\}$ we define $C'_j = \{\langle E'_j, Exec'_j \rangle \mid \langle E'_j/L, hide(Exec'_j, L) \rangle \in C_j\}$. From the definition of $\mathbf{B}'$ we derive that there exists $nter \subset dom(\psi)$ and $\psi'' \in \Psi$ with $\psi|_{nter} = hide(\psi'', L)$ such that, defined $nter_1 = dom(\psi'')$ and $nter_2 = range(\psi'')$, for $j \in \{1, 2\}$ it turns out:

    $C_j = \{\langle E''_j/L, hide(nter_j, L) \rangle \mid \langle E''_j, nter_j \rangle \in C'_j\}$.

    Thus for $j \in \{1, 2\}$ we have

    $$ter(s_j, C_j) = \begin{cases} ter(\langle E_j, Exec_j \rangle, C'_j) & \text{if } \psi'' \subset \psi' \\ 0 & \text{otherwise} \end{cases}$$

    In the case of $\psi'' \subset \psi'$ the result derives from the following considerations.

    - If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in terel($ $\sim_{\mathbf{GSMBF}}, \psi')$. From $\langle E_1, Exec_2 \rangle \sim_{GSMBF,\psi'} \langle E_2, Exec_2 \rangle$ it follows that $ter(s_1, C_1) =$ $ter(s_2, C_2)$.
    - If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in terel(\sim_{\mathbf{GSMBF}}, \psi')$. Therefore $ter(s_2, C'_2) = False$. It follows $ter(s_1, C_1) = False = ter(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

(iii) Given $\varphi \in ARFun$, we consider the family of relations:

$$\mathcal{B}_{\psi} = \{(\langle E_1[\varphi], relab(Exec_1, \varphi)\rangle, \langle E_2[\varphi], relab(Exec_2, \varphi)\rangle) \mid \exists \psi'.\, \psi = relab(\psi', \varphi) \wedge \langle E_1, Exec_1\rangle \sim_{GSMBF,\psi'} \langle E_2, Exec_2\rangle\}$$

The proof that $\mathbf{B}' = \{\mathcal{B}'_{\psi} \mid \psi \in \Psi\}$ where

- $\mathcal{B}'_{I_{Exec}} = \mathcal{B}_{I_{Exec}} \cup I_{\{\langle E, Exec\rangle \in LabS \mid E \in \mathcal{G}\}} \qquad \forall\, Exec \subseteq AVId$
- $\mathcal{B}'_{\psi} = \mathcal{B}_{\psi} \qquad \forall\, \psi \neq I_{Exec}$

is a strong GSMBF is completely analogous to the one developed in (ii).

(iv) Let $E_1, E_2 \in \mathcal{G}$ be such that $\langle E_1, \emptyset\rangle \sim_{GSMBF,\emptyset} \langle E_2, \emptyset\rangle$

Let $\mathbf{B}$ be a strong GSMBF such that $(\langle E_1, \emptyset\rangle, \langle E_2, \emptyset\rangle) \in \mathcal{B}_{\emptyset}$. Given $F \in \mathcal{G}$, the proof that $\mathbf{B}' = \{\mathcal{B}'_{\psi} \mid \psi \in \Psi\}$ where

- $\mathcal{B}'_{\emptyset} = (\mathcal{B}_{\emptyset} \cup \{(\langle E_1 + F, \emptyset\rangle, \langle E_2 + F, \emptyset\rangle), (\langle E_2 + F, \emptyset\rangle, \langle E_1 + F, \emptyset\rangle)\})^{+}$
- $\mathcal{B}'_{\psi} = \mathcal{B}_{\psi} \qquad \forall\, \psi \neq \emptyset$

is a strong GSMBF is similar to the one developed in the first part of (i). The main differences are the following ones. Let $(s_1, s_2) \in \mathcal{B}'_{\psi}$. Considered $(C_1, C_2) \in strel(\mathbf{B}', \emptyset)$ in the last subcase (where $(s_1, s_2) \in \mathcal{B}'_{\emptyset} - \mathcal{B}_{\emptyset}$) the result follows from the fact that for $j \in \{1, 2\}$ we have:

$tweight(s_j, C_j) = tweight(\langle E_j, \emptyset\rangle, C_j) + tweight(\langle F, \emptyset\rangle, C_j)$

Since $(C_1, C_2) \in strel(\mathbf{B}, \emptyset)$ then from $(\langle E_1, \emptyset\rangle, \langle E_2, \emptyset\rangle) \in \mathcal{B}_{\emptyset}$ we derive $tweight(\langle E_1, \emptyset\rangle, C_1) = tweight(\langle E_2, \emptyset\rangle, C_2)$. Moreover since $stmap(\mathbf{B}, \emptyset)$ is an equivalence relation we have $C_1 = C_2$, hence $tweight(\langle F, \emptyset\rangle, C_1) = tweight(\langle F, \emptyset\rangle, C_2)$. It turns out $tweight(s_1, C_1) = tweight(s_2, C_2)$.

Considered $(C_1, C_2) \in terel(\mathbf{B}', \psi)$ in the last subcase (where $(s_1, s_2) \in \mathcal{B}'_{\emptyset} - \mathcal{B}_{\emptyset}$) the result follows from the fact that for $j \in \{1, 2\}$ we have trivially $ter(s_j, C_j) = False$.

(v) Given $S \subseteq AType$, we consider the family of relations:

$$\mathcal{B}_{\psi} = \{(\langle E_1 \|_S F, Exec_1\rangle, \langle E_2 \|_S F, Exec_2\rangle) \mid Exec_1 = dom(\psi) \wedge Exec_2 = range(\psi) \wedge I_{right(Exec_1)} = right(\psi) \wedge F \in \mathcal{G} \wedge \langle E_1, left(Exec_1)\rangle \sim_{GSMBF,left(\psi)} \langle E_2, left(Exec_2)\rangle\}$$

The proof that $\mathbf{B}' = \{\mathcal{B}'_{\psi} \mid \psi \in \Psi\}$ where

- $\mathcal{B}'_{I_{Exec}} = \mathcal{B}_{I_{Exec}} \cup I_{\{\langle E, Exec\rangle \in LabS \mid E \in \mathcal{G}\}} \qquad \forall\, Exec \subseteq AVId$
- $\mathcal{B}'_{\psi} = \mathcal{B}_{\psi} \qquad \forall\, \psi \neq I_{Exec}$

is a strong GSMBF is similar to the one developed in (ii). The main differences are the following ones. Let $(s_1, s_2) \in \mathcal{B}_{\psi}$, then $s_1 \equiv \langle E_1 \|_S F, Exec_1\rangle$ and $s_2 \equiv \langle E_2 \|_S F, Exec_2\rangle$ where $\langle E_1, left(Exec_1)\rangle \sim_{GSMBF,left(\psi)} \langle E_2, left(Exec_2)\rangle$ with $right(Exec_1) = right(Exec_2)$. Considered $(C_1, C_2) \in strel(\mathbf{B}', \psi)$

we prove the result in the subcase where each of the states of C_1 and C_2 has "$\_ \|_S \_$" as outermost operator. For $j \in \{1,2\}$ we define $C'_j = \{\langle E'_j, \mathit{left}(\mathit{Exec'}_j)\rangle \mid \langle E'_j \|_S G, \mathit{Exec'}_j\rangle\}$. From the definition of $\mathbf{B}'$ and of $\mathit{strel}(\mathbf{B}', \psi)$, we derive that there are three cases for the structure of the sets C_1 and C_2.

- There exists $G \in \mathcal{G}$, $a \in \mathit{AType}$ and $v \in \mathit{AVis}$ with $a \notin S$ or $v = h$ such that for $j \in \{1,2\}$ it turns out:

  $C_j = \{\langle E'_j \|_S G, \mathit{Exec}_j \cup \{a^v_{\swarrow loc}\}\rangle \mid \langle E'_j, \mathit{left}(\mathit{Exec}_j) \cup \{a^v_{loc}\}\rangle \in C'_j\}$

  Thus for $j \in \{1,2\}$ we have

  $$\mathit{tweight}(s_j, C_j) = \begin{cases} \mathit{tweight}(\langle E_j, \mathit{left}(\mathit{Exec}_j)\rangle, C'_j) & \text{if } F \equiv G \\ 0 & \text{otherwise} \end{cases}$$

  In the case of $F \equiv G$ the result derives from the following considerations.

  * If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in \mathit{strel}(\sim_{\mathbf{GSMBF}}, \mathit{left}(\psi))$. From $\langle E_1, \mathit{left}(\mathit{Exec}_1)\rangle \sim_{GSMBF,\mathit{left}(\psi)} \langle E_2, \mathit{left}(\mathit{Exec}_2)\rangle$ it follows that $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$.
  * If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in \mathit{strel}(\sim_{\mathbf{GSMBF}}, \mathit{left}(\psi))$. Therefore $\mathit{tweight}(s_2, C'_2) = 0$. It follows $\mathit{tweight}(s_1, C_1) = 0 = \mathit{tweight}(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

- There exists $G \in \mathcal{G}$, $loc \in \mathit{Loc}$, $a \in \mathit{AType}$ and $v \in \mathit{AVis}$ with $a \notin S$ or $v = h$ such that for $j \in \{1,2\}$ it turns out:

  $C_j = \{\langle E'_j \|_S G, \mathit{Exec}_j \cup \{a^v_{\searrow loc}\}\rangle \mid \langle E'_j, \mathit{left}(\mathit{Exec}_j)\rangle \in C'_j\}$

  Thus for $j \in \{1,2\}$ we have

  $$\mathit{tweight}(s_j, C_j) = \begin{cases} \mathit{tweight}(\langle F, \mathit{right}(\mathit{Exec}_j)\rangle, \{\langle G, \mathit{right}(\mathit{Exec}_j) \cup \{a^v_{loc}\}\rangle\}) & \text{if } \langle E_j, \mathit{left}(\mathit{Exec}_j)\rangle \in C'_j \\ 0 & \text{otherwise} \end{cases}$$

  The result derives from the following considerations.

  * If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in \mathit{crel}(\sim_{GSMBF,\mathit{left}(\psi)})$. From $\mathit{right}(\mathit{Exec}_1) = \mathit{right}(\mathit{Exec}_2)$ and $\langle E_1, \mathit{left}(\mathit{Exec}_1)\rangle \sim_{GSMBF,\mathit{left}(\psi)} \langle E_2, \mathit{left}(\mathit{Exec}_2)\rangle$ it follows that $\mathit{tweight}(s_1, C_1) = \mathit{tweight}(s_2, C_2)$.
  * If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in \mathit{crel}(\sim_{GSMBF,\mathit{left}(\psi)})$. Since $\langle E_1, \mathit{left}(\mathit{Exec}_1)\rangle \sim_{GSMBF,\mathit{left}(\psi)} \langle E_2, \mathit{left}(\mathit{Exec}_2)\rangle$ then $\langle E_2, \mathit{left}(\mathit{Exec}_2)\rangle \notin C''_2$. It follows $\mathit{tweight}(s_1, C_1) = 0 = \mathit{tweight}(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

- There exists $G \in \mathcal{G}$, $loc \in \mathit{Loc}$ and $a \in \mathit{AType}$ with $a \in S$ such that for $j \in \{1,2\}$ it turns out:

  $C_j = \{\langle E'_j \|_S G, \mathit{Exec}_j \cup \{a^o_{\langle loc' | loc\rangle}\}\rangle \mid \langle E'_j, \mathit{left}(\mathit{Exec}_j) \cup \{a^o_{loc'}\}\rangle \in C'_j\}$

Thus for $j \in \{1,2\}$ we have

$$tweight(s_j, C_j) = tweight(\langle E_j, left(Exec_j)\rangle, C'_j) \cdot$$
$$tweight(\langle F, right(Exec_j)\rangle, \{\langle G, right(Exec_j) \cup \{a^o_{loc}\}\rangle\})$$

Since $right(Exec_1) = right(Exec_2)$ it is sufficient to prove that $tweight(\langle E_1, left(Exec_1)\rangle, C'_1) = tweight(\langle E_2, left(Exec_2)\rangle, C'_2)$ by following the same argument as the first subcase. It follows $tweight(s_1, C_1) = tweight(s_2, C_2)$.

Considered $(C_1, C_2) \in terel(\mathbf{B}', \psi)$ we prove the result in the subcase where each of the states of C_1 and C_2 has "$\_ \|_S \_$" as outermost operator. For $j \in \{1,2\}$ we define $C'_j = \{\langle E'_j, left(Exec'_j)\rangle \mid \langle E'_j \|_S G, Exec'_j\rangle\}$. From the definition of $\mathbf{B}'$ and of $terel(\mathbf{B}', \psi)$, we derive that there are three cases for the structure of the sets C_1 and C_2.

- There exists $G \in \mathcal{G}$, ter_1 nonempty subset of $dom(\psi)$ with $right(ter_1) = \emptyset$ such that, defined $ter_2 = \psi(ter_1)$ ($right(ter_1) = \emptyset$ implies $right(ter_2) = \emptyset$ since $right(\psi) = I_{right(Exec_1)}$), for $j \in \{1,2\}$ it turns out:

  $C_j = \{\langle E'_j \|_S G, Exec_j - ter_j\rangle \mid \langle E'_j, left(Exec_j) - left(ter_j)\rangle \in C'_j\}$

  Thus for $j \in \{1,2\}$ we have

  $$ter(s_j, C_j) = \begin{cases} ter(\langle E_j, left(Exec_j)\rangle, C'_j) & \text{if } F \equiv G \\ 0 & \text{otherwise} \end{cases}$$

  In the case of $F \equiv G$ the result derives from the following considerations.

  * If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in terel(\sim_{\mathbf{GSMBF}}, left(\psi))$. From $\langle E_1, left(Exec_1)\rangle \sim_{GSMBF, left(\psi)} \langle E_2, left(Exec_2)\rangle$ it follows that $ter(s_1, C_1) = ter(s_2, C_2)$.
  * If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in terel(\sim_{\mathbf{GSMBF}}, left(\psi))$. Therefore $ter(s_2, C'_2) = False$. It follows $ter(s_1, C_1) = False = ter(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

- There exists $G \in \mathcal{G}$, ter_1 nonempty subset of $dom(\psi)$ with $left(ter_1) = \emptyset$ such that, defined $ter_2 = \psi(ter_1)$ ($left(ter_1) = \emptyset$ implies $left(ter_2) = \emptyset$ since $right(\psi) = I_{right(Exec_1)}$), for $j \in \{1,2\}$ it turns out:

  $C_j = \{\langle E'_j \|_S G, Exec_j - ter_j\rangle \mid \langle E'_j, left(Exec_j)\rangle \in C'_j\}$

  Thus for $j \in \{1,2\}$ we have

  $$ter(s_j, C_j) = \begin{cases} ter(\langle F, right(Exec_j)\rangle, \{\langle G, right(Exec_j) - right(ter_j)\rangle\}) & \text{if } \langle E_j, left(Exec_j)\rangle \in C'_j \\ 0 & \text{otherwise} \end{cases}$$

  The result derives from the following considerations.

* If $C'_1 \neq \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that $(C'_1, C'_2) \in crel(\sim_{GSMBF,left(\psi)})$. From $right(Exec_1) = right(Exec_2)$, $right(ter_1) = right(ter_2)$ and $\langle E_1, left(Exec_1)\rangle \sim_{GSMBF,left(\psi)} \langle E_2, left(Exec_2)\rangle$ it follows that $ter(s_1, C_1) = ter(s_2, C_2)$.
* If $C'_1 = \emptyset$ and $C'_2 \neq \emptyset$ then from the definition of $\mathbf{B}'$ we derive that there exists C''_2 such that $C'_2 \subseteq C''_2$ and $(\emptyset, C''_2) \in crel(\sim_{GSMBF,left(\psi)})$. Since $\langle E_1, left(Exec_1)\rangle \sim_{GSMBF,left(\psi)} \langle E_2, left(Exec_2)\rangle$ then $\langle E_2, left(Exec_2)\rangle \notin C''_2$. It follows $ter(s_1, C_1) = False = ter(s_2, C_2)$. For $C'_1 \neq \emptyset$ and $C'_2 = \emptyset$ we follow a symmetric argument.

– There exists $G \in \mathcal{G}$, ter_1 nonempty subset of $dom(\psi)$ with $left(ter_1) \neq \emptyset$ and $right(ter_1) \neq \emptyset$ such that, defined $ter_2 = \psi(ter_1)$, for $j \in \{1,2\}$ it turns out:

$C_j = \{\langle E'_j \,\|_S\, G, Exec_j - ter_j\rangle \mid \langle E'_j, left(Exec_j) - left(ter_j)\rangle \in C'_j\}$

Thus for $j \in \{1,2\}$ we have

$ter(s_j, C_j) = ter(\langle E_j, left(Exec_j)\rangle, C'_j) \wedge ter(\langle F, right(Exec_j)\rangle, \{\langle G, right(Exec_j) - right(ter_j)\rangle\})$

Since $right(Exec_1) = right(Exec_2)$ and $right(ter_1) = right(ter_2)$ it is sufficient to prove that $ter(\langle E_1, left(Exec_1)\rangle, C'_1) = ter(\langle E_2, left(Exec_2)\rangle, C'_2)$ by following the same argument as the first subcase. It follows $ter(s_1, C_1) = ter(s_2, C_2)$.

□

We are now in a position to prove Theorem 13.1.

Proof of Theorem 13.1 It derives directly from the definition of $\sim_{\mathrm{GSMBE}}$ and from Lemma D.1. ■

D.2 Congruence Property of $\sim_{\mathrm{GSMBE}}$ w.r.t. Recursive Definitions

In this section we provide the proof of Theorem 13.2 that says that $\sim_{\mathrm{GSMBE}}$ is a congruence w.r.t. recursive definitions. To do that, we have to preliminarily extend the definition of $\sim_{\mathrm{GSMBE}}$ to terms that are guardedly closed up to constants devoid of defining equation.

Definition D.3 *A constant $A \in Const$ is free if and only if, for no $E \in \mathcal{L}$, $A \stackrel{\Delta}{=} E$.* ■

Let us denote by st the relation subterm-of, and by $Subst(E)$ the set of terms obtained from E by repeatedly replacing constants by the right-hand side terms of their defining equations.

Definition D.4 *The set of constants occurring in $E \in \mathcal{L}$ is defined by*

$$Const(E) = \{A \in Const \mid \exists F \in Subst(E).\, A\, st\, F\}$$

■

Definition D.5 *A term $E \in \mathcal{L}$ is partially guardedly closed (pgc) if and only if for each constant $A \in Const(E)$ either A is free or*

- *A is equipped with exactly one defining equation $A \stackrel{\Delta}{=} E'$, and*
- *there exists $F \in Subst(E')$ such that, whenever an instance of a nonfree constant B satisfies $B\, st\, F$, then the same instance satisfies $B\, st <a^v_n, w>.G\, st\, F$.* ■

Definition D.6 *Let $E \in \mathcal{L}$, $A \in Const$ free, and $B \in \mathcal{G}$. The term $E\langle\!\langle A := B\rangle\!\rangle$ obtained from E by replacing each occurrence of A with B is defined by induction on the syntactical structure of E as follows:*

- $\underline{0}\langle\!\langle A := B\rangle\!\rangle \equiv \underline{0}$
- $(<a^v_n, w>.E)\langle\!\langle A := B\rangle\!\rangle \equiv <a^v_n, w>.(E\langle\!\langle A := B\rangle\!\rangle)$
- $(E/L)\langle\!\langle A := B\rangle\!\rangle \equiv (E\langle\!\langle A := B\rangle\!\rangle)/L$
- $(E[\varphi])\langle\!\langle A := B\rangle\!\rangle \equiv (E\langle\!\langle A := B\rangle\!\rangle)[\varphi]$
- $(E_1 + E_2)\langle\!\langle A := B\rangle\!\rangle \equiv (E_1\langle\!\langle A := B\rangle\!\rangle) + (E_2\langle\!\langle A := B\rangle\!\rangle)$
- $(E_1 \,\|_S\, E_2)\langle\!\langle A := B\rangle\!\rangle \equiv (E_1\langle\!\langle A := B\rangle\!\rangle) \,\|_S (E_2\langle\!\langle A := B\rangle\!\rangle)$
- $A'\langle\!\langle A := B\rangle\!\rangle \equiv \begin{cases} B & \text{if } A' \equiv A \\ A' & \text{if } A' \not\equiv A \wedge A' \text{ free} \\ A'' & \text{if } A' \not\equiv A \wedge A' \stackrel{\Delta}{=} E \wedge A'' \stackrel{\Delta}{=} E\langle\!\langle A := B\rangle\!\rangle \end{cases}$ ■

Definition D.7 *Let $E_1, E_2 \in \mathcal{L}$ be pgc, and suppose that $Const(E_1) \cup Const(E_2)$ contains $\{A_i \in Const \mid i \in I\}$ as free constants. We say that E_1 and E_2 are strongly GSMBE if and only if, for all sets $\{B_i \in \mathcal{G} \mid i \in I\}$ such that $E_1\langle\!\langle A_i := B_i\rangle\!\rangle_{i\in I}, E_2\langle\!\langle A_i := B_i\rangle\!\rangle_{i\in I} \in \mathcal{G}$, it turns out that*

$$E_1\langle\!\langle A_i := B_i\rangle\!\rangle_{i\in I} \sim_{\mathrm{GSMBE}} E_2\langle\!\langle A_i := B_i\rangle\!\rangle_{i\in I}$$

■

We are now in a position to prove Theorem 13.2.

Proof of Theorem 13.2 Let $E_1, E_2 \in \mathcal{L}$ be pgc, and suppose that $Const(E_1) \cup Const(E_2)$ contains only $A \in Const$ as a free constant. Let $A_1 \stackrel{\Delta}{=} E_1\langle\!\langle A := A_1\rangle\!\rangle$ and $A_2 \stackrel{\Delta}{=} E_2\langle\!\langle A := A_2\rangle\!\rangle$ be in $\mathcal{G}$. We have to prove that $A_1 \sim_{\mathrm{GSMBE}} A_2$ whenever $E_1 \sim_{\mathrm{GSMBE}} E_2$. To achieve this, it sufficies to consider the family of relations

$$\mathcal{B}_{I_{Exec}} = \{(\langle F_1, Exec\rangle, \langle F_2, Exec\rangle) \mid F_1 \equiv F\langle\!\langle B := A_1\rangle\!\rangle \wedge F_2 \equiv F\langle\!\langle B := A_2\rangle\!\rangle \wedge$$
$$F \in \mathcal{L} \text{ pgc with at most } B \in Const(F) \text{ free}\}$$

where $Exec \subseteq AVId$, and to prove that $\mathbf{B}' = \{\mathcal{B}'_\psi \mid \psi \in \Psi\}$ where

- $\mathcal{B}'_{I_{Exec}} = \mathcal{B}_{I_{Exec}} \cup \mathcal{B}^{-1}_{I_{Exec}} \qquad \forall\, Exec \subseteq AVId$

- $\mathcal{B}'_\psi = \emptyset \qquad \forall\, \psi \neq I_{Exec}$

is a strong GSMBF up to $\sim_{\mathbf{GSMBF}}$: the result will follow by taking $Exec = \emptyset$ and $F \equiv B$. We first observe that for each $Exec \subseteq AVId$, $\mathcal{B}'_{I_{Exec}}$ is an equivalence relation: it is reflexive (because if we consider $F \in \mathcal{G}$ then it does not contain free variables and $F_1 \equiv F \equiv F_2$), symmetric (by definition), and transitive (for any $F \in \mathcal{L}$ pgc with at most $B \in Const(F)$ free we have $(\langle F_1, Exec\rangle, \langle F_2, Exec\rangle) \in \mathcal{B}'_{I_{Exec}}$ and $(\langle F_2, Exec\rangle, \langle F_1, Exec\rangle) \in \mathcal{B}'_{I_{Exec}}$, and transitivity is guaranteed by $F_1, F_2 \in \mathcal{G}$ and reflexivity). As a consequence $\mathbf{B}'$ satisfies the first three conditions of the definition of strong GSMBF up to $\sim_{\mathbf{GSMBF}}$. Given $(s_1, s_2) \in \mathcal{B}'_{I_{Exec}}$, we must prove that: for each equivalence class C of $stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})$, $tweight(s_1, C) = tweight(s_2, C)$, and for each equivalence class C of $temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})$, $ter(s_1, C) = ter(s_2, C)$.

We start by showing that $tweight(s_1, C) \leq tweight(s_2, C)$ for each $(s_1, s_2) \in \mathcal{B}'_{I_{Exec}}$ and C equivalence class of $stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})$. If (s_1, s_2) and C are such that $tweight(s_1, C) = 0$ there is nothing to prove, otherwise we prove the result by proceeding by induction on the maximum depth d of the inferences of the start moves for s_1 having derivative state in C.

- If $d = 1$, then only the rule for the prefix operator has been used to deduce the existing start move. Therefore $F \equiv \, <a^v_n, l, w>.F'$ and for $j \in \{1, 2\}$ we have $s_j \equiv \langle <a^v_n, w>.(F'\langle\!\langle B := A_j\rangle\!\rangle), \emptyset\rangle$. Since $(\langle <a^v_n, w>.(F'\langle\!\langle B := A_1\rangle\!\rangle), \{a^v_n\}\rangle, \langle <a^v_n, w>.(F'\langle\!\langle B := A_2\rangle\!\rangle), \{a^v_n\}\rangle) \in \mathcal{B}_{\{(a^v_n, a^v_n)\}}$, it turns out that $C = [\langle <a^v_n, w>.(F'\langle\!\langle B := A_1\rangle\!\rangle), \{a^v_n\}\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, \emptyset)} = [\langle <a^v_n, w>.(F'\langle\!\langle B := A_2\rangle\!\rangle), \{a^v_n\}\rangle\,]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, \emptyset)}$ hence $tweight(s_1, C) = w = tweight(s_2, C)$.

- If $d > 1$, then several subcases arise depending on the syntactical structure of F.

  - If $F \equiv F'/L$, then for $j \in \{1, 2\}$ we have $s_j \equiv \langle (F'\langle\!\langle B := A_j\rangle\!\rangle)/L, hide(Exec', L)\rangle$. Since s_1 has a start move having derivative state in C, there exist $G \in \mathcal{G}$ and $Exec''$ such that $\langle G/L, hide(Exec'', L)\rangle)$ is the derivative state and $C = [\langle G/L, hide(Exec'', L)\rangle\,]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})}$. Because of the congruence property w.r.t. "$\_/L$" of $\sim_{\mathbf{GSMBF}}$ (see lemma D.1) and of $\mathbf{B}'$ (easily provable) we have that:

    $$\langle H/L, hide(Exec'', L)\rangle \in [\langle G/L, hide(Exec'', L)\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})} \Rightarrow$$
    $$[\langle H, Exec''\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec'})}/L \subseteq [\langle G/L, hide(Exec'', L)\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})}$$ [1]

    As a consequence:

    $$[\langle G/L, hide(Exec'', L)\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})} = (\bigcup_i C_i/L) \cup D$$

    where C_i are distinguished equivalence classes $C_i = [\langle G_i, Exec''\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec'})}$ with $\langle G_i/L, hide(Exec'', L)\rangle \in [\langle G/L, hide(Exec'', L)\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})}$ and D is a set of states not having "$\_/L$" as outermost operator, hence not reachable from s_1 or s_2. Since d is

[1] Given a set of states C, C/L is the set of states $\{\langle E/L, hide(Exec, L)\rangle \mid \langle E, Exec\rangle \in C\}$.

the maximum depth of the inferences of the start moves for s_1 having derivative state in C, for each i: either (i) $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, Exec'\rangle$ has no start moves having derivative state in C_i, or (ii) $d-1$ is the maximum depth of the inferences of the start moves for $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, Exec'\rangle$ having derivative state in C_i. For $j \in \{1,2\}$ we have:

$$tweight(s_j, C) = \sum_i tweight(s_j, C_i/L)$$

where for each i:

$$tweight(s_j, C_i/L) = tweight(\langle F'\langle\!\langle B := A_j\rangle\!\rangle, Exec'\rangle, C_i)$$

By applying the induction hypothesis to each $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, Exec'\rangle, C_i$ such that $tweight(\langle F'\langle\!\langle B := A_j\rangle\!\rangle, Exec'\rangle, C_i) \neq 0$ we have that $tweight(s_1, C_i/L) \leq tweight(s_2, C_i/L)$ for any i. It follows that $tweight(s_1, C) \leq tweight(s_2, C)$.

– If $F \equiv F'[\varphi]$, then the proof is completely analogous to the one developed in the previous subcase.

– If $F \equiv F' + F''$, then for $j \in \{1,2\}$ we have $s_j \equiv \langle (F'\langle\!\langle B := A_j\rangle\!\rangle) + (F''\langle\!\langle B := A_j\rangle\!\rangle), \emptyset\rangle$. Since d is the maximum depth of the inferences of the start moves for s_1 having derivative state in C, for each $s \in \{\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, \langle F''\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle\}$: either (i) s has no start moves having derivative state in C, or (ii) $d-1$ is the maximum depth of the inferences of the start moves for s having derivative state in C. For $j \in \{1,2\}$ we have:

$$tweight(s_j, C) = tweight(\langle F'\langle\!\langle B := A_j\rangle\!\rangle, \emptyset\rangle, C) + tweight(\langle F''\langle\!\langle B := A_j\rangle\!\rangle, \emptyset\rangle, C)$$

By applying the induction hypothesis to $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C$ if $tweight(\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C) \neq 0$, and to $\langle F''\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C$ if $tweight(\langle F''\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C) \neq 0$, we have that $tweight(\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C) \leq tweight(\langle F'\langle\!\langle B := A_2\rangle\!\rangle, \emptyset\rangle, C)$ and $tweight(\langle F''\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C) \leq tweight(\langle F''\langle\!\langle B := A_2\rangle\!\rangle, \emptyset\rangle, C)$. It follows that $tweight(s_1, C) \leq tweight(s_2, C)$.

– If $F \equiv F' \,\|_S\, F''$, then for $j \in \{1,2\}$ we have $s_j \equiv \langle (F'\langle\!\langle B := A_j\rangle\!\rangle) \,\|_S (F''\langle\!\langle B := A_j\rangle\!\rangle), Exec\rangle$. Since s_1 has a start move having derivative state in C, there exist $G', G'' \in \mathcal{G}$ and $Exec'$ such that $\langle G' \,\|_S\, G'', Exec'\rangle)$ is the derivative state and $C = [\langle G' \,\|_S\, G'', Exec'\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$.

  * If $Exec' = Exec \cup \{a^v_{\swarrow loc}\}$ (or $Exec' = Exec \cup \{a^v_{\searrow loc}\}$) then $a \notin S$, $G'' \equiv F''\langle\!\langle B := A_1\rangle\!\rangle$ ($G' \equiv F'\langle\!\langle B := A_1\rangle\!\rangle$) and we are in the case of synchronization not taking place. In the following we treat only the case of the lefthand subterm making start moves, the other case is completely symmetrical.

    Because of the congruence property w.r.t. "$\_ \|_S \_$" of $\sim_{\mathbf{GSMBF}}$ (see lemma D.1) and of $\mathbf{B}'$ (easily provable) we have that:

    $\langle H \,\|_S (F''\langle\!\langle B := A_1\rangle\!\rangle), Exec'\rangle \in [\langle G' \,\|_S (F''\langle\!\langle B := A_1\rangle\!\rangle), Exec'\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})} \Rightarrow$
    $\langle \pi_t([\langle H, left(Exec')\rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{left(Exec)})}) \,\|_S (F''\langle\!\langle B := A_1\rangle\!\rangle), Exec'\rangle \subseteq$

$$[\langle G' \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$$ [2]

Moreover for the congruence property w.r.t. "$\_ \|_S \_$" of $\mathbf{B}'$ we have that for all $\mathcal{T} \in \mathcal{P}(\mathcal{G})$:

$$\langle \mathcal{T} \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle \subseteq [\langle G' \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})} \Leftrightarrow$$
$$\langle \mathcal{T} \,\|_S (F''\langle\!\langle B := A_2 \rangle\!\rangle), Exec' \rangle \subseteq [\langle G' \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$$

As a consequence: $[\langle G' \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})} =$

$$(\bigcup_i \langle \pi_t(C_i) \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle) \cup (\bigcup_i \langle \pi_t(C_i) \,\|_S (F''\langle\!\langle B := A_2 \rangle\!\rangle), Exec' \rangle) \cup D$$

where C_i are distinguished equivalence classes $C_i = [\langle G_i, \mathit{left}(Exec') \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{left}(Exec)})}$ with $\langle G_i \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle \in [\langle G' \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$ and D is a set of states not of the form $\langle H \,\|_S (F''\langle\!\langle B := A_j \rangle\!\rangle), Exec' \rangle$, hence not reachable from s_1 or s_2. Since d is the maximum depth of the inferences of the start moves for s_1 having derivative state in C, for each i: either (i) $\langle F'\langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(Exec) \rangle$ has no start moves having derivative state in C_i, or (ii) $d-1$ is the maximum depth of the inferences of the start moves for $\langle F'\langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(Exec) \rangle$ having derivative state in C_i. For $j \in \{1,2\}$ we have:

$$\mathit{tweight}(s_j, C) = \sum_i \mathit{tweight}(s_j, \langle \pi_t(C_i) \,\|_S (F''\langle\!\langle B := A_j \rangle\!\rangle), Exec' \rangle)$$

where for each i:

$$\mathit{tweight}(s_j, \langle \pi_t(C_i) \,\|_S F''\langle\!\langle B := A_j \rangle\!\rangle), Exec' \rangle) = \mathit{tweight}(\langle F'\langle\!\langle B := A_j \rangle\!\rangle, \mathit{left}(Exec) \rangle, C_i)$$

By applying the induction hypothesis to each $\langle F'\langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(Exec) \rangle, C_i$ such that $\mathit{tweight}(\langle F'\langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(Exec) \rangle, C_i) \neq 0$ we have that $\mathit{tweight}(s_1, \langle \pi_t(C_i) \,\|_S (F''\langle\!\langle B := A_1 \rangle\!\rangle), Exec' \rangle) \leq \mathit{tweight}(s_2, \langle \pi_t(C_i) \,\|_S (F''\langle\!\langle B := A_2 \rangle\!\rangle), Exec' \rangle)$ for any i. It follows that $\mathit{tweight}(s_1, C) \leq \mathit{tweight}(s_2, C)$.

* If $Exec' = Exec \cup \{a^o_{\langle loc' | loc'' \rangle}\}$ then $a \in S$ and we are in the case of synchronization.

Because of the congruence property w.r.t. "$\_ \|_S \_$" of $\sim_{\mathbf{GSMBF}}$ and of $\mathbf{B}'$ we have that:

$$\langle H' \,\|_S H'', Exec' \rangle \in [\langle G' \,\|_S G'', Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})} \Rightarrow$$
$$\langle \pi_t([\langle H', \mathit{left}(Exec') \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{left}(Exec)})}) \,\|_S$$
$$\pi_t([\langle H'', \mathit{right}(Exec') \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{right}(Exec)})}), Exec' \rangle \subseteq$$
$$[\langle G' \,\|_S G'', Exec' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$$ [3]

As a consequence:

[2] Given a set of states C, $\pi_t(C)$ is the set of terms included in the states of C. Given a set of terms $\mathcal{T}$, a term E and a set $Exec$, $\langle \mathcal{T} \,\|_S E, Exec \rangle$ is the set of states $\{\langle F \,\|_S E, Exec \rangle \mid F \in \mathcal{T}\}$.

[3] Given a set of terms $\mathcal{T}_1$, a set of terms $\mathcal{T}_2$ and a set $Exec$, $\langle \mathcal{T}_1 \,\|_S \mathcal{T}_2, Exec \rangle$ is the set of states $\{\langle E' \,\|_S E'', Exec \rangle \mid E' \in \mathcal{T}_1 \wedge E'' \in \mathcal{T}_2\}$.

$$[\langle G' \parallel_S G'', \mathit{Exec}' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})} = (\bigcup_i \langle \pi_t(C'_i) \parallel_S \pi_t(C''_i), \mathit{Exec}' \rangle) \cup D$$

where C'_i, C''_i are distinguished pairs of equivalence classes $C'_i = [\langle G'_i, \mathit{left}(\mathit{Exec}') \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{left(Exec)})}$, $C''_i = [\langle G''_i, \mathit{right}(\mathit{Exec}') \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{right(Exec)})}$ with $\langle G'_i \parallel_S G''_i, \mathit{Exec}' \rangle \in [\langle G' \parallel_S G'', \mathit{Exec}' \rangle]_{stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})}$ and D is a set of states not having "$\_ \parallel_S \_$" as outermost operator, hence not reachable from s_1 or s_2. Since d is the maximum depth of the inferences of the start moves for s_1 having derivative state in C, for each i: either (i) $\langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(\mathit{Exec}) \rangle$ or $\langle F'' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{right}(\mathit{Exec}) \rangle$ has no start moves having derivative state in C'_i (C''_i), or (ii) $d-1$ is the maximum depth of the inferences of the start moves for $\langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(\mathit{Exec}) \rangle$ and for $\langle F'' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{right}(\mathit{Exec}) \rangle$ having derivative state in C'_i (C''_i). For $j \in \{1, 2\}$ we have:

$$\mathit{tweight}(s_j, C) = \sum_i \mathit{tweight}(s_j, \langle \pi_t(C'_i) \parallel_S \pi_t(C''_i), \mathit{Exec}' \rangle)$$

where for each i:

$$\mathit{tweight}(s_j, \langle \pi_t(C'_i) \parallel_S \pi_t(C''_i), \mathit{Exec}' \rangle) = \mathit{tweight}(\langle \langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(\mathit{Exec}) \rangle, C'_i) \cdot \mathit{tweight}(\langle \langle F'' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{right}(\mathit{Exec}) \rangle, C''_i)$$

By applying the induction hypothesis to each $\langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(\mathit{Exec}) \rangle, C'_i$ and $\langle F'' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{right}(\mathit{Exec}) \rangle, C''_i$ such that both $\mathit{tweight}(\langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{left}(\mathit{Exec}) \rangle, C'_i) \neq 0$ and $\mathit{tweight}(\langle F'' \langle\!\langle B := A_1 \rangle\!\rangle, \mathit{right}(\mathit{Exec}) \rangle, C''_i) \neq 0$, we have that $\mathit{tweight}(s_1, \langle \pi_t(C'_i) \parallel_S \pi_t(C''_i), \mathit{Exec}' \rangle) \leq \mathit{tweight}(s_2, \langle \pi_t(C'_i) \parallel_S \pi_t(C''_i), \mathit{Exec}' \rangle)$ for any i. It follows that $\mathit{tweight}(s_1, C) \leq \mathit{tweight}(s_2, C)$.

– If $F \equiv B'$, then for $j \in \{1, 2\}$ we have $s_j \equiv \langle B' \langle\!\langle B := A_j \rangle\!\rangle, \emptyset \rangle$.

  * If $B' \equiv B$, then for $j \in \{1, 2\}$ we have $s_j \equiv \langle A_j, \emptyset \rangle$. Since d is the maximum depth of the inferences of the start moves for s_1 having derivative state in C, then $d-1$ is the maximum depth of the inferences of the start moves for $\langle E_1 \langle\!\langle A := A_1 \rangle\!\rangle, \emptyset \rangle$ having derivative state in C. For $j \in \{1, 2\}$ we have:

$$\mathit{tweight}(s_j, C) = \mathit{tweight}(\langle E_j \langle\!\langle A := A_j \rangle\!\rangle, \emptyset \rangle, C)$$

  By applying the induction hypothesis to $\langle E_1 \langle\!\langle A := A_1 \rangle\!\rangle, \emptyset \rangle, C$ we have that $\mathit{tweight}(\langle E_1 \langle\!\langle A := A_1 \rangle\!\rangle, \emptyset \rangle, C) \leq \mathit{tweight}(\langle E_1 \langle\!\langle A := A_2 \rangle\!\rangle, \emptyset \rangle, C)$. From the fact that $E_1 \sim_{\mathrm{GSMBE}} E_2$ we derive $\mathit{tweight}(\langle E_1 \langle\!\langle A := A_2 \rangle\!\rangle, \emptyset \rangle, C) = \mathit{tweight}(\langle E_2 \langle\!\langle A := A_2 \rangle\!\rangle, \emptyset \rangle, C)$. It follows that $\mathit{tweight}(s_1, C) \leq \mathit{tweight}(s_2, C)$.

  * If $B' \not\equiv B$, then let $B' \stackrel{\Delta}{=} F'$ be the defining equation of B'. Since d is the maximum depth of the inferences of the start moves for s_1 having derivative state in C, then $d-1$ is the maximum depth of the inferences of the start moves for $\langle F' \langle\!\langle B := A_1 \rangle\!\rangle, \emptyset \rangle$ having derivative

state in C. For $j \in \{1,2\}$ we have:

$$tweight(s_j, C) = tweight(\langle F'\langle\!\langle B := A_j \rangle\!\rangle, \emptyset\rangle, C)$$

By applying the induction hypothesis to $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C$ we have that $tweight(\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, C) \leq tweight(\langle F'\langle\!\langle B := A_2\rangle\!\rangle, \emptyset\rangle, C)$. It follows that $tweight(s_1, C) \leq tweight(s_2, C)$.

By applying a symmetric argument we have also that $tweight(s_2, C) \leq tweight(s_1, C)$ for each $(s_1, s_2) \in \mathcal{B}'_{I_{Exec}}$ and C equivalence class of $stmap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})$, hence we conclude that $tweight(s_1, C) = tweight(s_2, C)$.

In order to complete the proof, we show that $ter(s_1, C) \Rightarrow ter(s_2, C)$ for each $(s_1, s_2) \in \mathcal{B}'_{I_{Exec}}$ and C equivalence class of $temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec})$. If (s_1, s_2) and C are such that $ter(s_1, C) = False$ there is nothing to prove, otherwise we prove the result by proceeding by induction on the depth d of the inference of the termination move for s_1 having derivative state in C.

- If $d = 1$, then only the rule for the prefix operator has been used to deduce the termination move. Therefore $F \equiv {<}a_n^v, l, w{>}.F'$ and for $j \in \{1,2\}$ we have $s_j \equiv \langle {<}a_n^v, w{>}.(F'\langle\!\langle B := A_j\rangle\!\rangle), \{a_n^v\}\rangle$. Since $(\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle, \langle F'\langle\!\langle B := A_2\rangle\!\rangle, \emptyset\rangle) \in \mathcal{B}_\emptyset$, it turns out that $C = [\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \emptyset\rangle]_{temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, \emptyset)} = [\langle F'\langle\!\langle B := A_2\rangle\!\rangle, \emptyset\rangle]_{temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, \emptyset)}$ hence $ter(s_1, C) = True = ter(s_2, C)$.

- If $d > 1$, then the following subcases arise depending on the syntactical structure of F.

  - If $F \equiv F'/L$, then for $j \in \{1,2\}$ we have $s_j \equiv \langle (F'\langle\!\langle B := A_j\rangle\!\rangle)/L, hide(Exec', L)\rangle$. Since s_1 has a termination move having derivative state in C, such that the depth of its inference is d, $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, Exec'\rangle$ has a termination move having derivative state $\langle G, Exec''\rangle \in C'$ with C' equivalence class of $temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{Exec'})$, such that the depth of its inference is $d-1$ and $C = [\langle G/L, hide(Exec'', L)\rangle]_{temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})}$. From the induction hypothesis, it follows that $\langle F'\langle\!\langle B := A_2\rangle\!\rangle, Exec'\rangle$ has a termination move having derivative state $\langle H, Exec''\rangle \in C'$; therefore s_2 has a termination move having derivative state $\langle H/L, hide(Exec'', L)\rangle$. Since $[\langle H/L, hide(Exec'', L)\rangle]_{temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})} = [\langle G/L, hide(Exec'', L)\rangle]_{temap((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{hide(Exec', L)})}$ for the congruence property w.r.t. "$\_/L$" of $\sim_{\mathbf{GSMBF}}$ and of $\mathbf{B}'$, it follows that $ter(s_1, C) \Rightarrow ter(s_2, C)$.
  - If $F \equiv F'[\varphi]$, then the proof is completely analogous to the one developed in the first subcase.
  - If $F \equiv F' \,\|_S\, F''$, then for $j \in \{1,2\}$ we have $s_j \equiv \langle (F'\langle\!\langle B := A_j\rangle\!\rangle) \,\|_S (F''\langle\!\langle B := A_j\rangle\!\rangle), Exec\rangle$. Since s_1 has a termination move having derivative state in C, such that the depth of its inference is d, we have the following two cases.

- $*$ Only a subterm $F'\langle\!\langle B := A_1\rangle\!\rangle$ or $F''\langle\!\langle B := A_1\rangle\!\rangle$ is involved in the termination of actions. In the following we treat only the case of the lefthand subterm making a termination move, the other case is completely symmetrical.

  We have that $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \mathit{left}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle G, \mathit{left}(\mathit{Exec}')\rangle \in C'$ with C' equivalence class of $\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{left}(\mathit{Exec})})$, such that the depth of its inference is $d-1$ and $C = [\langle G \,\|_S (F''\langle\!\langle B := A_1\rangle\!\rangle), \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})}$. From the induction hypothesis, it follows that $\langle F'\langle\!\langle B := A_2\rangle\!\rangle, \mathit{left}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle H, \mathit{left}(\mathit{Exec}')\rangle \in C'$; therefore s_2 has a termination move having derivative state $\langle H \,\|_S (F''\langle\!\langle B := A_2\rangle\!\rangle), \mathit{Exec}'\rangle$. Since $[\langle G \,\|_S (F''\langle\!\langle B := A_1\rangle\!\rangle), \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})} = [\langle H \,\|_S (F''\langle\!\langle B := A_2\rangle\!\rangle), \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})}$ for the congruence property w.r.t. "$\_ \,\|_S \_$" of $\sim_{\mathbf{GSMBF}}$ and of $\mathbf{B}'$, it follows that $\mathit{ter}(s_1, C) \Rightarrow \mathit{ter}(s_2, C)$.

- $*$ Both subterms are involved in the termination of actions.

  We have that $\langle F'\langle\!\langle B := A_1\rangle\!\rangle, \mathit{left}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle G', \mathit{left}(\mathit{Exec}')\rangle \in C'$ with C' equivalence class of $\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{left}(\mathit{Exec})})$, such that the depth of its inference is at most $d-1$, and $\langle F''\langle\!\langle B := A_1\rangle\!\rangle, \mathit{right}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle G'', \mathit{right}(\mathit{Exec}')\rangle \in C''$ with C'' equivalence class of $\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{right}(\mathit{Exec})})$, such that the depth of its inference is at most $d-1$; besides, $C = [\langle G' \,\|_S G'', \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})}$. From the induction hypothesis, it follows that $\langle F'\langle\!\langle B := A_2\rangle\!\rangle, \mathit{left}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle H', \mathit{left}(\mathit{Exec}')\rangle \in C'$ and $\langle F''\langle\!\langle B := A_2\rangle\!\rangle, \mathit{right}(\mathit{Exec})\rangle$ has a termination move having derivative state $\langle H'', \mathit{right}(\mathit{Exec}')\rangle \in C''$; therefore s_2 has a termination move having derivative state $\langle H' \,\|_S H'', \mathit{Exec}'\rangle$. Since $[\langle G' \,\|_S G'', \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})} = [\langle H' \,\|_S H'', \mathit{Exec}'\rangle]_{\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})}$ for the congruence property w.r.t. "$\_ \,\|_S \_$" of $\sim_{\mathbf{GSMBF}}$ and of $\mathbf{B}'$, it follows that $\mathit{ter}(s_1, C) \Rightarrow \mathit{ter}(s_2, C)$.

By applying a symmetric argument we have also that $\mathit{ter}(s_2, C) \Rightarrow \mathit{ter}(s_1, C)$ for each $(s_1, s_2) \in \mathcal{B}'_{I_{\mathit{Exec}}}$ and C equivalence class of $\mathit{temap}((\mathbf{B}' \cup \sim_{\mathbf{GSMBF}})^+, I_{\mathit{Exec}})$, hence we conclude that $\mathit{ter}(s_1, C) = \mathit{ter}(s_2, C)$. ■

References

[1] L. Aceto, *"On "Axiomatising Finite Concurrent Processes" "* in SIAM Journal on Computing 23(4):852-863, 1994

[2] L. Aceto, *"A Static View of Localities"*, in Formal Aspects of Computing, 6:201-222, 1994

[3] L. Aceto, M.C.B. Hennessy, *"Towards Action Refinement in Process Algebras"*, in Information and Computation 103:204-269, 1993

[4] L. Aceto, M.C.B. Hennessy, *"Adding Action Refinement to a Finite Process Algebra"*, in Information and Computation 115:179-247, 1994

[5] L. Aceto, D. Murphy, *"On the ill-timed but well-caused"*, in Proc. of the *4th Int. Conf. on Concurrency Theory (CONCUR '93)*, LNCS 715:97-111, Springer-Verlag, 1993

[6] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *"Modelling with Generalized Stochastic Petri Nets"*, John Wiley & Sons, 1995

[7] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto, A. Valenzano, *"A LOTOS Extension for the Performance Analysis of Distributed Systems"*, in IEEE/ACM Trans. on Networking 2:151-164, 1994

[8] A. Aldini, M. Bernardo, R. Gorrieri, *"An Algebraic Model for Evaluating the Performance of an ATM Switch with Explicit Rate Marking"*, in Proc. of the *7th Int. Workshop on Process Algebras and Performance Modeling (PAPM '99)*, Prensas Universitarias de Zaragoza, pp. 119-138, Zaragoza (Spain), 1999

[9] A. Aldini, M. Bernardo, R. Gorrieri, M. Roccetti, *"A Simulative Analysis of Internet Audio Mechanisms Using Formal Methods"*, in ACM Trans. on Modelling and Computer Simulation 11(1), pp. 1-42, 2001

[10] A. Aldini, M. Bravetti, *"An Asynchronous Calculus for Generative-reactive Probabilistic Systems"*, in Proc. of the *8th Int. Workshop on Process Algebras and performance Modeling (PAPM 2000)*, J.D.P. Rolim et al. editors, pp. 591-605, Carleton Scientific, Geneva (Switzerland), July 2000

[11] R. Alur, C. Courcoubetis, D. Dill *"Model-Checking in Dense Real-Time"*, in Information and Computation 104:2-34, 1993

[12] E. Badouel, P. Darondeau, *"On Guarded Recursion"*, in Theoretical Computer Science 82(2):403-408, 1991

[13] J.C.M. Baeten, J.A. Bergstra, S.A. Smolka, *"Axiomatizing Probabilistic Processes: ACP with Generative Probabilities"*, in Information and Computation 121:234-255, 1995

[14] J.C.M. Baeten, W.P. Weijland, *"Process Algebra"*, Cambridge University Press, 1990

[15] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, *"On the Logical Characterisation of Performability Properties"*, in Proc. of the *27th Int. Coll. on Automata, Languages and Programming (ICALP '00)*, LNCS 1853:780-792, Geneve (Switzerland), 2000

[16] C. Baier, H. Hermanns, *"Weak Bisimulation for Fully Probabilistic Processes"*, in Proc. of the *9th Int. Conf. on Computer Aided Verification (CAV '97)*, LNCS 1254:119-130, 1997

[17] C. Baier, J.-P. Katoen, H. Hermanns, *"Approximate Symbolic Model Checking of Continuous Time Markov Chains"*, in Proc. of the *10th Int. Conf. on Concurrency Theory (CONCUR '00)*, LNCS 1664:146-162, Eindhoven (The Netherlands), 1999

[18] J.A. Bergstra, J.W. Klop, *"Algebra of Communicating Processes with Abstraction"*, in Theoretical Computer Science 37(1):77-121, 1985

[19] M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph.D. Thesis, University of Bologna (Italy), 1999

[20] M. Bernardo, *"An Algebra-Based Method to Associate Rewards with EMPA Terms"*, in Proc. of the *24th Int. Coll. on Automata, Languages and Programming (ICALP '97)*, LNCS 1256:358-368, Bologna (Italy), 1997

[21] M. Bernardo, M. Bravetti, *"Performance Measure Sensitive Congruences for Markovian Process Algebras"*, to appear in Theoretical Computer Science

[22] M. Bernardo, M. Bravetti, *"Reward Based Congruences: Can We Aggregate More?"*, in Proc. of the *Int. Joint Workshop on Process Algebra and Performance Modelling, Probabilistic Methods in Verification (PAPM-PROBMIV 2001)*, LNCS 2165:136-151, Aachen (Germany), September 2001

[23] M. Bernardo, M. Bravetti, *"Functional and Performance Modeling and Analysis of Token Ring using EMPA"*, in Proc. of the *6th Italian Conf. on Theoretical Computer Science (ICTCS '98)*, P. Degano, U. Vaccaro and G. Pirillo editors, World Scientific, pp. 204-215, Prato (Italy), November 1998

[24] M. Bernardo, P. Ciancarini, L. Donatiello, *"ÆMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures"*, in Proc. of the *2nd Int. Workshop on Software and Performance (WOSP '00)*, ACM Press, pp. 1-11, Ottawa (Canada), 2000

[25] M. Bernardo, W.R. Cleaveland, *"A Theory of Testing for Markovian Processes"*, in Proc. of the *11th Int. Conf. on Concurrency Theory (CONCUR '00)*, LNCS 1877:305-319, State College (PA), 2000

[26] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart, *"TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems"*, in Proc. of the *IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '98)*, Kluwer, 457-467, Paris (France), 1998

[27] M. Bernardo, R. Gorrieri, *"A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time"*, in Theoretical Computer Science 202:1-54, 1998. Corrigendum in Theoretical Computer Science 254:691-694, 2001.

[28] M. Bernardo, R. Gorrieri, M. Roccetti, *"Formal Performance Modelling and Evaluation of an Adaptive Mechanism for Packetised Audio over the Internet"*, in Formal Aspects of Computing 10:313-337, 1999

[29] T. Bolognesi, E. Brinksma, *"Introduction to the ISO Specification Language LOTOS"*, in Computer Networks and ISDN Systems 14:25-59, 1987

[30] G. Boudol, I. Castellani, *"Permutation of Transitions: An Event Structure Semantics for CCS and SCCS"*, in Proc. of the *Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354:411-427, Noordwijkerhout (The Netherlands), 1988

[31] G. Boudol, I. Castellani, M.C.B. Hennessy, A. Kiehn, *"A theory of processes with localities"*, in Formal Aspects of Computing 6(2):165-200, 1994

[32] M. Bravetti, *"Towards the Integration of Real-Time and Probabilistic-Time Process Algebras"*, in Proc. of the *3rd European Research Seminar on Advances in Distributed Systems (ERSADS '99)*, Madeira Island (Portugal), April 1999

[33] M. Bravetti, A. Aldini, *"Discrete Time Generative-Reactive Probabilistic Processes with Different Advancing Speeds"*, to appear in Theoretical Computer Science

[34] M. Bravetti, A. Aldini, *"Non-Determinism in Probabilistic Timed Systems with General Distributions"*, to appear in Proc. of the *2nd Int. Workshop on Models for Time-Critical Systems (MTCS 2001)*, ENTCS, Aalborg (Denmark), August 2001

[35] M. Bravetti, A. Aldini, *"Expressing Processes with Different Action Durations through Probabilities"*, in Proc. of the *Int. Joint Workshop on Process Algebra and Performance Modelling, Probabilistic Methods in Verification (PAPM-PROBMIV 2001)*, LNCS 2165:168-183, Aachen (Germany), September 2001

[36] M. Bravetti, M. Bernardo, *"Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time"*, in Proc. of the *1st Int. Workshop on Models for Time-Critical Systems (MTCS 2000)*, ENTCS 39(3), State College (PA), 2000

[37] M. Bravetti, R. Bernardo, *"Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time"*, Technical Report UBLCS-2000-01, University of Bologna (Italy), January 2000 (revised January 2001)

[38] M. Bravetti, M. Bernardo, R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in Proc. of the *6th Int. Workshop on Process Algebras and Performance Modeling (PAPM '98)*, C. Priami editor, pp. 153-164, Nice (France), September 1998

[39] M. Bravetti, M. Bernardo, R. Gorrieri, *"Towards Performance Evaluation with General Distributions in Process Algebras"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, D. Sangiorgi and R. de Simone editors, LNCS 1466:405-422, Nice (France), September 1998

[40] M. Bravetti, M. Bernardo, R. Gorrieri, *"GSMPA: A Core Calculus with Generally Distributed Durations"*, Technical Report UBLCS-98-06, University of Bologna (Italy), June 1998

[41] M. Bravetti, M. Bernardo, R. Gorrieri, *"From EMPA to GSMPA: Allowing for General Distributions"*, in Proc. of the *5th Int. Workshop on Process Algebras and Performance Modeling (PAPM '97)*, E. Brinksma and A. Nymeyer editors, pp. 17-33, Enschede (The Netherlands), June 1997

[42] M. Bravetti, M. Bernardo, R. Gorrieri, *"Generalized Semi-Markovian Process Algebra"*, Technical Report UBLCS-97-09, University of Bologna (Italy), October 1997

[43] M. Bravetti, R. Gorrieri, *"Deciding and Axiomatizing Weak ST Bisimulation for a Process Algebra with Recursion and Action Refinement"*, to appear in ACM Transactions on Computational Logic

[44] M. Bravetti, R. Gorrieri, *"The Theory of Interactive Generalized Semi-Markov Processes"*, to appear in Theoretical Computer Science

[45] M. Bravetti, R. Gorrieri, *"A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors"*, in Proc. of the *27th Int. Colloquium on Automata, Languages and Programming (ICALP 2000)*, U. Montanari, J.D.P. Rolim and E. Welzl editors, LNCS 1853:744-755, Geneva (Switzerland), July 2000

[46] M. Bravetti, R. Gorrieri, *"A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors"*, Technical Report UBLCS-99-18, University of Bologna (Italy), July 1999

[47] M. Bravetti, R. Gorrieri, *"Interactive Generalized Semi-Markov Processes"*, in Proc. of the *7th Int. Workshop on Process Algebras and Performance Modeling (PAPM '99)*, J. Hillston and M. Silva editors, pp. 83-98, Zaragoza (Spain), September 1999

[48] M. Bravetti, R. Gorrieri, *"Axiomatizing ST Bisimulation for a Process Algebra with Recursion and Action Refinement (Extended Abstract)"*, in Proc. of the *6th Int. Workshop on Expressiveness in Concurrency (EXPRESS '99)*, ENTCS 27, Eindhoven (The Netherlands), August 1999

[49] E. Brinksma, J.P. Katoen, R. Langerak, D. Latella, *"A Stochastic Causality-Based Process Algebra"*, in Computer Journal 38:553-565, 1995

[50] J. Bryans, J. Derrick, *"Stochastic Specification and Verification"*, In Proc. of the *3rd Irish Workshop in Formal Methods*, Electronic Workshops in Computing, July 1999

[51] P. Buchholz, *"Markovian Process Algebra: Composition and Equivalence"*, in Proc. of the *2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM '94)*, pp. 11-30, Erlangen (Germany), 1994

[52] N. Busi, R.J. van Glabbeek, R. Gorrieri, *"Axiomatising ST-Bisimulation Equivalence"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '94)*, pp. 169-188, S. Miniato (Italy), 1994

[53] L. Cardelli, *"Real time agents"*, in Proc. of the *9th Int. Colloquium on Automata, Languages and Programming (ICALP '82)*, LNCS 140:94-106, 1982

[54] C.G. Cassandras, *"Discrete Event Systems. Modeling and Performance Analysis"*, Aksen Associates, Irwin, 1993

[55] I. Castellani, *"Observing Distribution In Processes: Static and Dynamic Localities"*, in Int. Journal of Foundations of Computer Science 6:353-393, 1995

[56] G. Ciardo, J. Muppala, K.S. Trivedi, *"On the Solution of GSPN Reward Models"*, in Performance Evaluation 12:237-253, 1991

[57] G. Clark, *"Formalising the Specification of Rewards with PEPA"*, in Proc. of the *4th Workshop on Process Algebras and Performance Modelling (PAPM '96)*, CLUT, pp. 139-160, Torino (Italy), 1996

[58] G. Clark, S. Gilmore, J. Hillston, *"Specifying Performance Measures for PEPA"*, in Proc. of the *5th AMAST Int. Workshop on Formal Methods for Real Time and Probabilistic Systems (ARTS '99)*, LNCS 1601:211-227, Bamberg (Germany), 1999

[59] E.M. Clarke, O. Grumberg, D.A. Peled, *"Model Checking"*, MIT Press, 1999

[60] W.R. Cleaveland, M.C.B. Hennessy, *"Priorities in Process Algebra"*, in Information and Computation 87:58-77, 1990

[61] W.R. Cleaveland, G. Luttgen, V. Natarajan, *"A Process Algebra with Distributed Priorities"*, in Proc. of the *7th Int. Conf. on Concurrency Theory (CONCUR '96)*, LNCS 1119:34-49, 1996

[62] W.R. Cleaveland, G. Luttgen, V. Natarajan, *"Priority in Process Algebras"*, in Handbook of Process Algebra, Elsevier, pp. 711-765,2001

[63] W.R. Cleaveland, S. Sims, *"The NCSU Concurrency Workbench"*, in Proc. of the *8th Int. Conf. on Computer Aided Verification (CAV '96)*, LNCS 1102:394-397, New Brunswick (NJ), 1996

[64] F. Corradini, M. Pistore, *"Closed Interval Process Algebra versus Interval Process Algebra"*, in Acta Informatica 37, pp. 467-509, 2001

[65] D.R. Cox, *"The Analysis of non-Markovian Stochastic Processes by the Inclusion of Supplementary Variables"*, in Proc. of the Cambridge Philosophical Society 51:433-440, 1955

[66] P.R. D'Argenio, *"Algebras and Automata for Timed and Stochastic Systems"*, Ph.D. Thesis, Univ. Twente, 1997

[67] P.R. D'Argenio, *"A Compositional Translation of Stochastic Automata into Timed Automata"*, Technical Report CTIT 00-08, Univ. Twente, May 2000

[68] P.R. D'Argenio, H. Hermanns, J.P. Katoen, *"On Generative Parallel Composition"*, in Proc. of the *"1st Int. Workshop on Probabilistic Methods in Verification (PROBMIV '98)*, ENTCS 22, 2000

[69] P.R. D'Argenio, J.P. Katoen, E. Brinksma, *"An Algebraic Approach to the Specification of Stochastic Systems (extended abstract)"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '98)*, New York (USA), 1998

[70] P.R. D'Argenio, J.-P. Katoen, E. Brinksma, *"A Stochastic Automata Model and its Algebraic Approach"* in Proc. of the *5th Workshop on Process Algebras and Performance Modelling (PAPM '97)*, pp. 1-16, Enschede (The Netherlands), 1997

[71] P. Darondeau, P. Degano, *"Causal Trees"*, in Automata, Languages and Programming, LNCS 372:234-248, 1989

[72] P. Degano, R. De Nicola, U. Montanari, "Partial Ordering Description of Nondeterministic Concurrent Systems", in Proc. of the *REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354:438-466, Noordwijkerhout, 1989

[73] P. Degano, R. Gorrieri, *"A Causal Operational Semantics of Action Refinement"*, in Information and Computation 122:97-119, 1995.

[74] R. De Nicola, M. C. B. Hennessy, *"Testing Equivalences for Processes"*, in Theoretical Computer Science 34:83-133, 1984

[75] R. Devillers, *"Maximality Preserving Bisimulation"*, in Theoretical Computer Science 102:165-183, 1992

[76] S.B. Fröschle, *"Decidability of Plain and Hereditary History-preserving Bisimilarity for BPP"*, in Proc. of the *6th Int. Workshop on Expressiveness in Concurrency (EXPRESS '99)*, ENTCS 27, Eindhoven (The Netherlands), August 1999

[77] S.B. Fröschle, T.T. Hildebrandt, *"On Plain and Hereditary History-preserving Bisimulation"*, in Proc. of the *24th Int. Symp. on Mathematical Foundations of Computer Science (MFCS '99)*, LNCS 1672:354-365, Szklarska Poreba (Poland), 1999

[78] R.J. van Glabbeek, *"The refinement theorem for ST-bisimulation semantics"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '90)*, pp. 27-52, Sea of Gallilea (Israel), 1990

[79] R.J. van Glabbeek, U. Goltz, *"Equivalence Notions for Concurrent Systems and Refinement of Actions"*, Arbeitspapiere der GMD 366, Sankt Augustin, Germany 1989. Extended abstract in: Proc. of the *14th Int. Symp. on Mathematical Foundations of Computer Science (MFCS '89)*, A. Kreczmar and G. Mirkowska editors, LNCS 379:237-248, Poræbka-Kozubnik (Poland), 1989.

[80] R.J. van Glabbeek, S.A. Smolka, B. Steffen, *"Reactive, Generative and Stratified Models of Probabilistic Processes"*, in Information and Computation 121:59-80, 1995

[81] R.J. van Glabbeek, F.W. Vaandrager, *"Petri Net Models for Algebraic Theories of Concurrency"*, in Proc. of the *Conf. on Parallel Architectures and Languages Europe (PARLE '87)*, LNCS 259:224-242, Eindhoven (The Netherlands), 1987

[82] P.W. Glynn, *"A GSMP formalism for discrete event simulation"*, in Proc. of the IEEE, 77(1): 14-23, 1989

[83] R. Gorrieri, C. Laneve, *"The limit of $split_n$ bisimulations for CCS agents"*, in Proc. of the *Int. Symp. on Mathematical Foundations of Computer Science (MFCS '91)*, LNCS 520:170-180, 1991

[84] R. Gorrieri, C. Laneve, *"An Extensional Formulation of ST Bisimulation"*, in Proc. of the *4th Italian Conference on Theoretical Computer Science (ICTCS '92)*, A. Marchetti Spaccamela, P. Mentrasti and M. Venturini Zilli editors, World Scientific, L'Aquila (Italy), pp. 239-253, 1992

[85] R. Gorrieri, C. Laneve, *"Split and ST Bisimulation Semantics"*, in Information and Computation 118:272-288, 1995

[86] R. Gorrieri, A. Rensink, *"Action Refinement"*, in Handbook of Process Algebra, Elsevier, 2001

[87] N. Götz, U. Herzog, M. Rettelbach, *"TIPP - A Stochastic Process Algebra"*, in Proc. of the *1st Workshop on Process Algebras and Performance Modelling (PAPM '93)*, pp. 31-36, Edinburgh (UK), 1993

[88] N. Götz, U. Herzog, M. Rettelbach, *"Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis Using Stochastic Process Algebras"*, in Proc. of the *16th Int. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE '93)*, LNCS 729:121-146, Rome (Italy), 1993

[89] J.F. Groote, *"Transition System Specifications with Negative Premises"*, in Theoretical Computer Science 118(2):263-299, 1993

[90] P.G. Harrison, B. Strulo, *"Stochastic Process Algebra for Discrete Event Simulation"*, in *Quantitative Methods in Parallel Systems*, Springer, pp. 18-37, 1995

[91] B.R. Haverkort, K.S. Trivedi, *"Specification Techniques for Markov Reward Models"*, in Discrete Event Dynamic Systems: Theory and Applications 3:219-247, 1993

[92] W. Henderson, *"Insensitivity and Reversed Markov Processes"*, in Advanced Applied Probab. 15:752-768, 1983

[93] W. Henderson, D. Lucic, *"Aggregation and Disaggregation through Insensitivity in Stochastic Petri Nets"*, in Performance Evaluation 17:91-114, 1993

[94] W. Henderson, P. Taylor, *"Insensitivity of Processes with Interruptions"*, in Journal of Applied Probability 26:242-258, 1989

[95] M.C.B. Hennessy, *"Algebraic Theory of Processes"*, MIT Press, 1988

[96] M.C.B. Hennessy, *"A Proof System for Weak ST-bisimulation over a Finite Process Algebra*, Computer Science Report 6/91, University of Sussex (UK), 1991.

[97] M.C.B. Hennessy, R. Milner, *"Algebraic Laws for Nondeterminism and Concurrency"*, in Journal of the ACM 32:137-161, 1985

[98] M. Hennessy, T. Regan, *"A Process Algebra for Timed Systems"*, in Information and Computation, 117(2):221-239, 1995

[99] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, Universität Erlangen-Nürnberg (Germany), 1998

[100] H. Hermanns, *"An Operator for Symmetry Representation and Exploitation in Stochastic Process Algebras"*, in Proc. of the *5th Workshop on Process Algebras and Performance Modeling*, pp. 55-70, Twente (The Netherlands), 1997

[101] H. Hermanns, *"Leistungsvorhersage von Verhaltensbeschreibungen mittels Temporaler Logik"*, contribution to the *GI/ITG Fachgespraech '95: Formale Beschreibungstechniken fuer Verteilte Systeme*, 1995

[102] H. Hermanns, M. Lohrey, *"Priority and Maximal Progress Are Completely Axiomatisable (Extended Abstract)"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:237-252, Nice (France), 1998

[103] H. Hermanns, M. Lohrey, *"Observational Congruence in a Stochastic Timed Calculus with Maximal Progress"*, Technical Report IMMD-VII/7-97, Universität Erlangen-Nürnberg, 1997

[104] H. Hermanns, M. Rettelbach, *"Syntax, Semantics, Equivalences, and Axioms for MTIPP"*, in Proc. of the *2nd Int. Workshop on Process Algebra and Performance Modelling (PAPM '94)*, pp. 71-87, Erlangen (Germany), 1994

[105] H. Hermanns, M. Rettelbach, T. Weiss, *"Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching"*, The Computer Journal (38)7, 1995

[106] D.P. Heyman, M.J. Sobel, *"Stochastic Models in Operations Research"*, McGraw-Hill, 1982

[107] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

[108] J. Hillston, *"The Nature of Synchronisation"*, in *Proc. of the 2nd Workshop on Process Algebras and Performance Modeling (PAPM '94)*, pp. 51-70, Erlangen (Germany), 1994

[109] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[110] R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971

[111] M. Jurdziński, M. Nielsen, *"Hereditary History Preserving Bisimilarity Is Undecidable"*, in Proc. of the *17th Int. Symp. on Theoretical Aspects of Computer Science (STACS 2000)*, LNCS 1770:358-369, Lille (France), February 2000

[112] C.C. Jou, S.A. Smolka, *"Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes"*, in Proc. of the *1st Int. Conf. on Concurrency Theory (CONCUR '90)*, LNCS 458:367-383, 1990

[113] K. Kanani, *"A Unified Framework for Systematic Quantitative and Qualitative Analysis of Communicating Systems"*, Ph.D. Thesis, Imperial College (UK), 1998

[114] A. Kiehn, M.C.B. Hennessy, *"On the decidability of non-interleaving process equivalences"*, in Fundamenta Informaticae 30(1):18-33, 1997.

[115] L. Kleinrock, *"Queueing Systems"*, John Wiley & Sons, 1975

[116] K.G. Larsen, P. Pettersson, W. Yi, *"Uppaal in a Nutshell"*, in Int. J STTT 1:134-152, 1997

[117] K.G. Larsen, A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991

[118] K.G. Larsen, A. Skou, *"Compositional Verification of Probabilistic Processes"*, in Proc. of the *3rd Int. Conf. on Concurrency Theory (CONCUR '92)*, LNCS 630:456-471, 1992

[119] N. Lopez, M. Nuñez *"Weak Stochastic Bisimulation for Non-Markovian Processes"*, Submitted for publication, 2002

[120] N.A. Lynch, M.R. Tuttle, *"Hierarchical Correctness Proofs for Distributed Algorithms"*, in Proc. of the *6th ACM Symp. on Principles of Distributed Computing (PODC '87)*, ACM Press, pp. 137-151, Vancouver (Canada), 1987

[121] K. Matthes, *"Zur Theorie der Bedienungsprozesse"*, in Trans. of the *3rd Prague Conf. on Information Theory, Stat. Dec. Fns. and Random Processes*, pp. 513-528, 1962

[122] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[123] R. Milner, *"A Complete Inference System for a Class of Regular Behaviours"*, in Journal of Computer and System Sciences 28:439-466, 1984

[124] R. Milner, *"A Complete Axiomatization for Observational Congruence of Finite-state Behaviours"*, in Information and Computation 81:227-247, 1989

[125] F. Moller, C. Tofts, *"A Temporal Calculus of Communicating Systems"*, in Proc. of the *1st Int. Conf. on Concurrency Theory (CONCUR '90)*, LNCS 458:401-415, 1990

[126] U. Montanari, M. Pistore, *"Minimal Transition Systems for History-Preserving Bisimulation"*, in Proc. of the *14th Symp. on Theoretical Aspects of Computer Science (STACS'97)*, LNCS 1200, 1997

[127] U. Montanari, D. Yankelevich, *"Location Equivalence in Parametric Setting"*, in Theoretical Computer Science 149(2):299-332, 1995

[128] V. Natarajan, I. Christoff, L. Christoff, R. Cleaveland, *"Priorities and Abstraction in Process Algebra"*, in Proc. of *Foundations of Software Technology and Theoretical Computer Science*, LNCS 880:217-230, 1994

[129] M.F. Neuts, *"Matrix-Geometric Solutions in Stochastic Models - An Algorithmic Approach"*, John Hopkins University Press, 1981

[130] V.F. Nicola, *"Lumping in Markov Reward Processes"*, Tech. Rep. RC-14719, IBM T.J. Watson Research Center, Yorktown Heights (NY), 1990

[131] X. Nicollin, J. Sifakis, *"The Algebra of Timed Processes, ATP: Theory and Application"*, in Information and Computation 114:131-178, 1994

[132] X. Nicollin, J. Sifakis, S. Yovine, *"Compiling Real-Time Specifications into Extended Automata"*, in IEEE Trans. on Software Engineering, 18(9):794-804, 1992

[133] X. Nicollin, J. Sifakis, *"An Overview and Synthesis on Timed Process Algebras"*, in Real-Time: Theory in Practice, LNCS 600, 1991

[134] M. Nielsen, P.S. Thiagarajan, *"Degrees of Nondeterminism and Concurrency"*, in Proc. of the *4th Conf. on Foundations of Software Technologies and Theoretical Computer Science*, LNCS 181:89-117, 1984

[135] R. Paige, R.E. Tarjan, *"Three Partition Refinement Algorithms"*, in SIAM Journal of Computing 16:973-989, 1987

[136] D. Park, *"Concurrency and Automata on Infinite Sequences"*, in Proc. of the *5th GI-Conf. on Theoretical Computer Science*, LNCS 104:167-183, 1981

[137] G. Plotkin, *"A Structural Approach to Operational Semantics"*, Tech. Rep. DAIMI FN-19, Aarhus University (Denmark), 1981.

[138] C. Priami, *"Stochastic π-Calculus"*, in Computer Journal 38(6):578-589, 1995

[139] C. Priami, *"Stochastic π-Calculus with General Distributions"*, in *Proc. of the 4th Workshop on Process Algebras and Performance Modelling (PAPM '96)*, CLUT, pp. 41-57, Torino (Italy), 1996

[140] M.A. Qureshi, W.H. Sanders, *"Reward Model Solution Methods with Impulse and Rate Rewards: An Algorithm and Numerical Results"*, in Performance Evaluation 20:413-436, 1994

[141] M. Rettelbach, *"Probabilistic Branching in Markovian Process Algebras"*, in Computer Journal 38:590-599, 1995

[142] A. Rabinovich, B. Trakhtenbrot, *"Behaviour Structures and Nets"*, in Fundamenta Informaticae 11:357-404, 1988.

[143] W.H. Sanders, J.F. Meyer, *"A Unified Approach for Specifying Measures of Performance, Dependability, and Performability"*, in Dependable Computing and Fault Tolerant Systems 4:215-237, 1991

[144] P. Schweitzer, *"Aggregation Methods for Large Markov Chains"*, in Mathematical Computer Performance and Reliability, North Holland, pp. 275-286, 1984

[145] R. Segala, *"Modeling and Verification of Randomized Distributed Real-Time Systems"*, Ph.D. Thesis, MIT, Boston (MA), 1995

[146] M. Shaw, D. Garlan, *"Software Architecture: Perspectives on an Emerging Discipline"*, Prentice Hall, 1996

[147] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994

[148] A.S. Tanenbaum, *"Computer Networks"*, Prentice Hall, 1996

[149] H.M. Taylor, S. Karlin, *"An Introduction to Stochastic Modeling"*, 3rd ed., Academic Press, 1998

[150] C.M.N. Tofts, *"Processes with Probabilities, Priority and Time"*, in Formal Aspects of Computing 6:536-564, 1994

[151] C.A. Vissers, G. Scollo, M. van Sinderen, E. Brinksma, *"Specification Styles in Distributed Systems Design and Verification"*, in Theoretical Computer Science 89:179-206, 1991

[152] W. Vogler, *"Bisimulation and action refinement"*, in Theoretical Computer Science 114:173-200, 1993

[153] W. Vogler, *"Timed Testing of Concurrent Systems"*, in Information and Computation 121:149-171, 1995

[154] S. Yovine, *"Kronos: a Verification Tool for Real-Time Systems"*, in Int. J STTT 1:123-133, 1997

[155] S.-H. Wu, S.A. Smolka, E.W. Stark, *"Composition and Behaviors of Probabilistic I/O Automata"*, in Theoretical Computer Science 176:1-38, 1997