Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: 04/06

Synchronization Strategies for Global Computing Models

Ivan Lanese

Supervisor
Prof. Ugo Montanari

Referee
Dr. Reiko Heckel

Referee
Dr. Barbara König

Referee
Prof. Davide Sangiorgi

July 4, 2006

Addr: Largo B. Pontecorvo 3, I-56127 Pisa – Italy
Tel: +39-050-2213145 — Fax: +39-050-2212726 — E-mail: lanese@di.unipi.it
Web page: http://www.di.unipi.it/~lanese

Abstract

This thesis aims at analyzing different aspects of synchronization in models for Global Computing, concentrating on mobility, heterogeneity and compositionality.

We start by presenting a comparison among three different models from the literature: Fusion Calculus, a calculus with mobility; Synchronized Hyperedge Replacement (SHR), a graph transformation framework that can be equipped with both Milner (i.e., CCS style) and Hoare (i.e., CSP style) synchronization; and Logic Programming, used as goal rewriting mechanism. We show that Milner SHR is essentially a graphical concurrent extension of Fusion Calculus with multi-party synchronization. The comparison between Hoare and Milner SHR shows that the two models have different expressiveness, and that Hoare SHR can be mimicked using Milner SHR, while the contrary is not always possible. The last step of the comparison shows a strict correspondence between Hoare SHR and an extension of Logic Programming with a transactional mechanism and an hiding operator.

The research in the second part of the thesis is triggered by the observation that the choice of the synchronization policy to be used in a modeling framework is crucial, since implementing a synchronization policy on top of another one is not an easy task. Thus we introduce Synchronization Algebras with Mobility (SAMs), an abstract description of a synchronization model obtained by extending Winskel's synchronization algebras to cope with mobility and local resource handling. SAMs are used to make existing frameworks parametric w.r.t. a synchronization model, which can thus be chosen to match the characteristics of the modeled system. Parametric SHR applies this approach to SHR while PRISMA Calculus is the corresponding extension for Fusion Calculus. Furthermore, we introduce SHR for heterogeneous systems, where different SAMs can be used at the same time to manage interactions performed in different parts of an heterogeneous system.

The last part of the thesis concentrates on compositionality properties of the observational semantics of the analyzed frameworks. In particular we choose bisimilarity as semantics and we show whether this is a congruence w.r.t. the operators of system composition. We extend standard bialgebraic techniques to prove that bisimilarity is a congruence for parametric SHR. We also present a concurrent operational semantics for Fusion Calculus and we prove that the corresponding bisimilarity is a congruence, while the same property does not hold for standard interleaving semantics.

Acknowledgments

I want to thank my supervisor Ugo Montanari for having driven this work, in particular at the beginning when I was a bit lost in the world of computer science research. A special thank is for Roberto Bruni, as coworker, coauthor and friend, for his suggestions and for having always found the time to help me. I also want to thank all the other persons I have worked with: Emilio Tuosto, Dan Hirsch, Marzia Buscemi, Hernan Melgratti, ...

Special thanks also for my friends Roberto Zunino and Maurizio Atzori, which, even if they work on very different topics, have been able and willing to give me useful advices. Thanks also to all my friends, for their help and for being my friends.

Last but absolutely not least, thanks to my parents Pino and Claudia, for having always helped me, even if I showed up in my hometown Aosta only two or three times a year.

Very very last, thanks to all the persons that helped me and that I forgot to thank above.

Thanks to all,

Ivan

Contents

List of Figures

List of Tables

Introduction

In this thesis we discuss different issues concerning *formal approaches* to the development of *global computing* (GC) systems and applications. Global computing, as described in Section 0.1, rises new challenges to the software engineering community, since different issues such as distribution, heterogeneity and mobility interact in complex ways. A main aspect is that complex synchronization patterns involving many different components are required to specify how to reach the desired computational goal. We argue in Section 0.2 that formal methods are the right approach to face these challenges. We will discuss some of the approaches in the literature in Section 0.3, while the contributions of this thesis are outlined in the remaining sections.

0.1 Global computing

Global computing, that is computing using networks deployed on world-scale areas, is becoming more and more important. In fact Internet, the most famous and developed GC system, interconnects today hundreds of millions of users, ranging from private users with their home PCs to industrial LANs (local area networks). In addition to Internet, also other WANs (wide area networks), such as wireless communication networks or ATM networks, exist.

GC systems have different characteristics w.r.t. previous kinds of systems such as LANs, and one must keep these differences into account not only when designing them, but also in the more frequent case of developing applications for this environment.

First of all, GC systems are *distributed*, but in addition to normal problems posed by distribution in LANs, such as routing problems or remote resource access problems, new difficulties arise because of the large distances involved. Since the cost of communications (in terms of time, but maybe also in terms of money) is not uniform, applications can not abstract from physical locations: *network awareness* becomes an issue. Thus applications must deal with locations explicitly in order to reach the wanted levels of quality of service (maybe by connecting to a near server instead of a far away one). In some cases, physical locations can also be involved in a more semantic way in the behaviour of an application. A classic example of

location-dependent application is one that provides tourism information, such as restaurant or hotel advertising: it should possibly suggest places near to the current position of the user.

This kind of problems becomes fundamental for mobile systems. The importance of *mobility* nowadays is enormous, since everyone has its cellular phone, and maybe also a mobile computer and a PDA. These examples refer to *physical mobility*. Since one wants that these devices, while moving, keep computing and they remain connected to some network, one wants GC systems to be able to support mobility. This means that locations must be treated as dynamic information, and connectivity can not be taken for granted. Another kind of mobility is *software mobility*, where hardware is fixed but code can change its location. A common example of software mobility is given by Java applets that can be downloaded from a remote server and then executed locally. A more interesting example is given by software agents, which can move from host to host in order to collect some data and then go back home to deliver the result of their computation. The important aspect here is that, not only code, but the whole computation (with its state) is moved. Software mobility can be useful since moving code may be cheaper than moving the data needed by the computation.

Another important feature of GC systems is *heterogeneity*: usually they are composed by hosts which differ in computational power (from mainframes to PDAs), in connectivity (from fast LANs to telephone cables), in memory and resource availability, in operating system and language support and in applicative environment. Thus a main issue in GC is to make heterogeneous systems able to communicate, synchronize and coordinate, abstracting away from the differences and allowing interoperability. In other cases, differences must not be abstracted, but taken into account in order to exploit them, e.g. by connecting to some host where a particular kind of resource is available at the minimum cost. In order to do that, a negotiation between the client and some possible servers may be needed. Another kind of heterogeneity is of administrative nature: systems can be composed by domains which belong to different organizations, thus being subject to different accounting and security policies. In order to interact with or to move to other domains, applications may have to exhibit some authorizations or passwords.

Security is also a direct concern for GC applications, since they can not always trust all the components of the system they communicate with or even their own execution environment (consider, e.g., the case of an agent executing on a remote server). Thus issues such as secrecy of communications, integrity of data and code, and availability of the services must be considered.

Another constraint on most GC systems is that they must be *open*. This means that components can be added to or removed from a system while its other parts continue their executions. A simple, striking example: you can not ask to shut down Internet in order to connect or disconnect your PC. Thus available resources, the parts of a system to be synchronized or the subsystems participating in a computation can change dynamically and without warning. This aspect is even more stressed

in the case of mobile systems which can disconnect and then reconnect, maybe from a different location. Sometimes not only components are added or removed, but the whole structure of the system can be changed, while keeping the system running. This may happen for many different reasons, for instance in order to satisfy new requirements for the system, or simply in order to optimize the structure of the system w.r.t. the current service requests, ensuring load balance. Failures of servers or communication links may trigger reconfigurations too. Mobile systems are continuously reconfigurated in response to the movement of their own components.

Among the features of GC models discussed above, we leave aside security issues and we focus on distribution and mobility, since security is usually analyzed using dedicated techniques. As far as heterogeneity and openness are concerned, even when they do not appear explicitly, they must be kept in mind since they influence the choice of the best approach to use. For instance, we can not use models that impose specific requirements on the structure of components since they are not suitable for modeling heterogeneous systems.

0.2 Formal models for global computing

In order to manage the complexity of the development of applications for such an environment, where different problems exist and interact in complex ways, *formal approaches* are needed. Using formal methods one can build models of the system to be developed, in order to be able to analyze some aspects of it, before starting the actual implementation process. This is useful, first of all, to make clear the requirements and then the structure of the system to be built, concentrating on the aspects of interest and abstracting away from details. Many correctness checks can be performed directly on the model, in order to find mistakes as soon as possible. Correcting an error in the model is, in fact, much easier and much cheaper than detecting and correcting it during the implementation phase. The developed models can be used to guide and help the implementation of the system (tools exist for generating part of the code from models) and then as reference when testing the actual system. For instance, an executable specification of a system can be used to generate the correct behaviours of a system under test, and then the real system, which is more complex since it must handle many more aspects, can be tested against these behaviours.

The use of formal methods is consolidated in computer science, but traditional models, built for sequential or parallel programs, are not enough for dealing with GC systems. In fact, in addition to standard concerns such as efficiency and input-output behaviour, other aspects have to be considered. As an example, GC models usually handle aspects such as locations or probability (for security concerns for instance) explicitly. Furthermore GC systems are usually made of traditional preexisting systems which interact in order to reach a common goal. Thus models must deal

with *coordination* among given entities, which interact using some set of primitives that depends on the underlying middleware. More precisely, models must allow the separation between computation, which can be abstracted, at least at the higher levels of abstraction. and coordination, which is always a central aspect. Even after this first abstraction, GC models can be large, thus further levels of abstraction can be necessary. For instance, at the low level of abstraction, communications are asynchronous (and maybe faulty) as they are in real systems, but, at the more abstract level, one wants to consider (part of) the system as synchronous, leaving to other levels the implementation of synchronization mechanisms. This may require to exploit probabilistic techniques, since in many cases deterministic solutions are not possible [Lyn89]. Furthermore, at the low level, the basic interacting mechanism is usually a synchronization between two entities, that is an abstraction of some kind of message passing, while at the higher levels synchronizations are wider in space, involving multiple entities, and in time, grouped in transactions that must be completed atomically or parts of complex reconfigurations. For instance, different components of the system may have to agree on some information, such as the bandwidth dedicated to a data transfer, and at the low level one is interested in the protocol that drives the negotiation. At the more abstract level components provide their requests, as constraints on the behaviour of the system, and the result of the synchronization is a behaviour that satisfies all the constraints. It is useful to have these complex forms of synchronization as primitives in the model, without the need of simulating them using simpler ones, since such encodings may be complex and may obfuscate the description of the aspects of interest. The relationships between models at different levels of abstraction can be explicitly defined as refinements between them.

Finally, because of the huge size of the systems involved, it is important to have *compositional* models, that is to be able to combine models of different parts of the system to have a model of the whole system. A common approach to this problem is to define systems that can be interconnected through some kind of interfaces, and then to consider as behaviour of the system the one that can be inferred by analyzing what happens on its connecting interfaces. A model is compositional if its external behaviour can be inferred from the external behaviours of its components, without taking into account their internal structure. Different methods for analyzing this kind of behaviours exist in the literature, and one of their common goals is to define when two systems are equivalent, that is when they exhibit the same set of behaviours. In a compositional framework, equivalent subsystems are interchangeable without altering the behaviour of the whole system. Furthermore, if a system is equivalent to a simpler specification (w.r.t. some set of properties) then it correctly satisfies the requirements imposed by it.

The importance of GC systems, and in particular the need for theoretical research in the field, has been recognized by the European community, which started in 2001 the GC initiative [Glob], which has now been followed by GC II [Gloa].

These initiatives consist of a cluster of projects which are devoted to the analysis of different issues raised by GC systems, with particular attention to the foundational aspect.

Our work has been carried on in this framework, and it is mainly related to the GC projects Agile [AGI] on architectures for mobility and Profundis [PRO] on proofs of functionality for mobile distributed systems and to the more recent GC II project Sensoria [SEN] on software engineering for service-oriented overlay computers.

0.3 Overview of models for GC

We give now a general overview of some models and techniques that have been successfully used in the GC area, emphasizing in particular their merits and drawbacks. A more detailed and technical description of the ones that are more closely related to our work is given in Chapter 1.

An important approach is the *process calculi* one: since process calculi are situated between real programming languages and mere mathematical abstractions, they facilitate rigorous system analysis, offering the basis for prototypical implementations and for verification tools. Many running implementations of languages [PT00, CL99, SU01, BDPF98] based on calculi originally proposed to experiment basic interaction primitives exist. In the process calculi approach a system is modeled as a term in a suitable term algebra, which has operators such as parallel composition, nondeterministic choice and local resource declaration. The operational semantics of a system is defined by inference rules (SOS semantics, see [Plo81, Plo04]) that define a labeled transition system (LTS) or via reduction rules. In the second case a relation specifies which states are reachable from a given one, while in the first one also a label specifying the interactions with the environment is added. SOS semantics is quite important since it allows to easily define observational equivalences. These equivalences quotient systems which are indistinguishable by observing their interactions with the environment, even if they have a different internal structure. The first and simpler equivalence is trace equivalence [Hoa80], which just considers the sequence of labels of the steps in the computation. A more refined approach is bisimulation [Par81, Mil89], which considers also the branching structure of the computation. Testing equivalences [DH84] concentrate instead on the ability of processes to produce a particular observation when inserted in some contexts. Another approach, which started with [HM85], is based on the set of formulas of a suitable modal logic that are satisfied by the process. An interesting recent thread of research is devoted to find automatic ways to build a labeled semantics starting from a reduction one [Sew98, LM00]. An important drawback of process calculi is that they use the same syntactic structures to define both the topological structure of the system and its allowed behaviours. For instance, the parallel composition operator can be considered both as parallel composition of systems and as defining

the concurrent execution of two processes. Similarly, a restriction operator can be interpreted as definition of an administrative domain, but also as name declaration.

The process calculi approach is born in the area of communicating systems with CCS [Mil82] and CSP [Hoa80], which allow to define systems interacting using respectively *Milner synchronization*, where two processes synchronize by doing complementary actions, simulating message passing systems, and *Hoare synchronization*, where any number of processes can synchronize by performing the same action. ACP [BK84] instead has a parametric synchronization model and a more developed algebraic theory. These calculi are not apt to model GC systems since they can not handle, e.g., mobility. This problem is analyzed in *π-calculus* [MPW92], a successor of CCS, which introduces name mobility: instead of moving a process, a name, i.e. a reference to it, is transmitted. This kind of mobility is mathematically simpler but it has been shown powerful enough to model also process mobility [San01], where the actual process is communicated. An important feature of π-calculus is its ability to generate new names which are guaranteed distinct from any other known name. This allows to give access to a resource to exactly one process, that is the one that receives the name, and to model security features such as nonces. An asynchronous version of π-calculus has been proposed in [HT91].

The π-calculus is the first calculus of a whole family of calculi for mobility (see [Dal00] for a nice survey). Among these we are mainly interested in the *Fusion Calculus* [PV98, Vic98], which is a convenient simplification of π-calculus. Nevertheless π-calculus can be mapped into a subset of Fusion Calculus preserving the one-step semantics [Vic98]. Thus Fusion Calculus is, from this point of view, at least as expressive as its predecessor, and furthermore it allows fusions of names, thus modeling a further mobility feature. However, since fusions of arbitrary names can be performed, names can no more be guaranteed distinct. An approach that enforces this property back is the one presented in [BBM04, BBM05]. An asynchronous version of Fusion Calculus is presented in [GW00].

The Join Calculus [FG96] is similar to the asynchronous π-calculus but it enforces a unique site for each channel, making it more easily implementable in a distributed setting (since no conflicts among receivers of a message are possible).

Other variants of π-calculus have been proposed to deal directly with specific issues, such as locations [HR98], probability [Pri95, HP00], broadcast communications [EM99], cryptography [AG97], terms [AF01] and semistructured data [AB05].

Another important calculus is *Ambient calculus* [CG98], which is more suited to study hierarchical models where the mobility space is partitioned in different environments called ambients. Processes live inside ambients and they can drive their ambients with primitives for moving or destroying an ambient. A known problem of Ambient calculus is that it has an artificious and involved SOS semantics [MZ03], while reduction semantics is simpler.

An interesting calculus is KLAIM [DFP98], a process calculus based on LINDA [Gel85], which thus uses (distributed) repositories called tuple spaces as communication infrastructure. This communication paradigm allows asynchronous interactions

among processes, which can drop pieces of information in tuple spaces. Other processes can come and read those pieces of information, and searching is done via pattern matching, thus the sender and the receiver do not need to know each other. KLAIM is important since it contains also primitives for modeling software mobility, thus allowing easy creation of systems composed by mobile agents. Furthermore KLAIM is not only a process calculus, but it has also a prototypical implementation, and thus it can be used to experiment with real mobile applications [BDPF98].

An approach that is quite related to the process calculi one is the coalgebraic one [Rut00]: a *coalgebra* in a suitable category[1] can be used, e.g., to model a LTS, and the notion of bisimulation naturally arises [AM89] as the dual notion to that of algebraic congruence. This kind of models is well-suited for studying the observational properties of systems and in particular their compositionality aspects. With this aim, one is interested in combining the observational behaviour of the system, i.e. its coalgebraic structure, with its compositional structure, i.e. its algebraic structure. In particular, one wants that the coalgebraic notion of bisimulation has the algebraic property of being a congruence w.r.t. the operators in the algebra. This happens in bialgebras [PT97]. Some techniques for proving that a coalgebra has indeed a bialgebraic structure will be presented in Section 1.2.

An extension of this approach is the so called *Tile Model* [GM00], which is also inspired by rewriting logic [Mes92] and context systems [LX90]. In the Tile Model both the system and the observations are modeled as arrows in a category, and basic blocks called tiles represent the evolution of open system components. In particular, a tile specifies how a given state is rewritten into another one, which are the triggers required from subsystems to make the evolution possible and which is the observable behaviour of the step. Basic tiles can be composed in different ways in order to model interactions of different parts of the system or sequences of computational steps. The Tile Model has been frequently used as a metamodel for studying process calculi, see e.g. [FM00, MT97, GM02].

A classic approach to concurrent systems is the *Petri net* one [Pet62, Rei85]. In this approach, a system is represented by a network of places and transitions. Places hold tokens, which represent the state of the system, and transition executions can consume tokens from some places producing tokens in other places. Many transitions can be executed simultaneously. Interestingly, both states and computations have a monoidal structure [MM90]. Some steps in the Petri net field towards GC have been accomplished with the introduction of zero-safe nets [BM00], which allow to synchronize different transitions in a transactional style. Furthermore this model can be seen as a refinement of the classical one. The (quite developed) theory of Petri Nets can be generalized to other approaches more apt to the GC field, in particular to *graph transformations* [Roz97, EKMR99].

Graphs are a natural model for systems made up of interconnected components,

[1] An introduction to category theory can be found in [AHS90].

since the topological structure can be directly mapped into the structure of the graph. We mainly consider the approach where edges, or in general hyperedges, namely edges connecting any number of nodes, represent processes or hosts which communicate through ports or links represented by shared nodes, but dual approaches also exist [Le 98]. Graph transformations can animate the graph, thus allowing to model a system whose configuration changes dynamically. In general, a graph transformation system is made up of rules that specify how a particular graph (the LHS of the rule) can be rewritten into another one (the RHS of the rule). The approaches differ mainly in the constraints on the structure of the LHS (one node, one edge, a generic graph), in the method used to match the LHS with a subgraph of the graph to be rewritten and on the gluing strategy used to merge the RHS of the rule with the rest of the graph. Sometimes special notation is provided to express the preservation of a part of the LHS in the RHS.

The algebraic approach (so called because graphs are seen as elements in particular algebras), which allows to rewrite generic graphs, started with the double pushout definition [EPS73], which uses two pushout constructions in the category of graphs and total graph morphisms to apply rules. More recently a single pushout in the category of graphs and partial graph morphisms has been used to the same purpose giving rise to the single pushout approach [Löw93]. An approach based on a double pullback [CEHW01] instead is used to specify minimal requirements on the transformation, allowing also additional unspecified reconfigurations to be performed at the same time. The use of categorical constructions to define graph transformations has the advantage of being largely independent on the exact structure of the objects to be rewritten, thus it is easily generalized to more complex situations such as higher-level graphs [EHKP91]. These approaches to graph transformations are difficult to implement in a GC setting where graphs are distributed systems. In fact, when writing applications to implement reconfigurations modeled as graph transformations of this kind, we may discover that hosts have not enough information about the topology of the whole system to decide if a rule needs to be applied or not. In general, the approach to coordination called choreography does not require such information, while orchestration supposes a global view of the system [BGG+05].

Thus a more choreographic approach, called *Synchronized Hyperedge Replacement* (SHR), has been proposed. SHR uses context free rules called productions to describe the evolution of single hyperedges (that is single processes or hosts). Process synchronizations are modeled by imposing conditions on nodes, which specify constraints on which productions can be applied in a single step. When the productions to be applied have been chosen (this is called the rule-matching problem) in such a way that the conditions are satisfied, the actual rewriting can be carried on separately by single components. This approach was first proposed in [CM83] with the name of "Grammars for Distributed Systems". The rule-matching problem can be modeled as a finite domain constraint problem [CDM85] and solved through a distributed algorithm. Infinite computations and concurrency issues have been

considered in [DM87]. An interesting extension of this approach allows also communication of nodes during transitions [RM99, HM01], and this allows to model mobile systems whose connections change at runtime. An extended description of SHR is presented in Section 1.5.

A recent interesting approach to graph transformations is the one of bigraphs [Mil01, JM03], where two graphical structures are considered together: a forest representing the spatial relation of containment, and a generic graph representing the connections between entities.

A well known computational model that can also be used fruitfully in the GC area is *Logic Programming* [Llo93]. To this end however one must abandon the usual idea of Logic Programming, where the aim is to find a SLD-refutation for a goal, and thus a substitution that makes the corresponding clause true. It is useful instead to consider logic predicates as concurrent processes that interact through shared variables using the unification engine as communication mechanism. In this framework one must consider partial or even infinite computations. The first step in this direction is the work on compositionality in Logic Programming (see e.g. [GS89]). An interesting paper for the GC perspective is [BMR01] where observational semantics is analyzed and open systems are considered (i.e. the semantics can deal with instantiation of variables and insertion of the goal into a larger context).

A modeling framework that is widely used both in industry and in academia is UML [RJB99], which provides a syntactic way of describing component-based systems. The main drawback of UML is that it has not a complete formal semantics, thus it is difficult to make formal analysis on UML diagrams. Extensions of UML to deal with GC features such as mobility have been considered, e.g., in [BKKW02].

This overview shows just the top of the iceberg of GC models, but it is enough to highlight that different approaches exist, with different merits and drawbacks. Models developed to deal more easily with specific issues coexist with general frameworks aiming at reunification. It is clear that even if a model able to deal with all the issues raised by GC systems could exist, it is still far away. Our aim is, however, to take some steps towards a reunification or at least a categorization of these models.

0.4 Contributions of the thesis

As already said, we have tried to move some steps towards the definition of a model for GC apt to describe complex coordination patterns among different heterogeneous, distributed, mobile entities, analyzed at the high level of abstraction.

A good intuition of the idea underlying our approach can be given by a physical analogy. In physics, one is usually interested in systems where multiple objects interact, for instance fluid molecules in a tube or antennas transmitting in the sky. One can use physical equations to analyze all the interactions between the components, but usually this approach is by far too complex to be carried on. Thus one considers small parts of the space and (s)he describes each of them by using partial

derivative equations. When suitable boundary conditions are added, one gets (if (s)he is able to solve the system of equations) a solution that describes the possible evolutions of the system.

In a similar way, we want to describe the interactions among GC components by considering the interfaces as the interesting "small parts of the space" described by equations specifying the conditions on system interactions, with boundary conditions specified by components requirements. Solving these equations gives the allowed behaviours of the system.

Essentially any synchronous form of interaction can be seen in this perspective, but until now this aspect has not been exploited in its full power. In fact, the great majority of models in the literature use very simple equations, corresponding to message passing (Milner synchronization) or to other simple patterns. We think that these models are not enough for modeling at the right level of abstraction the complex kinds of interactions we are interested in.

From a technical point of view, we have started by analyzing different existing models from the literature, with a double aim. First of all we wanted to better understand their relationships to find out which features they have in common, which features existing in some of them are instead missing in others (and what we could get by adding them), and which correspond to choices depending on the specific application one has in mind. We think that this last case individuates a dimension in a "space of GC models", and that this allows to give a categorization of GC models, increasing their understanding and allowing to build parametric models where the choices made on different dimensions have been made orthogonal.

Since a comparison among all the different models was far beyond our possibilities, we have focused on some of them that showed at a first sight some interesting commonalities, namely Fusion Calculus, Synchronized Hyperedge Replacement (SHR) and Logic Programming. As hoped, the comparisons suggested many useful insights of the kind described above. By transferring features from one model to the others we obtained some interesting "hybrid models". Among others, we have analyzed the extension of Logic Programming with an hiding operator taken from process calculi, and we have found a semantics for Fusion Calculus with good compositionality properties defined using a mapping into SHR.

The most important improvement however has been triggered by the comparison between Hoare and Milner synchronizations performed in the SHR framework: this has shown that implementing a synchronization model using another one is not a trivial issue, and that one wants to be able to freely choose the synchronization model to use, independently from other features of the modeling framework. In other words, this has exactly shown to be a dimension in the space of GC models. Mathematically, the synchronization policy has been modeled using Synchronization Algebras with Mobility (SAMs), that extend synchronization algebras (which are introduced in [Win84], and recalled here in Section 1.3) to deal with name mobility. Identifying the synchronization model as a first-class entity in designing models has allowed for many interesting results: for instance, SAMs have been proved to form a

category, and thus suitable categorical constructions can be used to compose SAMs in different ways to define easily new complex models, able to give the wanted kind of high-level abstraction.

After having modeled different synchronization policies as SAMs, we have analyzed their applicability in different frameworks. The first application has been inside SHR, since this framework already had two different formulations, one using Milner synchronization and the other one using Hoare synchronization. We propose parametric SHR, where synchronization among productions is specified by a SAM, as a general framework that allows both to recover the two previous instances of SHR and to easily generate new ones, such as broadcast or priority SHR. Having one parametric framework instead of many separated ones has the advantage that properties need to be proven just once, and a unique implementation is possible, allowing the users to choose each time the synchronization model more suitable for the planned application.

A similar approach has been considered also in the field of process calculi, in order to study a framework more apt to analyze the linguistic aspect of interaction. In fact, process calculi can be easily extended with different features, while the structure of SHR is more constrained since different aspects are specified in different ways (e.g., the structure of the system in the graph and its behaviour in the productions). On the other side however SHR provides a cleaner framework, useful mainly for architectural modeling. The choice of which process calculus to extend ended with the selection of Fusion Calculus, for three main reasons. First of all the properties of Fusion and in particular the mapping from π-calculus into Fusion guarantee that it is expressive enough to deal with the main features of GC systems and with mobility in particular. Also, Fusion Calculus has been proved strongly related to SHR, and this gives many insights on how to extend it. Also, this means that moving from SHR to Fusion Calculus corresponds to move just in one dimension in the space of GC models, namely the (really important, and maybe composed by different, yet to discover, subdimensions) dimension that deals with the tools used to represent system structure and behaviour. In particular, one dimension which is kept fixed is the one of name handling, since our comparison proved that both models use the same approach. Finally, Fusion Calculus has a symmetric form of communication, which is easier to extend to models with more than two interacting participants than the asymmetric communication of π-calculus. This extension produced the PRISMA Calculus.

A further step has been to consider the contemporary presence of different synchronization models inside the same system. Notice that such a development is possible only after having isolated and abstractly defined what a synchronization model is (in our approach, a SAM). We have considered this extension in the framework of SHR, and called it SHR for heterogeneous systems, stressing the fact that it allows to model systems where different forms of synchronization coexist. Such a feature is useful for modeling, e.g., a network where wireless broadcast communications are available, together with message passing between wired components.

Also, our framework allows to change the communication policy at runtime. While it looks hard to imagine this option exploited at the communication level, where communication mechanisms are determined by hardware choices, it is a natural feature at the application level, where the details of the communication mechanism are decided by a negotiation among the participants.

The final step in our work has been to consider the compositionality properties of the analyzed frameworks, since each model aiming at representing GC systems must allow compositional reasoning to allow scalability up to systems of the required size. Technically, we have chosen one of the most widely used approaches to the analysis of the observable behaviours of systems, that is bisimilarity. In this framework, the semantics is compositional when bisimilarity is a congruence w.r.t. the different operators for system composition. The first intuitions on this topic have been obtained by analyzing possible mappings of SHR into the Tile Model, where standard techniques exploit the structure of the framework to provide easy proofs of those results. However, all the results have been restated and fully analyzed in general coalgebraic terms to make them accessible to a larger audience. One main result is that bisimilarity is a congruence for parametric SHR with a large class of synchronization models (and, in particular, for Milner and Hoare SHR from the literature). The connection between Fusion Calculus and SHR could suggest that bisimilarity is a congruence also for Fusion Calculus, but, as known from the literature, this is not the case. In fact, to get a compositional semantics for SHR, one must use hyperbisimilarity, i.e., substitution closed bisimilarity. The comparison with SHR showed that this difference is due to the interleaving semantics usually given to Fusion, while the semantics of SHR is inherently concurrent. We have thus been able to produce a concurrent semantics for Fusion Calculus, which is indeed a congruence. Also, the property of Fusion Calculus that hyperbisimilarity is a congruence has been proved to extend to PRISMA Calculus on any SAM.

0.5 Outline of the thesis

The main contributions of this thesis are related in different ways, and these relations can not be well represented by the linear structure of chapters needed in a thesis, thus, after a description of the content of the thesis in order of appearance, we will show a graph representing the relationships between the different topics.

Chapter 1 introduces some technical concepts necessary in order to understand the following chapters. In particular:

- Section 1.1 introduces the basic mathematical notations and the terminology used throughout the thesis.
- Section 1.2 introduces labeled transition systems, bisimilarity and the bialgebraic techniques used to prove that bisimilarity is a congruence. Even if a large part of this material will be used only in Part III we will introduce it first

since, e.g., the concept of labeled transition system will be used throughout the whole thesis, included the following part of background.

- Section 1.3 presents Winskel's synchronization algebras [Win84].

- Section 1.4 presents the syntax and the semantics of Fusion Calculus, following the approach of [Vic98].

- Section 1.5 introduces the representation of graphs as syntactic judgements and the details of Synchronized Hyperedge Replacement (SHR), using both Hoare and Milner synchronization models. Since SHR is central in our work, and since the existing literature proposes different variations and applications of it, we give also a short overview of the different forms in which it has been presented.

- Section 1.6 gives a rapid overview of Logic Programming. Since this framework is known worldwide we concentrate our attention on the particular use of Logic Programming needed for modeling GC systems.

The original contributions of the thesis are organized in three different parts. Part I presents the comparison between Fusion Calculus, SHR with both Hoare and Milner synchronization models, and Logic Programming. In particular:

- Chapter 2 concentrates on the first part of the comparison, between Fusion Calculus and SHR. The relation is explicitly stated in Section 2.1. Section 2.2 introduces the inference rules for Interleaving Milner SHR, a subset of Milner SHR which has an interleaving semantics and which thus corresponds to the image of Fusion Calculus along the mapping. Finally, Section 2.3 presents a concurrent semantics of Fusion Calculus obtained via the mapping as a sample application.

- Chapter 3 presents a detailed comparison of the expressive power of Hoare and Milner synchronization models in the framework of SHR. In particular, Section 3.1 introduces the used concept of expressiveness. Some results proving the different expressive power of the two models are presented in Section 3.2. Section 3.3 continues the comparison in a more restricted setting, where there is no interface toward the environment and each node is attached to exactly two hyperedges, showing that here Hoare synchronization can be easily simulated using Milner synchronization, and introducing a suitable translation for graphs to have a simulation also in the opposite direction. Section 3.4 goes back to graphs with any number of connections (but again without interface) and it presents some simulations between the two synchronization models exploiting the above mentioned translation for graphs.

- Chapter 4 concentrates on the last part of the mapping, from Hoare SHR into Logic Programming. In particular, it introduces Synchronized Logic Programming (SLP) (Section 4.1), a variation of Logic Programming exploiting a transactional mechanism to synchronize Logic Programming steps, and Logic Programming with hiding (Section 4.2), an extension of Logic Programming with an operator of variable hiding taken from SHR. Section 4.3 exploits the fact that the two extensions are orthogonal to merge them and to present a mapping from Hoare SHR into SLP with hiding.

Part II presents frameworks that are parametric w.r.t. the concepts of synchronization model and mobility patterns, thanks to the use of Synchronization Algebras with Mobility (SAMs). In particular:

- Chapter 5 introduces SAMs (Section 5.1) and it presents the formalization of commonly used synchronization policies as sample SAMs (Section 5.2). Also, Section 5.3 defines a category of SAMs and it shows how to compose simple SAMs to define more complex ones using categorical constructions.

- Chapter 6 presents some applications of SAMs in the field of SHR. Section 6.1 introduces parametric SHR, a general SHR framework which can be instantiated with a SAM that defines the interaction model used. SHR for heterogeneous systems (Section 6.2) generalizes the above approach by allowing different SAMs to be used at once, one for each node.

- Chapter 7 introduces the PRISMA Calculus (Section 7.1), a generalization of Fusion Calculus with a parametric synchronization obtained using SAMs, as done for SHR by parametric SHR. The relations of PRISMA Calculus with Fusion Calculus, π-calculus and bπ-calculus are analyzed in Section 7.2.

Part III analyzes the observational semantics of the different frameworks presented and in particular it looks for compositional semantics where the chosen observational equivalence (bisimilarity or hyperbisimilarity) is a congruence w.r.t. the different operators for system composition. In particular:

- Chapter 8 generalizes and applies the theory developed in Section 1.2 to parametric SHR. The basic definitions have to be restated in the bialgebraic framework (Section 8.1) in order to prove that the defined notion of bisimilarity is a congruence for a large class of SAMs (Section 8.2).

- Chapter 9 analyzes instead the observational properties of Fusion Calculus (Section 9.1) and of PRISMA Calculus (Section 9.2). For Fusion Calculus, we prove that the bisimilarity relation derived from the concurrent semantics of Section 2.3 is a congruence, while this is not the case for standard semantics. For PRISMA we prove that hyperbisimilarity is a congruence for any SAM, and we relate PRISMA processes on different SAMs.

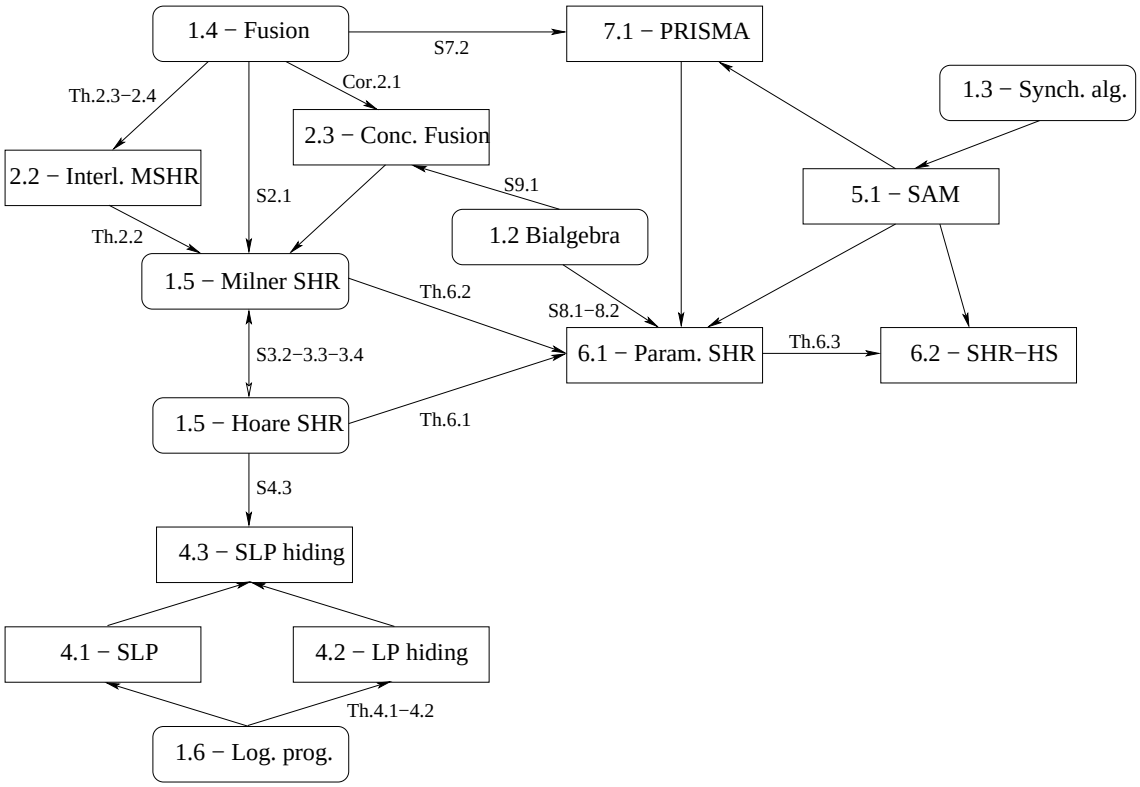

Figure 1: Graphical schema of the thesis.

Finally, Chapter 10 summarizes the obtained results (Section 10.1) and it describes some lines for future work, together with some ideas on the expected problems and results (Section 10.2).

As promised, we conclude this section with a graph summarizing the connections between the different topics presented in the thesis (Figure 1). Background topics have rounded corners, while original contributions are drawn as rectangles. The arrows are used to denote both formal mappings (and in this case they are labeled with a reference to the corresponding result), or just general correlation.

0.6 Origins of chapters

Many results of this thesis have been already presented in some conferences and some of them have been already published in a preliminary form. Although some improvements have been done w.r.t. published material, and many concepts have been extended or refined, the basic ideas behind the results remain the same. Hence, we give here a list of pointers to published papers, together with a short description of their contents. We recall that the whole work has been carried on inside the European GC project Agile [AGI] on architectures for mobility.

- The ideas of Synchronized Logic Programming, Logic Programming with hiding and the mapping from Hoare SHR into SLP with hiding were already present in our Master's Thesis [Lan02].

- The mapping from Hoare SHR (without restriction) into SLP, together with an application to the Ambient Calculus (not reported here, since done before the beginning of our Ph.D.), has been published as an invited talk of Ugo Montanari titled "Software Architectures, Global Computing and Graph Transformation via Logic Programming" [LM02] in the proceedings of the 16th Brazilian Symposium on Software Engineering.

- The interleaving SHR and the mapping from Fusion Calculus into interleaving SHR have been published as "A Graphical Fusion Calculus" [LM04a] in the ENTCS proceedings of the Final Workshop of the COMETA Project on Computational Metamodels.

- The whole chain of mappings (skipping the intermediate step of Milner SHR and without restriction) has been published as "Mapping Fusion and Synchronized Hyperedge Replacement into Logic Programming" [LM04b] in a special issue of Theory and Practice of Logic Programming on Multiparadigm Languages and Constraint Programming (still to appear).

- A less technical but probably more suggestive presentation of the whole chain of mappings (including Milner SHR and restriction) has been published as "Insights emerged while comparing three models for global computing" [LM05b] in the electronic proceedings of Dagstuhl Seminar n. 05081, on Foundations of Global Computing (still to appear).

- Synchronization Algebras with Mobility and their applications to parametric SHR have been published as "Synchronization Algebras with Mobility for Graph Transformations" [LM04c] in the ENTCS proceedings of the 3rd EATCS Workshop on the Foundations of Global Computing.

- SHR for heterogeneous systems has been published as "Synchronized Hyperedge Replacement for Heterogeneous Systems" [LT05] in the LNCS proceedings of COORDINATION'05, Seventh International Conference on Coordination Models and Languages.

- The comparison between Hoare and Milner synchronization in the SHR framework has been published as "Hoare vs Milner: Comparing Synchronizations in a Graphical Framework with Mobility" [LM05a] in the ENTCS proceedings of GT-VC'05, Workshop on Graph Transformation for Verification and Concurrency (still to appear).

We give now a very short overview on some works that have been part of our Ph.D. work but have not been included in the thesis since their topic is a bit outside our main line of research.

- Paper "On Graph(ic) Encodings" [BL04], published in the electronic proceedings of Dagstuhl Seminar n.04241, on Graph Transformations and Process Algebras for Modeling Distributed and Mobile Systems presents a summary of different encoding techniques for mapping process calculi and distributed formalisms to graphic frameworks.

- Paper "New Insights on Architectural Connectors" [BFL+04], published in the Kluwer proceedings of IFIP TCS'04, 3rd IFIP International Conference on Theoretical Computer Science presents a translation of CommUnity [FM97] (an architectural specification language with a categorical semantics) into the Tile Model, together with insights on the comparison between categorical and algebraic approaches to system compositionality.

- Paper "Complete Axioms for Stateless Connectors" [BLM05], published in the LNCS proceedings of CALCO'05, First Conference on Algebra and Coalgebra in Computer Science presents an algebra of connectors for enforcing synchronization and mutual exclusion between different components, together with a complete axiomatization for its bisimilarity.

Chapter 1

Technical background

1.1 Notation, symbols and terminology

This section summarizes the notation that will be used through the different parts of this thesis. More specific notation will be introduced in each chapter, when needed.

Basic mathematical notation. We use the usual notation for sets and operations over sets, with $\cup$ for set union, $\cap$ for set intersection, $\setminus$ for set difference and $\emptyset$ for the empty set. The cartesian product is $\times$, and it has higher priority w.r.t. the other set operators. S^n is the cartesian product of n copies of set S. Furthermore, $|S|$ is the cardinality of S, and $\wp(S)$ is its powerset. $\mathbb{N}$ is the set of natural numbers.

We write a vector $\vec{x}$ as $\langle x_1, \ldots, x_n \rangle$, and its length n as $|\vec{x}|$. The i-th element x_i of vector $\vec{x}$ is denoted as $\vec{x}[i]$. Given a set S we denote with $\mathrm{vect}(S)$ a vector containing the elements in S (in any order).

We denote a (possibly partial) function f from A to B as $f : A \to B$. We use id to denote an identity function. Moreover we define:

$$\mathrm{dom}(f) = \{a \in A | f(a) \text{ is defined}\}$$
$$\mathrm{Im}(f) = \{b \in B | \exists a \in A. f(a) = b\}$$

We denote function composition with $\circ$, i.e., $(f \circ g)(x) = f(g(x))$. Also, we denote with $f \restriction_S$ (resp. $f \restriction_{\setminus S}$) the function f' such that $f'(x) = f(x)$ if $x \in S$ (resp. $x \in \mathrm{dom}(f) \setminus S$), and f' is undefined otherwise. We use set operators also on functions, referring to their representation as sets of pairs.

We use $\uplus$ to denote disjoint set union, with inj_1 and inj_2 as the left and right inclusions, respectively. When no confusion can arise we write $\mathrm{inj}_i(x)$ simply as x. If $\mathrm{inj}_i(x) \in A \uplus B$ we denote with $comp(x)$ the element $\mathrm{inj}_{3-i}(x)$ in $B \uplus A$. Given two functions $f : A \to C$ and $g : B \to D$ we denote with $[f, g] : A \uplus B \to C \uplus D$ the function that applies f to the elements in A and g to the ones in B.

Given an element x in set S and an equivalence relation E on S we denote with $[x]_E$ the equivalence class of x modulo E.

Finally, S^* is the set of strings on alphabet S.

Substitutions. Given a (possibly infinite) set of names $\mathcal{N}$ and a signature Σ, let $T_{\Sigma,\mathcal{N}}$ be the set of terms built with variables in $\mathcal{N}$ and functions in Σ. We omit $\mathcal{N}$ when empty. A substitution on $\mathcal{N}$ is a function $\sigma : \mathcal{N} \to T_{\Sigma,\mathcal{N}}$ such that $\{a \in \mathcal{N} | \sigma(a) \neq a\}$ is finite, whereas for the other names σ behaves as the identity. We will use postfix notation $a\sigma$ for substitution application. Also, given two substitutions σ and σ', $\sigma\sigma'$ is the substitution that applies σ first, and then σ'. We denote with $\{b_1/a_1, \ldots, b_n/a_n\}$ the substitution σ that concurrently replaces all instances of a_i with b_i for each $i \in \{1, \ldots, n\}$, and it is the identity otherwise. We denote with $\mathrm{eqn}(\sigma)$ the set of equations $\{b_1 = a_1, \ldots, b_n = a_n\}$. Domain and image of σ contain only the elements a_i and b_i respectively with $a_i \neq b_i$. A renaming is a substitution σ such that $\mathrm{Im}(\sigma) \subseteq \mathcal{N}$. We allow to apply a substitution to any syntactic structure containing names (e.g., terms, sets, ...), with the intended meaning that the substitution is applied to each name in the syntactic structure. Given a set of equations E, we denote with $\mathrm{mgu}(E)$ any idempotent substitution σ which is a relevant most general unifier of E, if it exists (see [Llo93] for more details).

Names and binders. Given any syntactic structure S, we denote with $\mathrm{n}(S)$ the set of names occurring in S. Furthermore, if the structure contains binders, we denote with $\mathrm{fn}(S)$ the set of its free names (i.e., names which are not bound), while $\mathrm{bn}(S)$ is the set of bound names. Note that only $\mathrm{fn}(S)$ is meaningful if the structure is considered up to α-conversion (i.e., renaming of bound names). The structure S is closed iff $\mathrm{fn}(S) = \emptyset$. Also, when a substitution is applied to a structure defined up to α-conversion, capture-avoiding substitution is used, that is bound names that appear in the substitution are α-converted to new names before the substitution is applied.

1.2 Labeled transition systems and techniques for compositionality proofs

We present here the basic notions of labeled transition system and bisimilarity, together with some bialgebraic techniques to prove that bisimilarity is a congruence.

A labeled transition system is a formalization of the behaviour of a system with state.

Definition 1.1 (Labeled transition systems).
A labeled transition system *(LTS) over a set of states S and a set of labels L is a triple $lts = \langle S, L, \to \rangle$ where $\to \subseteq S \times L \times S$ is a labeled transition relation.*

Notation. *For $\langle p, l, q \rangle \in \to$ we write $p \xrightarrow{l} q$.*

Intuitively, $p \xrightarrow{l} q$ means that the system in state p can evolve to state q by performing an action l.

We are mainly interested in LTSs whose states have an algebraic structure.

We recall that an algebra A over a signature Σ (Σ-algebra in brief) is defined by a carrier set $|A|$ and, for each operation $op \in \Sigma$ of arity n, by a function $op^A : |A|^n \to |A|$. The term algebra T_Σ on the signature Σ has as carrier sets the sets of ground terms (i.e., terms without variables) built using the operators in Σ, and operators op mapping the tuple of terms $t_1, \ldots, t_n$ into the syntactic term $op(t_1, \ldots, t_n)$. Given a set of axioms E, the term algebra $T_{(\Sigma,E)}$ on Σ and E has as carrier sets the sets of equivalence classes $[t]_E$ of terms modulo the smallest congruence including axioms in E, and functions defined by $op([t_1]_E, \ldots, [t_n]_E) = [op(t_1, \ldots, t_n)]_E$.

A morphism between two Σ-algebras (called Σ-morphism) A and B is a function $h : |A| \to |B|$ that commutes with all the operators in Σ, namely for each operator $op \in \Sigma$ of arity n we have $op^B(h(a_1), \ldots, h(a_n)) = h(op^A(a_1, \ldots, a_n))$.

Notation. *We denote with $\mathcal{Alg}(\Sigma)$ the category of Σ-algebras and Σ-morphisms (see [AHS90] for an introduction to category theory). If A is an algebra, we write $lts = \langle A, L, \to \rangle$ to denote the LTS with states in $|A|$, labels in L and labeled transition relation $\to$.*

We now introduce bisimilarity, a notion of observational equivalence between LTSs that refines trace equivalence by keeping into account also the branching structure of the allowed computations.

Definition 1.2 (Bisimilarity). *Let $lts = \langle S, L, \to \rangle$ be a LTS. A relation $\mathcal{R}$ over S is a* bisimulation *iff $p \mathcal{R} q$ implies:*

- *for each $p \xrightarrow{l} p'$ there is some $q \xrightarrow{l} q'$ such that $p' \mathcal{R} q'$;*
- *for each $q \xrightarrow{l} q'$ there is some $p \xrightarrow{l} p'$ such that $p' \mathcal{R} q'$.*

Bisimilarity $\approx$ *is the largest bisimulation.*

More refined notions of bisimilarity, apt to deal with specific features such as name mobility, will be introduced later.

A collection of SOS rules [Plo81] can be regarded as a specification of those LTSs that have a transition relation closed under the given rules.

Definition 1.3 (SOS rules). *Given a signature Σ and a set of labels L, a* sequent *$p \xrightarrow{l} q$ is a triple where $l \in L$ and p, q are Σ-terms with variables in a given set X. An* SOS rule *r takes the form:*

$$\frac{p_1 \xrightarrow{l_1} q_1 \ \cdots \ p_n \xrightarrow{l_n} q_n}{p \xrightarrow{l} q}$$

where $p_i \xrightarrow{l_i} q_i$ *as well as* $p \xrightarrow{l} q$ *are sequents.*

We say that a LTS $lts = \langle A, L, \rightarrow \rangle$ satisfies *a rule* r *like above if each assignment to the variables in* X *that is a solution*[1] *to* $p_i \xrightarrow{l_i} q_i$ *for each* $i \in \{1, \ldots, n\}$ *is also a solution to* $p \xrightarrow{l} q$.

Definition 1.4 (Transition specifications). *A* transition specification *is a tuple* $\Delta = \langle \Sigma, L, R \rangle$ *consisting of a signature* Σ*, a set of labels* L *and a set of SOS rules* R *over* Σ *and* L*. A LTS over* Δ *is a LTS over* Σ *and* L *that satisfies rules* R.

It is well known that LTSs can be represented as coalgebras for a suitable functor [Rut00].

Definition 1.5 (Coalgebras). *Let* $F : \mathcal{C} \rightarrow \mathcal{C}$ *be a functor on a category* $\mathcal{C}$*. A* coalgebra *for* F*, or* F*-coalgebra, is a pair* $\langle A, f \rangle$ *where* A *is an object of* $\mathcal{C}$ *and* $f : A \rightarrow F(A)$ *is an arrow of* $\mathcal{C}$*. A* F*-morphism* $h : \langle A, f \rangle \rightarrow \langle B, g \rangle$ *is an arrow* $h : A \rightarrow B$ *of* $\mathcal{C}$ *such that* $h; g = f; F(h)$.

Notation. *We denote as* $\mathcal{C}\mathfrak{oalg}(F)$ *the category of* F*-coalgebras and* F*-morphisms. We denote with* $\mathcal{SET}$ *the category of sets and functions.*

Proposition 1.1. *For a fixed set of labels* L*, let* $P_L : \mathcal{SET} \rightarrow \mathcal{SET}$ *be the functor defined on objects as* $P_L(X) = \mathcal{P}(L \times X \cup X)$*, where* $\mathcal{P}$ *denotes the countable powerset functor, and on arrows as* $P_L(h)(S) = \{\langle l, h(p) \rangle | \langle l, p \rangle \in S \cap L \times X\} \cup \{h(p) | p \in S \cap X\}$*, for* $h : X \rightarrow Y$ *and* $S \subseteq L \times X \cup X$*. Then* P_L*-coalgebras are in one-to-one correspondence with LTSs on* L*. The correspondence associates to the LTS* lts *the coalgebra* f *given by* $f(p) = \{\langle l, q \rangle | p \xrightarrow{l} q \in lts\} \cup \{p\}$ *and, conversely,* $p \xrightarrow{l} q \in lts$ *iff* $\langle l, q \rangle \in f(p)$.

We present now some results that exploit bialgebras, that is coalgebras for a functor on $\mathcal{Alg}(\Sigma)$. In bialgebras the algebraic structure of terms and the coalgebraic structure provided by the LTS are coherent, thus bisimilarity is a congruence w.r.t. all the operators in the algebra. These results are taken from [PT97].

Theorem 1.1. *Let* Σ *be a signature,* $F : \mathcal{Alg}(\Sigma) \rightarrow \mathcal{Alg}(\Sigma)$ *a functor and* f *an* F*-coalgebra. Then bisimilarity is a congruence for the LTS represented by* f.

Bialgebras correspond to LTSs whose states have an algebraic structure, using the correspondence in Proposition 1.1, but the inverse correspondence holds only for well-behaved LTSs. We show now how to prove that a LTS has a bialgebraic structure and thus it enjoys the congruence property.

We start by introducing a syntactic condition on SOS rules called De Simone format [de 85].

[1] Given $h : X \rightarrow A$ and its extension to terms with variables in X, $\hat{h} : T_\Sigma(X) \rightarrow A$, h is a solution to $p \xrightarrow{l} q$ for *lts* if and only if $\hat{h}(p) \xrightarrow{l} \hat{h}(q)$.

Definition 1.6 (De Simone format). *A rule r over Σ and L is in* De Simone format *iff it has the form:*

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{op(x_1, \ldots, x_n) \xrightarrow{l} p}$$

where $op \in \Sigma$, $I \subseteq \{1, \ldots, n\}$, the variables y_i are distinct from variables x_i, except for $y_i = x_i$ if $i \notin I$, each of them occurs in p at most once and no other variable occurs in p.

The first result that we present concerns term algebras without structural axioms.

Proposition 1.2 (Lifting of P_L). *Let $\Delta = \langle \Sigma, L, R \rangle$ be a transition specification with rules in De Simone format.*

Let us define $P_\Delta : \mathcal{Alg}(\Sigma) \to \mathcal{Alg}(\Sigma)$ as follows:

- $|P_\Delta(A)| = P_L(|A|)$;
- *whenever $r : \frac{\{x_i \xrightarrow{l_i} y_i | i \in I\}}{op(x_1,\ldots,x_n) \xrightarrow{l} p} \in R$ then there exists an operator op_r in $P_\Delta(A)$ such that $op_r(S_1, \ldots, S_n) = \{\langle l, p\{p_i/y_i | i \in I\}\{q_j/y_j | j \notin I\}\rangle | \langle l_i, p_i\rangle \in S_i, i \in I \wedge q_j \in S_j, j \notin I\}$;*
- *if $h : A \to B$ is a morphism in $\mathcal{Alg}(\Sigma)$ then $P_\Delta(h) : P_\Delta(A) \to P_\Delta(B)$ and $P_\Delta(h)(S) = \{\langle l, h(p)\rangle | \langle l, p\rangle \in S \cap L \times |A|\} \cup \{h(p) | p \in S \cap |A|\}$.*

Then P_Δ is a well-defined functor on $\mathcal{Alg}(\Sigma)$.

Corollary 1.1. *Let Δ be a transition specification with rules in De Simone format. Then the category $\mathcal{Coalg}(P_\Delta)$ is a category of bialgebras.*

Note that until now nothing guarantees that a P_L-coalgebra can be lifted to a P_Δ-coalgebra, since it is possible that the morphism can not be lifted to a morphism in $\mathcal{Alg}(\Sigma)$. A simple and important case when this is guaranteed is when $f : A \to P_\Delta(A)$ with $A = T_\Sigma$.

Proposition 1.3. *A P_L-coalgebra f defined by rules in De Simone format can be lifted to a P_Δ-coalgebra f' if $f : A \to P_L(A)$ with $A = T_\Sigma$.*

The above result is guaranteed since f' exists and it is unique since T_Σ is the initial object in the category.

The following result is due to [BM02] and it provides sufficient conditions to lift the morphism from T_Σ to $T_{(\Sigma,E)}$.

Theorem 1.2. *Let us consider an algebra $T_{(\Sigma,E)}$ for a signature Σ and a set of axioms E. Let us consider a LTS with states in $T_{(\Sigma,E)}$ specified by rules in De*

Simone format. Let us assume that for all equations $t_1 =_{\mathrm{E}} t'_1$ *in* E*, with free variables* $\{x_i | i \in I\}$*, we have proofs of:*

$$\frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1 \xrightarrow{l} t_2} \text{ implies } \frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t'_1 \xrightarrow{l} t'_2} \text{ and } t'_1 =_{\mathrm{E}} t'_2$$

and vice versa. Then the LTS has a bialgebraic structure.

The additional condition essentially requires that each axiom bisimulates, namely the LHS and the RHS are bisimilar, and this implies that $=_{\mathrm{E}}$ is a bisimulation.

1.3 Synchronization algebras

We present here synchronization algebras (SAs), which were proposed in [Win84] to deal with synchronizations among processes, as they are performed e.g. in CCS [Mil82]. We use a slightly different presentation in order to be consistent with standard SHR notation that will be introduced in Section 1.5.

In general, we consider a framework where processes can perform actions, and actions may or may not synchronize. Suppose that two processes P and P' can perform respectively actions a and a'. If actions a and a' synchronize then their synchronized execution corresponds to just one action, otherwise a and a' can not be executed together. Since an action can also be performed asynchronously, we must introduce also an action ϵ that denotes "not taking part to the synchronization". Thus the synchronized execution of a and ϵ corresponds to the asynchronous execution of a.

Definition 1.7 (Synchronization algebras).
A synchronization algebra $\langle Act, \bullet, \epsilon \rangle$ *consists of a binary, partial, associative and commutative operator* $\bullet$ *on a set of actions* Act*, which includes a distinguished element* ϵ*. We require that* $\forall a, b \in Act. a \bullet b = \epsilon$ *iff* $a = b = \epsilon$*.*

The binary operator $\bullet$ specifies how actions combine to form synchronized actions: if $a \bullet b$ is undefined then a and b can not synchronize, otherwise $a \bullet b$ is the composed action. Associativity and commutativity are required to ensure that, when many actions synchronize, the final result depends only on the actions and not on how the result is computed. The additional condition requires that actions never disappear, thus the result of synchronizing two non ϵ actions can never be ϵ.

Category theory [AHS90] has shown to be a useful tool to reason about SAs.

Definition 1.8 (Morphisms between SAs). *Let* $\langle Act_A, \bullet_A, \epsilon_A \rangle$ *and* $\langle Act_B, \bullet_B, \epsilon_B \rangle$ *be two SAs. A* SA morphism *from the first to the second is a function* $h : Act_A \to Act_B$ *such that* $h(\epsilon_A) = \epsilon_B$ *and* $a \bullet_A b = c \Leftrightarrow h(a) \bullet_B h(b) = h(c)$*.*

Furthermore a morphism is called synchronous *(strict in original Winskel's terminology) if* $h(a) = \epsilon_B \Leftrightarrow a = \epsilon_A$*.*

The following proposition is taken from [Win84].

Proposition 1.4. *SAs and SA morphisms form a category $\mathcal{SA}$ with function composition as composition and identity functions as identities. SAs with synchronous SA morphisms form a subcategory $s\mathcal{SA}$ of $\mathcal{SA}$.*

For some examples on SAs and on the use of category theory to combine them we refer to [Win84]. However, in Section 5.3 we analyze the structure of the category of *Synchronization Algebras with Mobility*, which is our original generalization of the category of SAs.

1.4 Fusion Calculus

The Fusion Calculus [PV98, Vic98] is a calculus for modeling mobile systems, and it is an evolution of the π-calculus [MPW92]. The interesting point is that it adds to the π-calculus the concept of fusion, that is the ability to merge arbitrary names, and nevertheless it is obtained by simplifying the calculus. In fact, the two action prefixes for communication (input and output) are symmetric whereas in the π-calculus they are not, and there is just one binding operator called *scope* whereas the π-calculus has two (restriction and input). As shown in [PV98] the π-calculus can be encoded as a subcalculus of the Fusion Calculus (the key point is that the input of π-calculus is obtained using input and scope). An asynchronous version of Fusion Calculus, where name fusions are handled explicitly as messages, is described in [GW00, GW04]. Here we follow the approach of [PV98, Vic98].

We present here in detail the syntax and the SOS semantics of Fusion Calculus. In our discussion we distinguish between *sequential agents* S (which have a summation as topmost operator) and general agents P.

We suppose to have an infinite set $\mathcal{N}$ of names ranged over by $u, v, \ldots, z$ and an infinite set of agent variables with metavariable X. Names represent communication channels.

Notation. *We use ϕ to denote an equivalence relation on $\mathcal{N}$, called fusion, which can be represented by a finite set of equalities (we denote with* id *the identity relation). The set of names* $\mathrm{n}(\phi)$ *in a fusion ϕ contains the names in non singleton equivalence classes.*

We introduce free actions first.

Definition 1.9 (Free actions). *The* free actions *are defined by:*

$$\begin{array}{rll} \alpha ::= & u\vec{x} & \textit{(Input)} \\ & \overline{u}\vec{x} & \textit{(Output)} \\ & \phi & \textit{(Fusion)} \end{array}$$

$$\text{(AF1) } (P_1|P_2)|P_3 \equiv P_1|(P_2|P_3) \qquad \text{(AF2) } P_1|P_2 \equiv P_2|P_1 \qquad \text{(AF3) } P|0 \equiv P$$
$$\text{(AF4) } (x)P \equiv (y)P\{y/x\} \text{ if } y \notin \text{fn}(P)$$
$$\text{(AF5) } \text{rec}\, X.P \equiv \text{rec}\, Y.P\{Y/X\} \text{ if } Y \notin \text{var}(P)$$
$$\text{(AF6) } (x)0 \equiv 0 \qquad \text{(AF7) } (x)(y)P \equiv (y)(x)P$$
$$\text{(AF8) } (x)(P_1|P_2) \equiv P_1|(x)P_2 \text{ if } x \notin \text{fn}(P_1)$$
$$\text{(AF9) } \text{rec}\, X.P \equiv P\{\text{rec}\, X.P/X\}$$

Table 1.1: Structural congruence for Fusion agents.

In the above definition, u is called the *subject* of the action, while names in $\vec{x}$ are the parameters of the action.

Notation. *For uniformity with standard Fusion notation, in actions we shorten* $\langle x_1, \ldots, x_n \rangle$ *as* $x_1 \ldots x_n$.

The syntax of Fusion agents is defined below.

Definition 1.10 (Syntax of Fusion Calculus). *The* Fusion agents *are defined by:*

$$S ::= \sum_i \alpha_i.P_i \quad \textit{(Guarded sum)}$$

$$\begin{array}{rrlrl} P ::= & 0 & \textit{(Inaction)} & \text{rec}\, X.P & \textit{(Recursion)} \\ & S & \textit{(Sequential agent)} & X & \textit{(Agent variable)} \\ & P|P & \textit{(Composition)} & [x=y]P & \textit{(Match)} \\ & (x)P & \textit{(Scope)} & [x \neq y]P & \textit{(Mismatch)} \end{array}$$

The scope operator is a binder for names thus x is bound in $(x)P$. Similarly rec is a binder for agent variables.

Notation. *We denote with* $\text{var}(P)$ *the set of free agent variables in agent* P. *We use infix* $+$ *for binary sum.*

Note that binary sum is associative and commutative. Note also that this syntax allows only agents with guarded summation. Furthermore we only consider agents which are closed w.r.t. agent variables and where in $\text{rec}\, X.P$ each occurrence of X is within a sequential agent (guarded recursion).

Processes are agents considered up to structural axioms defined as follows.

Definition 1.11 (Structural congruence). *The structural congruence* $\equiv$ *between Fusion agents is the least congruence satisfying the axioms in Table 1.1.*

Rules (AF1), (AF2) and (AF3) are the abelian monoid laws for parallel composition (associativity, commutativity and 0 as identity) while (AF4) and (AF5) are the α-conversion laws for names and for agent variables respectively. We have then

$$\frac{-}{\alpha.P \xrightarrow{\alpha} P}\ PREF \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}\ SUM$$

$$\frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}\ PAR \qquad \frac{P \xrightarrow{u\vec{x}} P' \quad Q \xrightarrow{\overline{u}\vec{y}} Q' \quad |\vec{x}| = |\vec{y}|}{P|Q \xrightarrow{\{\vec{x}=\vec{y}\}} P'|Q'}\ COM$$

$$\frac{P \xrightarrow{\phi} P' \quad z\phi x \quad z \neq x}{(z)P \xrightarrow{\phi\setminus z} P'\{x/z\}}\ SCOPE$$

$$\frac{P \xrightarrow{\alpha} P' \quad z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha} (z)P'}\ PASS \qquad \frac{P \xrightarrow{(\vec{y})a\vec{x}} P' \quad z \in \mathrm{n}(\vec{x}) \setminus \mathrm{n}(\vec{y}) \quad a \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z\vec{y})a\vec{x}} P'}\ OPEN$$

$$\frac{P \xrightarrow{\alpha} P'}{[x=x]P \xrightarrow{\alpha} P'}\ MATCH \qquad \frac{P \xrightarrow{\alpha} P' \quad x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}\ MISMATCH$$

$$\frac{P \equiv Q \quad P \xrightarrow{\alpha} P' \quad P' \equiv Q'}{Q \xrightarrow{\alpha} Q'}\ STRUCT$$

Table 1.2: SOS semantics for Fusion Calculus.

the scope laws (AF6) and (AF7), the scope extrusion law (AF8) and finally the recursion law (AF9).

Note that the set of free names occurring in a process does not depend on its representation, thus function $\mathrm{fn}(-)$ is well-defined on processes.

Definition 1.12. *A* bound action *is of the form $(\vec{z})a\vec{x}$ where $|\vec{z}| > 0$ and all elements in $\vec{z}$ are also in $\vec{x}$. Names in $\vec{z}$ are bound names. The actions consist of the free actions and the bound actions.*

Notation. *We use α to denote an action. We define $\phi\setminus z$ to mean $(\phi \cap ((\mathcal{N} \setminus \{z\}) \times (\mathcal{N} \setminus \{z\}))) \cup \{(z,z)\}$.*

We have now all the tools needed to define the operational semantics of Fusion Calculus.

Definition 1.13 (SOS semantics for Fusion Calculus). *The SOS semantics for Fusion Calculus is the least LTS satisfying the rules in Table 1.2.*

When only closed agents are considered one can use the simpler reduction semantics [Vic98]. We need the following auxiliary definition.

Definition 1.14 (Substitutive effect). *A* substitutive effect *of a fusion ϕ is any idempotent renaming $\sigma : \mathcal{N} \to \mathcal{N}$ such that $x\sigma = y\sigma$ iff $x\phi y$ and σ sends all members of each equivalence class of ϕ to the same representative in the class.*

The reduction semantics for Fusion Calculus (we present here just the rules for Fusion Calculus without match and mismatch, see [Vic98] for the complete ones) is the least relation satisfying the following rules.

Definition 1.15 (Reduction semantics for Fusion Calculus).

$$(\vec{z})(R|(P' + u\vec{x}.P)|(\overline{u}\vec{y}.Q + Q')) \to (\vec{z})(R|P|Q)\sigma$$

where $|\vec{x}| = |\vec{y}|$ *and* σ *is a substitutive effect of* $\{\vec{x} = \vec{y}\}$ *such that* $\mathrm{dom}(\sigma) \subseteq \mathrm{n}(\vec{z})$.

$$(\vec{z})(R|(P' + \phi.P)) \to (\vec{z})(R|P)\sigma$$

where σ *is a substitutive effect of* ϕ *such that* $\mathrm{dom}(\sigma) \subseteq \mathrm{n}(\vec{z})$.

$$\frac{P \equiv Q \quad P \to P' \quad P' \equiv Q'}{Q \to Q'}$$

The relation between SOS semantics and reduction semantics is given by the following theorem, taken from [Vic98].

Theorem 1.3. *Let* P *be a closed Fusion process without match and mismatch. Then* $P \xrightarrow{\mathrm{id}} Q$ *iff* $P \to Q$.

The concept of bisimilarity can be applied to Fusion Calculus, but special care is required to deal with names.

Definition 1.16 (Bisimilarity for Fusion Calculus). *A* bisimulation for Fusion Calculus *is a relation* $\mathcal{R}$ *on Fusion processes such that* $P\ \mathcal{R}\ Q$ *implies:*

- $P \xrightarrow{\alpha} P' \wedge \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{\alpha} Q' \wedge P'\sigma\ \mathcal{R}\ Q'\sigma$ *where* σ *is an mgu of* α *if* α *is a fusion, the identity otherwise*

and vice versa. Bisimilarity $\approx_f$ *is the maximal bisimulation.*

As shown in [Vic98], bisimilarity is not a congruence for Fusion Calculus. To get a compositional equivalence hyperbisimilarity is used.

Definition 1.17 (Hyperbisimilarity).
A hyperbisimulation *is a substitution-closed bisimulation.* Hyperbisimilarity $\sim_f$ *is the maximal hyperbisimulation.*

The following theorem, taken from [Vic98] ensures compositionality.

Theorem 1.4. *Hyperbisimilarity is a congruence for Fusion Calculus.*

We show here an useful decomposition that can be defined for Fusion agents.

Definition 1.18 (Standard decomposition for Fusion agents). *The* standard decomposition *of an agent P w.r.t. a standard sequence $S_{\mathcal{N}}$ of names is defined as:*

$$P = \hat{P}\sigma_P$$

where σ_P is called the standard renaming and $\hat{P}$ the standard agent of P. $\hat{P}$ is obtained by substituting the n-th occurrence of a free name in P (according to some fixed order dictated by the structure of P) with the n-th name in $S_{\mathcal{N}}$. We denote with $\mathrm{fnarray}(P)$ *the array of the free name occurrences in P, ordered according to the fixed order above. We require $\sigma_P = \{\mathrm{fnarray}(P)/\mathrm{fnarray}(\hat{P})\}$.*

The following lemma ensures that the decomposition is well-behaved w.r.t. renamings.

Lemma 1.1. *The decomposition of Definition 1.18 satisfies:*

$$P = Q\sigma \text{ implies } \hat{P} = \hat{Q} \wedge \sigma_P = \sigma_Q\sigma$$

for each pair of agents P and Q and each renaming σ.

Proof. The first part is trivial, since the renaming does not change neither the structure of the agent nor the ordering on names. The second part follows from the definition of standard renaming. □

The standard decomposition can be easily extended to processes, by taking for each process an agent in the equivalence class as representative and by decomposing it. Equivalent agents give rise to equivalent decompositions, thus the choice of the representative is not important.

1.5 Synchronized Hyperedge Replacement

In this section we present *Synchronized Hyperedge Replacement* (SHR), an approach to graph transformation where local *productions* are synchronized to get global *transitions*. Since SHR is central in our work, and since different variations of it have appeared in the literature, we give also a short overview of them, analyzing their differences.

SHR is mainly aimed at modeling distributed systems (but also software architectures or process calculi) by using *hypergraphs*, that is graphs where each edge (or, more precisely, hyperedge) can be connected to any number of nodes (instead of just two). In the following we shorten hypergraph with graph and hyperedge with edge. Usually edges model subsystems that are connected via shared nodes representing communication channels or ports.

The allowed transitions of a system are obtained by combining context-free productions that specify the behaviour of single edges. In particular, productions define

how a single edge can be rewritten into a generic graph, the effect of the rewriting on the interface, and the conditions that this rewriting imposes on adjacent nodes. Thus global transitions are obtained by applying in parallel different productions whose conditions are compatible. What exactly compatible means depends on which synchronization model is used. The Hoare model (so called since it extends the synchronization model of CSP [Hoa80]), for instance, requires that all the edges connected to the same node execute the same action on it. The Milner model (extending the model of CCS [Mil82]) instead requires exactly two edges to interact by performing complementary actions while the other edges must stay idle on that node.

SHR has been introduced in [CM83] with the name of "Grammars for Distributed Systems". Here Hoare synchronization was used, and the emphasis was on analyzing the history of the computation, obtained by explicitly representing past events as part of the graph. Infinite computations and concurrency issues have been considered in [DM87]. The framework has been extended in [CDM85] by allowing to merge and split nodes. In [RM99] there is a presentation of SHR inside the Tile Model [GM00], and an approach to find the allowed transitions of a distributed system using constraint solving techniques is also proposed. A main extension is given in [KM01], where *node mobility* is added. This is obtained by allowing actions to carry tuples of nodes. When actions synchronize the carried tuples of nodes are merged. This allows to create new connections at runtime. In particular, in [KM01] only newly created nodes can be communicated and merged. Also, a mapping into the Tile Model [GM00] is used to prove that an ad hoc defined bisimilarity is a congruence. Finally, a notation for representing graphs as *syntactic judgements* (see Definition 1.19 below) is introduced. Another important step is made in [HM01], where also old nodes can be communicated, but they can be merged only with new nodes. This kind of SHR, with Milner synchronization, is shown to be strictly related to π-calculus [MPW92]. A last improvement is presented in [FMT01], where fusions of arbitrary nodes are allowed. These are exploited to give semantics to Ambient Calculus [CG98], and, in particular, to the open primitive. Many applications of SHR can be found in the literature, in particular in the field of process calculi [Tuo03], of software architectures [Hir03, HIM00, CH04] and of quality of service [HT05].

We start the technical part of this section by introducing graphs, and their representation as (syntactic) judgements [KM01].

An edge is an atomic item with a label and with as many ordered *tentacles* as the rank, $\mathrm{rank}(L)$, of its label L. A graph is composed by a set of nodes and a set of such edges, and each edge is connected, by its tentacles, to its attachment nodes. A graph is connected to its environment by an interface which is a subset of its nodes. Nodes in the interface are called free nodes, while other nodes are called bound (or restricted).

We consider graphs up to graph isomorphisms that preserve free nodes, labels of edges, and connections between edges and nodes. We denote with *Graphs* the set

(AG1) $(G_1 \mid G_2) \mid G_3 \equiv G_1 \mid (G_2 \mid G_3)$ (AG2) $G_1 \mid G_2 \equiv G_2 \mid G_1$ (AG3) $G \mid nil \equiv G$
(AG4) $\nu x\ \nu y\ G \equiv \nu y\ \nu x\ G$ (AG5) $\nu x\ G \equiv G$ if $x \notin \mathrm{fn}(G)$
(AG6) $\nu x\ G \equiv \nu y\ G\{y/x\}$ if $y \notin \mathrm{fn}(G)$
(AG7) $\nu x\ G_1 \mid G_2 \equiv G_1 \mid \nu x\ G_2$ if $x \notin \mathrm{fn}(G_1)$

Table 1.3: Structural congruence for graph terms.

of such graphs.

In the definition of graphs as judgements nodes correspond to names, free nodes to free names and edges to basic terms of the form $L(x_1, \ldots, x_n)$, where the x_i are arbitrary names and $\mathrm{rank}(L) = n$. Also, *nil* represents the empty graph, $\mid$ is the parallel composition of graphs (merging nodes with the same name) and νy is a declaration of a bound node y.

Definition 1.19 (Graphs as judgements). *Let $\mathcal{N}$ be a fixed infinite set of names and LE a ranked alphabet of labels. A* judgement *is of the form $\Gamma \vdash G$ where:*

1. *$\Gamma \subseteq \mathcal{N}$ is a finite set of names (the free nodes of the graph);*
2. *G is a* graph term *generated by the grammar*
   $G ::= L(\vec{x}) \mid G \mid G \mid \nu y\ G \mid nil$
   where $\vec{x}$ is a vector of names, L is an edge label with $\mathrm{rank}(L) = |\vec{x}|$ and y is a name.

We define the restriction operator ν as a binder, and we demand that $\mathrm{fn}(G) \subseteq \Gamma$.

Note that by requiring just one inclusion we allow free isolated nodes in the graph.

Notation. *We assume that restriction has lower priority than parallel composition. When defining the interfaces, we use the notation Γ, x to denote the set obtained by adding x to Γ, assuming $x \notin \Gamma$ and Γ_1, Γ_2 to denote the union of Γ_1 and Γ_2, assuming $\Gamma_1 \cap \Gamma_2 = \emptyset$.*

Next definition introduces a notion of structural congruence on judgements.

Definition 1.20 (Structural congruence on graph judgements). *Graph terms are considered up to the axioms of structural congruence $\equiv$ in Table 1.3. As far as judgements are concerned, we define $\Gamma \vdash G \equiv \Gamma' \vdash G'$ iff $\Gamma = \Gamma'$ and $G \equiv G'$.*

Axioms (AG1), (AG2) and (AG3) define respectively the associativity, commutativity and identity over *nil* for operator $\mid$. Axioms (AG4) and (AG5) state that nodes of a graph can be restricted only once and in any order. Axiom (AG6) defines α-conversion of a graph w.r.t its bound names. Axiom (AG7) defines the interaction between restriction and parallel composition.

Note that axiom (AG5) garbage-collects isolated bound nodes, and axioms (AG4) and (AG5) allow to write just $\nu\{x_1, \ldots, x_n\}$ instead of $\nu x_1 \ldots \nu x_n$. We will drop the braces if no confusion will arise. Note also that function fn is well-defined on equivalence classes.

Judgements up to structural axioms are isomorphic to graphs up to graph isomorphisms. For a formal statement of the correspondence see [Hir03].

We define now SHR transitions, and then we show some inference rules to derive them from basic productions. As far as mobility is concerned, we follow the approach of [FMT01], since it is the most general approach. As we will show in Chapter 2 and in Chapter 4, it provides the same mobility primitives used in Fusion Calculus and that correspond to unification in Logic Programming.

Definition 1.21 (SHR transitions). *Let Act be a set of actions, and given $a \in Act$ let* $\mathrm{ar}(a)$ *be its arity. A* SHR transition *is a relation of the form:*

$$\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$$

where $\Gamma \vdash G$ and $\Phi \vdash G'$ are judgements for graphs, $\Lambda : \Gamma \to (Act \times \mathcal{N}^*)$ is a total function and $\pi : \Gamma \to \Gamma$ is an idempotent renaming. Function Λ assigns to each node x the action $a \in Act$ and the vector $\vec{y}$ of node references sent to x. If $\Lambda(x) = (a, \vec{y})$ then we define $\mathrm{act}_\Lambda(x) = a$ and $\mathrm{n}_\Lambda(x) = \vec{y}$. We require that $\mathrm{ar}(\mathrm{act}_\Lambda(x)) = |\,\mathrm{n}_\Lambda(x)|$.

We define:

- $\mathrm{n}(\Lambda) = \{z | \exists x.z \in \mathrm{n}_\Lambda(x)\}$ *set of communicated names;*
- $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$ *set of communicated fresh names.*

Renaming π allows to merge nodes. Since π is idempotent, it maps every node into a standard representative of its equivalence class. We require that $\forall x \in \mathrm{n}(\Lambda).x\pi = x$, i.e., only references to representatives can be communicated. Furthermore, we require $\Phi = \Gamma\pi \cup \Gamma_\Lambda$, namely free nodes are never erased ($\supseteq$) and new nodes are bound unless communicated ($\subseteq$).

In the above definition, π is necessary to allow merges among old nodes, since in the derivation of a transition these merges may interact with new merges (see, e.g., condition 2 of rule (merge-H) in Definition 1.23) and so they must be traced.

Note that w.r.t. [FMT01] we choose a slightly different presentation: we use for Λ a total function instead of a partial one. The two presentations are equivalent, we use in fact an action ϵ (see Definition 1.7) standing for "not taking part to the synchronization" to make the function total. Naturally, ϵ carries no parameters and thus it has arity 0. This presentation allows frequently a more succinct description of inference rules.

Note that in a transition the set of free names Φ of the resulting graph is fully determined by Λ and π (since $\Gamma = \mathrm{dom}(\Lambda)$), thus when writing the inference rules the definition of Φ for the conclusion can be left implicit.

Notation. *When writing Λ as set of pairs we write the triple $(x, a, \vec{y})$ for the pair $(x, (a, \vec{y}))$. To make the notation more concise, in the examples if no triple for x is written, then it is $(x, \epsilon, \langle\rangle)$.*

Graphically, we draw edges as rectangles and nodes as bullets. Tentacles are lines. We may write numbers to make explicit the order of tentacles attached to each edge. Bound nodes are drawn in white while free nodes are black. Also, we represent Λ by decorating every node x in the LHS with $\mathrm{act}_\Lambda(x)$ and $\mathrm{n}_\Lambda(x)$.

Productions are special transitions specifying the behaviour of one edge.

Definition 1.22 (Productions). *A* production *is an SHR transition of the form:*

$$x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G$$

where $x_1, \ldots, x_n$ are all distinct.

For each edge label L of rank n there is a special idle production $x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda_\epsilon, id} x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$ where $\Lambda_\epsilon(x_i) = (\epsilon, \langle\rangle)$ for each i. Idle productions are included in all sets of productions, which are also closed under α-conversion of names in $\{x_1, \ldots, x_n\} \cup \Phi$.

A transition is obtained by composing productions, which are first applied to disconnected edges. Composition is performed by merging nodes and thus connecting the edges. Finally, nodes can be bound.

We present now the set of inference rules for Hoare SHR (HSHR for short). These rules have first appeared in [Lan02], and we give here an equivalent but hopefully clearer presentation.

Definition 1.23 (Inference rules for Hoare SHR). *The admissible behaviours of Hoare SHR are defined by the following inference rules.*

$$\textit{(par-H)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2'}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G_2'}$$

where $(\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset$.

$$\textit{(merge-H)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2\sigma\rho}$$

where $\sigma : \Gamma \to \Gamma$ is an idempotent renaming and:

1. $\forall x, y \in \Gamma. x\sigma = y\sigma \Rightarrow \mathrm{act}_\Lambda(x) = \mathrm{act}_\Lambda(y)$
2. $\rho = \mathrm{mgu}(\{(\mathrm{n}_\Lambda(x))\sigma = (\mathrm{n}_\Lambda(y))\sigma | x\sigma = y\sigma\} \cup \{x\sigma = y\sigma | x\pi = y\pi\})$ *where ρ maps names to representatives in $\Gamma\sigma$ whenever possible*
3. $\forall z \in \Gamma. \Lambda'(z\sigma) = (\Lambda(z))\sigma\rho$

4. $\pi' = \rho\lfloor_{\Gamma\sigma}$

$$\textit{(res-H)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda\lfloor_\Gamma,\pi\lfloor_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

5. $(\exists y \in \Gamma. x\pi = y\pi) \Rightarrow x\pi \neq x$
6. $Z = (\{x\} \cup \mathrm{n}(\Lambda)) \setminus \mathrm{n}(\Lambda\lfloor_\Gamma)$

$$\textit{(new-H)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,a,\vec{y})\},\pi} \Phi' \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$ *and* $\mathrm{n}(\vec{y}) \cap (\Gamma \cup \Phi \cup \{x\}) = \emptyset$.

Rule (par-H) deals with the composition of transitions which have disjoint sets of nodes and rule (merge-H) allows to merge nodes (note that σ is a projection into representatives of equivalence classes). Condition 1 requires that all (connected) edges exhibit the same action on merged nodes. Condition 2 defines the most general unifier ρ of the union of two sets of equations: the first set identifies (the representatives of) the tuples associated to nodes merged by σ, while the second set of equations adds previous merges traced by π. Thus ρ is the merge resulting from both π and σ. Note that (condition 3) Λ is updated with these merges and that (condition 4) π' is ρ restricted to the nodes of the graph which is the source of the transition. Rule (res-H) binds node x, guaranteeing that x is not a representative if it belongs to a non trivial equivalence class and binding also all the nodes that were extruded on node x in the starting transition (x too is bound if it does not appear in the final label). Rule (new-H) allows to add to the source graph an isolated free node where arbitrary actions (with fresh names) are performed.

Notation. *We write* $\mathcal{P} \Vdash_H \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *iff* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *can be obtained from the productions in* $\mathcal{P}$ *using Hoare inference rules.*

A similar set of rules can be defined also for Milner SHR (MSHR for short) [FMT01]. Milner synchronization models point-to-point communication. Milner synchronization requires that actions can be either normal actions a (representing input) or coactions $\overline{a}$ (representing "output"). We also assume $\overline{\overline{a}} = a$. Furthermore, there are two special actions, the usual ϵ and an action τ of arity 0 representing a completed binary synchronization. Thus, Milner synchronization on a node x requires two complementary actions to interact, while other connected edges must stay idle on x (i.e., they all exhibit action ϵ on x). The final result of the binary synchronization is τ.

Definition 1.24 (Inference rules for Milner SHR). *The admissible behaviours of Milner SHR are defined by the following inference rules.*

$$\textit{(par-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2'}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G_2'}$$

where $(\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset$.

$$\textit{(merge-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

1. $\forall x, y \in \Gamma . x\sigma = y\sigma \wedge \mathrm{act}_\Lambda(x) \neq \epsilon \wedge \mathrm{act}_\Lambda(y) \neq \epsilon \wedge x \neq y \Rightarrow$
   $(\forall z \in \Gamma \setminus \{x, y\} . z\sigma = x\sigma \Rightarrow \mathrm{act}_\Lambda(z) = \epsilon) \wedge$
   $\mathrm{act}_\Lambda(x) = a \wedge \mathrm{act}_\Lambda(y) = \overline{a} \wedge a \neq \tau$
2. $\rho = \mathrm{mgu}(\{(\mathrm{n}_\Lambda(x))\sigma = (\mathrm{n}_\Lambda(y))\sigma | x\sigma = y\sigma\} \cup \{x\sigma = y\sigma | x\pi = y\pi\})$ *where* ρ *maps names to representatives in* $\Gamma\sigma$ *whenever possible*
3. $\Lambda'(z) = \begin{cases} (\tau, \langle\rangle) & \text{if } x\sigma = y\sigma = z \wedge x \neq y \wedge \mathrm{act}_\Lambda(x) \neq \epsilon \wedge \mathrm{act}_\Lambda(y) \neq \epsilon \\ (\Lambda(x))\sigma\rho & \text{if } x\sigma = z \wedge \mathrm{act}_\Lambda(x) \neq \epsilon \\ (\epsilon, \langle\rangle) & \text{otherwise} \end{cases}$
4. $\pi' = \rho|_{\Gamma\sigma}$
5. $U = (\Phi\sigma\rho) \setminus \Phi'$

$$\textit{(res-M)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda|_\Gamma, \pi|_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

6. $(\exists y \in \Gamma . x\pi = y\pi) \Rightarrow x\pi \neq x$
7. $\mathrm{act}_\Lambda(x) = \epsilon \vee \mathrm{act}_\Lambda(x) = \tau$
8. $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda|_\Gamma), Z = \emptyset$ *otherwise*

$$\textit{(new-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,\epsilon,\langle\rangle)\}, \pi} \Phi, x \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$.

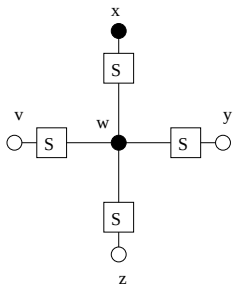

Figure 1.1: Star graph.

Rules are similar to those for Hoare synchronization. The main differences are that in rule (merge-M) during action synchronization we require to have (at most) two complementary non ϵ actions (condition 1), and that their composition is τ (condition 3). Thus we may have to reintroduce restrictions (condition 5) if some nodes were extruded by the synchronized actions. In rule (res-M), just nodes x where ϵ or τ actions are performed can be restricted (condition 7), and since these actions have arity 0 only node x may have to be restricted in the final graph. Finally, in rule (new-M) only action ϵ is allowed on the newly created node.

Notation. *We write* $\mathcal{P} \Vdash_M \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *iff* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *can be obtained from the productions in* $\mathcal{P}$ *using Milner inference rules.*

We show here a simple lemma, stating that transitions are closed w.r.t. injective renamings.

Lemma 1.2. *Let* ψ *be an injective renaming and* $\mathcal{P}$ *a set of SHR productions. If* $\mathcal{P} \Vdash_S \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *where either* $S = H$ *or* $S = M$*, then* $\mathcal{P} \Vdash_S \Gamma\psi \vdash G_1\psi \xrightarrow{\Lambda\psi,\pi\psi} \Phi\psi \vdash G_2\psi$.

Proof. By rule induction. □

The definition of computation is standard.

Definition 1.25 (SHR computations). *A* SHR computation *is a sequence of SHR transitions such that for each* i *the final graph of transition* i *is the starting graph of transition* $i+1$*. A SHR computation is called* trivial *iff the starting graph is equal to the final graph.*

We present now an example of HSHR computation.

Example 1.1 ([HIM00]). *We show here how to use HSHR to derive a 4 elements ring starting from a one element ring, and how we can then specify a reconfiguration that transforms the ring into the star graph in Figure 1.1.*

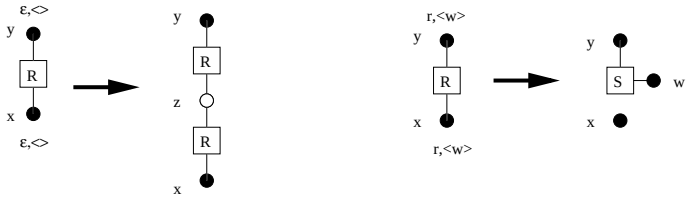

Figure 1.2: Productions.

We use the set of productions $\mathcal{P}$ generated by:

$$x, y \vdash R(x, y) \xrightarrow{(x,\epsilon,\langle\rangle),(y,\epsilon,\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y) \tag{1.1}$$

$$x, y \vdash R(x, y) \xrightarrow{(x,r,\langle w\rangle),(y,r,\langle w\rangle),\mathrm{id}} x, y, w \vdash S(y, w) \tag{1.2}$$

The two productions are graphically represented in Figure 1.2. The production 1.1 allows to create rings. In fact, we can create any ring with computations like:

$$\begin{aligned} x \vdash R(x, x) &\xrightarrow{(x,\epsilon,\langle\rangle),\mathrm{id}} x \vdash \nu y\ R(x, y)|R(y, x) \xrightarrow{(x,\epsilon,\langle\rangle),\mathrm{id}} \\ &\to x \vdash \nu y, z\ R(x, y)|R(y, z)|R(z, x) \xrightarrow{(x,\epsilon,\langle\rangle),\mathrm{id}} \\ &\to x \vdash \nu y, z, v\ R(x, y)|R(y, z)|R(z, v)|R(v, x) \end{aligned}$$

The above transitions are generated by applying a renamed version of production 1.1 to one edge, and idle productions to the other edges.

In order to perform the reconfiguration into a star, we need productions with non ϵ actions, such as production 1.2. By applying suitably renamed productions to each edge we can derive the wanted transition:

$$\begin{aligned} x \vdash \nu y, z, v\ R(x, y)|R(y, z)|R(z, v)|R(v, x) &\xrightarrow{(x,r,\langle w\rangle),\mathrm{id}} \\ &\to x, w \vdash \nu y, z, v\ S(y, w)|S(z, w)|S(v, w)|S(x, w) \end{aligned}$$

More in details we take four renamed versions of production 1.2:

$$\begin{aligned} x, y \vdash R(x, y) &\xrightarrow{(x,r,\langle w\rangle),(y,r,\langle w\rangle),\mathrm{id}} x, y, w \vdash S(y, w) \\ y_1, z \vdash R(y_1, z) &\xrightarrow{(y_1,r,\langle w_1\rangle),(z,r,\langle w_1\rangle),\mathrm{id}} y_1, z, w_1 \vdash S(z, w_1) \\ z_1, v \vdash R(z_1, v) &\xrightarrow{(z_1,r,\langle w_2\rangle),(v,r,\langle w_2\rangle),\mathrm{id}} z_1, v, w_2 \vdash S(v, w_2) \\ v_1, x_1 \vdash R(v_1, x_1) &\xrightarrow{(v_1,r,\langle w_3\rangle),(x_1,r,\langle w_3\rangle),\mathrm{id}} v_1, x_1, w_3 \vdash S(x_1, w_3) \end{aligned}$$

We can apply rule (par-H) three times to put all the edges in parallel (notice that this can be done since we have chosen productions with disjoint sets of names, and

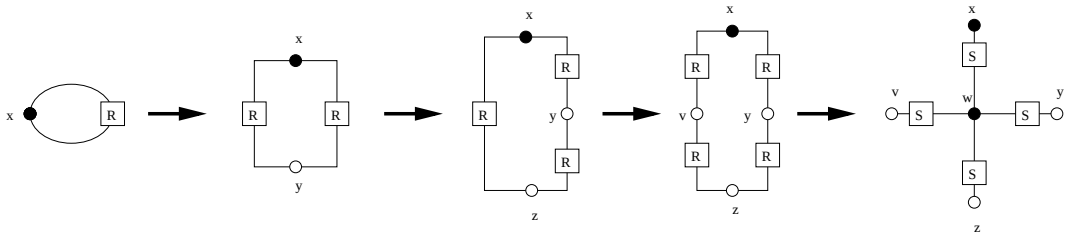

Figure 1.3: Ring creation and reconfiguration to star.

that the result is independent on the order in which we compose the productions, up to structural equivalence), obtaining the transition:

$$x, y, y_1, z, z_1, v, v_1, x_1 \vdash R(x,y)|R(y_1,z)|R(z_1,v)|R(v_1,x_1)$$
$$\xrightarrow{(x,r,\langle w\rangle),(y,r,\langle w\rangle),(y_1,r,\langle w_1\rangle),(z,r,\langle w_1\rangle),(z_1,r,\langle w_2\rangle),(v,r,\langle w_2\rangle),(v_1,r,\langle w_3\rangle),(x_1,r,\langle w_3\rangle),\mathrm{id}}$$
$$x, y, w, y_1, z, w_1, z_1, v, w_2, v_1, x_1, w_3 \vdash S(y,w)|S(z,w_1)|S(v,w_2)|S(x_1,w_3)$$

Now we can apply rule (merge-H), with a renaming $\sigma = \{y/y_1, z/z_1, v/v_1, x/x_1\}$. *The condition on actions given by Hoare synchronization is trivially satisfied, since the same action* r *is executed on each node. The renaming* ρ *that we obtain is* $\{w/w_1, w/w_2, w/w_3\}$. *Thus we get as transition:*

$$x, y, z, v \vdash R(x,y)|R(y,z)|R(z,v)|R(v,x) \xrightarrow{(x,r,\langle w\rangle),(y,r,\langle w\rangle),(z,r,\langle w\rangle),(v,r,\langle w\rangle),\mathrm{id}}$$
$$\rightarrow x, y, z, v, w \vdash S(y,w)|S(z,w)|S(v,w)|S(x,w)$$

Note that we could get the same result by applying many times the rule, using any sequence of renamings whose composition amounts to σ.

Finally, we can restrict nodes by applying rule (res-H) (one time for each node to be bound in the starting graph), obtaining, as desired:

$$x \vdash \nu y, z, v\ R(x,y)|R(y,z)|R(z,v)|R(v,x) \xrightarrow{(x,r,\langle w\rangle),\mathrm{id}}$$
$$\rightarrow x, w \vdash \nu y, z, v\ S(y,w)|S(z,w)|S(v,w)|S(x,w)$$

Note that, after having chosen to apply production 1.2 to one of the edges R, *then all the edges in the ring must use the same production, since they must synchronize via action* r. *They must agree also on* $\mathrm{n}_\Lambda(x)$ *for every* x, *thus all the newly created nodes are merged. Thus the shown transition is the only one that can be derived using the set of productions* $\mathcal{P}$ *and at least one production 1.2 (up to the choice of the name of the new node).*

The whole computation is represented in Figure 1.3.

1.6 Logic Programming

Since we think that Logic Programming is worldwide known, in this section we do not present a detailed description of Logic Programming techniques, for which we refer to [Llo93], but we just fix the notation that we will use and we describe the particular way in which we use Logic Programming, because it is a bit non standard.

In fact, we are not interested in logic computations as refutations of goals to discover which computed answer substitutions make a goal true, such as usually done for theorem proving, problem solving or artificial intelligence. We consider instead Logic Programming as a *goal rewriting mechanism*. We consider logic atoms as concurrent communicating processes that evolve according to the rules defined by the clauses and that use *unification* as the fundamental interaction primitive. A similar view of Logic Programming has been used in [BMR01].

In order to stress the similarities between Logic Programming and process calculi we present a semantics of Logic Programming based on a LTS. Let us introduce the syntax first.

Definition 1.26 (Syntax for clauses and goals). Clauses *(C) and* goals *(G) are defined by the following grammar:*

$$\begin{aligned} C &::= A \leftarrow G \\ G &::= G, G \mid A \mid \Box \end{aligned}$$

where A is a logic atom (i.e., a predicate), "," is the AND conjunction and $\Box$ is the empty goal. We assume "," to be associative and commutative and with unit $\Box$.

The informal semantics of $A \leftarrow B_1, \ldots, B_n$ is, in the common intuition, "for every assignment of the variables, if $B_1, \ldots, B_n$ are all true, then A is true". However, in our case, we consider as intuitive meaning "goal A can evolve to the system containing goals $B_1, \ldots, B_n$, while interacting with other concurrent goals by performing substitutions on the common variables".

Remark 1.1. *For historical reasons G is used to denote both a goal and a graph term, however we will always make clear in which way it is used.*

A *logic program* is a finite set of clauses. Computations in Logic Programming are also called SLD-derivations (from "Linear resolution for Definite clauses with Selection function"). In common Logic Programming one is interested in refutations, that is computations ending with the empty goal. In our setting instead refutations are not so interesting, since they represent computations that completely destroy the system. We are interested instead in partial computations describing the evolution of the system from one state to the other.

Definition 1.27 (SLD-steps). *Let P be a logic program. The allowed* SLD-steps *are given by the LTS defined by the following inference rules:*

$$\frac{H \leftarrow B_1, \ldots, B_k \in P \qquad \theta = \mathrm{mgu}(\{A = H\rho\})}{P \Vdash A \xrightarrow{\theta} (B_1, \ldots, B_k)\rho\theta} \qquad \textit{atomic goal}$$

where ρ is an injective renaming of variables such that all the variables in the clause variant $(H \leftarrow B_1, \ldots, B_k)\rho$ are fresh (i.e., not already used in the computation) and A is an atom.

$$\frac{P \Vdash G \xrightarrow{\theta} F}{P \Vdash G, G' \xrightarrow{\theta} F, G'\theta} \qquad \textit{conjunctive goal}$$

We will omit $P \Vdash$ if P is clear from the context.

The definition of SLD-computation is standard.

Definition 1.28 (SLD-computations). *A* SLD-computation *is a sequence of SLD-steps allowed by some program P and starting from some goal G, such that for each i the final goal of step i is the starting goal of step $i + 1$. The substitution associated to a SLD-computation is obtained by composing (in the order) the substitutions observed at each step.*

An example of our use of Logic Programming is, e.g., Example 4.2.

Part I

Comparing Models for Global Computing

Roadmap to part I

We present here some comparisons between different frameworks used in the literature for modeling GC systems. In particular, we present a chain of mappings starting from Fusion Calculus [PV98], going through SHR [HM01] and arriving to Logic Programming [Llo93].

In the first step (Chapter 2) the correspondence is stated at the level of LTS, and it concerns Fusion Calculus without match and mismatch and Milner SHR. The mapping is essentially an injective homomorphism, in fact it maps parallel composition and scope into the corresponding operators of SHR. The main point is that both the graph structure and the productions specifying the behaviour of the system are derived from the structure of Fusion process. Milner SHR adds to the Fusion framework the ability of performing many actions inside the same transition. The subset of SHR that more tightly corresponds to Fusion Calculus, which we call interleaving Milner SHR, is formally characterized. As a simple application, we show a concurrent semantics for Fusion Calculus obtained via the mapping.

The second step (Chapter 3) is internal to the SHR framework, and it corresponds to change the basic synchronization primitive from Milner synchronization (i.e., message passing) to Hoare synchronization (agreement). Instead of simply providing a mapping, we give a full comparison between the two models. Formal results prove that in general the two models have different expressive power. An important case in which the expressive power is the same is found in closed 2-shared graphs, graphs with no interface and where each node is shared by exactly two tentacles. The case of general closed graphs is recovered by introducing a suitable translation for graphs, based on the concept of amoeboid. As final result, a translation is given from Hoare SHR to Milner SHR in the case of closed graphs, and the inverse translation is proved to work only when at most three actions are performed on each node. In particular, interleaving Milner SHR satisfies this constraint. The correspondence is in terms of reachability for the unlabeled transition system. A correspondence between the LTSs is not meaningful since our approach works only for closed graphs.

The final step (Chapter 4) is from Hoare SHR into Logic Programming. Here again we have a correspondence at the level of LTS, and again the correspondence is essentially an injective homomorphism. To get the mapping however, two variations of Logic Programming have to be used. Synchronized Logic Programming (SLP), which adds to Logic Programming a transactional mechanism, is considered to mimic SHR Hoare synchronization. Moreover, an hiding operator is added to model SHR restriction.

Chapter 2

From Fusion Calculus to Milner SHR

In this chapter we present a mapping from Fusion Calculus into Milner SHR, extending the work of [HM01] for the monadic π-calculus [MPW92]. The mapping is essentially an injective homomorphism, since it maps parallel composition into parallel composition and scope into restriction. This proves that the two models are strongly related, e.g., they use the same primitives for synchronization (Milner synchronization) and mobility (i.e., they allow to merge arbitrary names/nodes). SHR proves to be more general, since it allows to concurrently execute many synchronizations, both because many actions can be executed in the same transition and because the same edge can perform multiple actions at the same time (this allows multi-party synchronization). The image of Fusion Calculus along the mapping does not allow these possibilities, and it is characterized by a particular set of productions and inference rules, giving rise to interleaving Milner SHR. As an application we use the mapping to define a concurrent semantics for Fusion processes.

2.1 Mapping Fusion Calculus to Milner SHR

We define here the translation from Fusion Calculus into Milner SHR. It refers only to Fusion processes without match or mismatch. The mapping shows tight analogies between the two formalisms, that is the same underlying structure based on parallel composition and scope/restriction, the same synchronization model, i.e., Milner synchronization, and the same techniques for name-handling, based on fusions. This is a bit surprising since the two formalisms come from different areas (process calculi and graph transformation) and they have been developed independently. Clearly, the two formalisms have been chosen exactly since they showed interesting analogies, and even if SHR can be used as a general framework for giving visual representations to process calculi, in other cases (see, e.g., [FMT01] where the case of Ambient calculus [CG98] is analyzed) the mapping is not so straightforward.

The mapping allows to give a visual presentation to Fusion Calculus, and it also provides a clean separation between the topological part represented by the graph and the behavioural part defined by productions, which are instead mixed in Fusion Calculus and in general in process calculi. Furthermore, since Fusion Calculus proves to correspond to a subset of MSHR, the translation suggests many generalizations for Fusion Calculus. Among the others, a concurrent semantics can be derived and action prefixes can be generalized to allow different forms of synchronization (see PRISMA Calculus in Section 7.1).

We present now the different parts of the mapping. First of all, we must extract from the Fusion process the topology of the system, and represent it as a graph. The general idea is to map sequential processes into edges, and names into nodes. In order to have a finite number of productions that describe the evolution of each process, we use edges and productions only for standard sequential agents (see Definition 1.18). The actual names are instead tracked by nodes, and the name-handling ability of SHR allows to animate general agents.

Definition 2.1 (Translation of Fusion agents). *The translation* $[\![-]\!]^F_{SHR}$ *from Fusion agents to graph terms is defined by:*

- $[\![0]\!]^F_{SHR} = nil$
- $[\![S]\!]^F_{SHR} = L_{\hat{P}}(\text{fnarray}(S))$ *where* P *is the process for agent* S *and* $\text{fnarray}(S)$ *is the sequence of free name occurrences in agent* S
- $[\![P_1|P_2]\!]^F_{SHR} = [\![P_1]\!]^F_{SHR}|[\![P_2]\!]^F_{SHR}$
- $[\![(x)P]\!]^F_{SHR} = \nu x\ [\![P]\!]^F_{SHR}$
- $[\![\text{rec}\, X.P]\!]^F_{SHR} = [\![P\{\text{rec}\, X.P/X\}]\!]^F_{SHR}$

Note that essentially the translation is an isomorphism, and the most important part is the translation of sequential agents, which are decomposed according to Definition 1.18 to separate their structure from their names. In the translation we have not to deal explicitly with agent variables, since they can occur only inside sequential agents since recursion is guarded. Also, note that the last rule can not cause the translation to cycle since after a step a sequential agent is reached. It is interesting to note that labels of edges are just suitable placeholders, used to provide a link between edges and their behaviour. Similarly, standard names link positions in the agent and the corresponding tentacles.

For simplicity, we suppose that the set of nodes in SHR coincides with the set of names in Fusion Calculus (otherwise we need a bijective translation function).

Next lemma shows that the translation $[\![-]\!]^F_{SHR}$ maps structurally congruent agents to structurally congruent graph terms (and vice versa), thus it can be used also on equivalence classes.

Lemma 2.1. *Let* P_1 *and* P_2 *be Fusion agents. Then* $P_1 \equiv P_2$ *iff* $[\![P_1]\!]^F_{SHR} \equiv [\![P_2]\!]^F_{SHR}$.

Proof. Rules (AF1), (AF2) and (AF3) are the exact counterparts of rules (AG1), (AG2) and (AG3), up to the translation. Also, rule (AF4) corresponds to (AG6), (AF7) to (AG4), and (AF8) to (AG7).

Rules (AF5) and (AF9) can be applied only inside a sequential agent, since recursion is guarded. Thus, since the translation of a sequential agent considers only its equivalence class, applying these rules can not change the result of the translation. As far as the inverse implication is concerned, by definition the labels of two edges are equal only if the two corresponding agents are structurally congruent.

Also, rule (AF6) is a particular case (up to the translation) of rule (AG5). For the inverse implication, rule (AG5) can be proved using rules (AF3), (AF8) and (AF6), up to the translation. □

The lemma shows that the two formalisms have the same underlying structure, that is a parallel composition operator and a binder for names, ruled by equivalent congruences. Thus the translation can trivially be extended to processes.

Next lemma shows that the translation is also well-behaved w.r.t. renamings.

Lemma 2.2. *Let P be a Fusion process and σ a renaming. Then $[\![P]\!]^F_{SHR}\sigma = [\![P\sigma]\!]^F_{SHR}$.*

Proof. By structural induction on an agent representation of P. □

Fusion communication actions (and in particular Fusion communication prefixes) are translated into SHR actions according to the following definition.

Definition 2.2 (Translation of communication actions). *Communication actions are translated as follows:*

- $[\![(\vec{y})u\vec{x}]\!]^F_{SHR} = (in_n, \vec{x})$ *where* $n = |\vec{x}|$
- $[\![(\vec{y})\overline{u}\vec{x}]\!]^F_{SHR} = (out_n, \vec{x})$ *where* $n = |\vec{x}|$

We define in_n and out_n as complementary actions, i.e. $in_n = \overline{out_n}$ and $out_n = \overline{in_n}$.

Note that for each Fusion action (that is, either input or output), we have a family of SHR actions, indexed by their arity. This allows to manage the side condition $|\vec{x}| = |\vec{y}|$ of Fusion rule COM using only standard SHR primitives.

Note that the translation makes the distinction between free names and bound names in the label to vanish, since $\vec{y}$ is discarded. However for any transition $P \xrightarrow{\alpha} P'$ we have $\mathrm{bn}(\alpha) = \mathrm{n}(\alpha) \setminus \mathrm{fn}(P) = \mathrm{n}([\![\alpha]\!]^F_{SHR}) \setminus \mathrm{fn}([\![P]\!]^F_{SHR})$, thus we can retrieve the set of bound names when necessary. However, the fact that this information does not appear in the label will influence the observational semantics. We will discuss this issue in more detail in Section 9.1.

As the last step in our translation, we need to define the set of productions specifying the behaviour of the graphs. In particular, productions are required for each edge label, that is for each standard sequential agent that can be generated

by the computation. Even if this defines an infinite number of productions, only a finite amount of them are required for all the possible derivatives of one process. For a proof of that fact, see the analogous result for π-calculus in [Hir03].

Definition 2.3 (Productions for Fusion agents). *We define productions only for translations of standard sequential agents $\sum_i \alpha_i.P_i$. Let Γ be $\mathrm{fn}([\![\sum_i \alpha_i.P_i]\!]^F_{SHR})$. We have a production for each i, whose form depends on α_i being a communication or a fusion action.*

For communication actions:

$$\Gamma \vdash [\![\sum_i \alpha_i.P_i]\!]^F_{SHR} \xrightarrow{\Lambda_{\alpha_i}, id} \Gamma \vdash [\![P_i]\!]^F_{SHR}$$

where $\Lambda_{\alpha_i}(x) = [\![\alpha_i]\!]^F_{SHR}$ if x is the subject of α_i, $\Lambda_{\alpha_i}(x) = (\epsilon, \langle\rangle)$ otherwise.

For fusion actions:

$$\Gamma \vdash [\![\sum_i \alpha_i.P_i]\!]^F_{SHR} \xrightarrow{\Lambda_\epsilon, \pi} \Gamma\pi \vdash [\![P_i\pi]\!]^F_{SHR}$$

where $\Lambda_\epsilon(x) = (\epsilon, \langle\rangle)$ for each x in Γ and π is a substitutive effect of α_i.

Note that since in MSHR computations nodes are never deleted, then the graphs obtained during the computation will have some unused nodes (but these are isolated nodes, so they do not influence the computation).

We want to show now that the translation preserves the behaviour of processes. We start with a simple lemma relating substitutive effects of fusions and mgus.

Lemma 2.3. *Let ϕ be a fusion and σ be a renaming. Then the following two propositions are equivalent:*

- *σ is a substitutive effect of ϕ;*
- *σ is a relevant (i.e., it does not add new names) mgu of ϕ.*

Proof. First of all, note that ϕ contains just equations between names, thus the mgu of ϕ always exists and it is a renaming. The condition that $x\sigma = y\sigma$ iff $x\phi y$ is exactly the definition of most general unifier. The additional condition specifies that it is relevant. □

An interesting characteristic of transitions obtained from Fusion Calculus is that they cause no generation of new names as shown by the following lemma (but they can cause extrusions of bound names). However we can have infinitely many names since the axiom for recursion allows a finite representation of an infinite agent (which may contain infinitely many names).

Lemma 2.4. *For each process P, if $P \xrightarrow{\alpha} P'$ then $\mathrm{fn}(\alpha) \subseteq \mathrm{fn}([\![P]\!]^F_{SHR})$.*

Proof. Just note that all the productions satisfy the lemma, and that the property is preserved by all the inference rules. □

Next lemma shows a correctness result for the translation, namely that all the transitions of a Fusion process P give rise to corresponding transitions for its translation.

Theorem 2.1 (Correctness). *Let $\mathcal{P}_F$ be the set of Fusion productions. For each Fusion process P and each $\Gamma \supseteq \mathrm{fn}([\![P]\!]^F_{SHR})$ if $P \xrightarrow{\alpha} P'$ and $\Gamma \cap \mathrm{bn}(\alpha) = \emptyset$ then:*

- *if α is a communication action, $\mathcal{P}_F \Vdash_M \Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda_\alpha, id} \Gamma, \Gamma_E \vdash [\![P']\!]^F_{SHR}$ where $\Lambda_\alpha(x) = [\![\alpha]\!]^F_{SHR}$ if x is the subject of α, $\Lambda_\alpha(x) = (\epsilon, \langle\rangle)$ otherwise and $\Gamma_E = \mathrm{bn}(\alpha)$;*
- *if α is a fusion action, for each substitutive effect π of α, $\mathcal{P}_F \Vdash_M \Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda, \pi} \Gamma\pi \vdash [\![P'\pi]\!]^F_{SHR}$ where either $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \Gamma$ or there exists $y \in \Gamma$ such that $\Lambda(y) = (\tau, \langle\rangle)$ and $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \Gamma \setminus \{y\}$.*

Proof. The proof is by rule induction on the derivation of $P \xrightarrow{\alpha} P'$. In the following, we will consider explicitly only the case $\Gamma = \mathrm{fn}([\![P]\!]^F_{SHR})$. For the general case where $\Gamma \supseteq \mathrm{fn}([\![P]\!]^F_{SHR})$ we have just to add the missing names using rule (new-M) (notice that this can be done since by hypothesis $\Gamma \cap \mathrm{bn}(\alpha) = \emptyset$).

Rule PREF) Let us consider the standard decomposition $\widehat{\alpha.P}\sigma_{\alpha.P}$ of $\alpha.P$, and let us denote with α' its prefix and with P' its continuation. Since $\widehat{\alpha.P}$ is a standard sequential process, then we have a production for the edge with label $\widehat{\alpha.P}$ of the form defined in Definition 2.3. We can thus apply SHR rule (merge-M) to that production with $\sigma = \sigma_{\alpha.P}$. Let us consider the case of a communication prefix first. We have $\rho = \mathrm{id}$, thus we obtain:

$$\Gamma_{\widehat{\alpha.P}}\sigma_{\alpha.P} \vdash [\![\widehat{\alpha.P}]\!]^F_{SHR}\sigma_{\alpha.P} \xrightarrow{\Lambda\sigma_{\alpha.P}, \mathrm{id}} \Gamma_{\widehat{\alpha.P}}\sigma_{\alpha.P} \vdash [\![P']\!]^F_{SHR}\sigma_{\alpha.P}$$

Thanks to Lemma 2.2 we get the thesis.

Let us consider now the case of a fusion prefix.

We have $\rho = \mathrm{mgu}(\{x\sigma = y\sigma | x\pi = y\pi\})$ where π is an mgu of fusion α'. Also, π' in the consequence of rule (merge-M) equals ρ thanks to Lemma 2.4. Thus we have:

$$\Gamma_{\widehat{\alpha.P}}\sigma_{\alpha.P} \vdash [\![\widehat{\alpha.P}]\!]^F_{SHR}\sigma_{\alpha.P} \xrightarrow{\Lambda_\epsilon, \pi'} \Gamma_{\widehat{\alpha.P}}\sigma_{\alpha.P}\pi' \vdash [\![P']\!]^F_{SHR}\sigma_{\alpha.P}\pi'$$

The only thing to prove is that π' can be any substitutive effect of $\alpha = \alpha'\sigma_{\alpha.P}$, but this follows easily from the definition.

Rule SUM) Since we have only guarded sums, P must be of the form $\alpha.P'$. Thus the proof is essentially equal to the one for rule PREF, considering now a production for the standard agent for $\alpha.P' + Q$.

Rule PAR) We have that $[\![P|Q]\!]^F_{SHR} = [\![P]\!]^F_{SHR} | [\![Q]\!]^F_{SHR}$. By inductive hypothesis we have a SHR transition T corresponding to $P \xrightarrow{\alpha} P'$. Let us now consider a

bijective renaming σ such that $\mathrm{fn}([\![Q]\!]^F_{SHR}\sigma)$ contains only names that are new w.r.t. T. As specified in Definition 1.22, we have idle productions for each edge in $[\![Q]\!]^F_{SHR}$. We can easily derive from those productions an idle transition for $[\![Q]\!]^F_{SHR}\sigma$. Since all the names therein are new w.r.t. T, then we can apply rule (par-M) to the two transitions. After that, we can apply rule (merge-M) with a renaming σ^{-1} which is the inverse of σ. The result is exactly a transition for $[\![P]\!]^F_{SHR}|[\![Q]\!]^F_{SHR}$. If α is a communication action, then the only non ϵ action is the one from α, as desired, and no new merges are produced since ρ and thus π' are identities. If α is a fusion action instead, then $\pi' = \pi$ up to a change of representatives, and it is applied to the occurrence of $[\![Q]\!]^F_{SHR}$ in the RHS, as required.

Rule COM) By hypothesis we have two transitions of the form:

$$\Gamma_1 \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda_{u\vec{x}},id} \Gamma_1 \vdash [\![P']\!]^F_{SHR} \tag{2.1}$$

$$\Gamma_2 \vdash [\![Q]\!]^F_{SHR} \xrightarrow{\Lambda_{\overline{u}\vec{y}},id} \Gamma_2 \vdash [\![Q']\!]^F_{SHR} \tag{2.2}$$

where $\Lambda_{u\vec{x}}$ and $\Lambda_{\overline{u}\vec{y}}$ perform the corresponding action on node u and actions $(\epsilon, \langle\rangle)$ on the other nodes.

Let us consider a bijective renaming σ that maps each name in transition 2.1 to names not occurring in transition 2.2. Thanks to Lemma 1.2, we can take the renamed version of transition 2.2 according to σ and we can use rule (par-M) to compose it with transition 2.1. Finally, we can apply rule (merge-M) with the renaming σ^{-1} that is the inverse of σ. With respect to the case for rule PAR, when applying the rule a synchronization happens. In fact, the two transitions perform non ϵ actions on both the nodes mapped to u, and the two actions are complementary. The resulting transition has as starting graph $[\![P|Q]\!]^F_{SHR}$, a synchronization Λ which is $(\epsilon, \langle\rangle)$ on each node but u, and which evaluates to $(\tau, \langle\rangle)$ on u. Also, renaming $\pi' = \rho = \mathrm{mgu}(\{\vec{x} = \vec{y}\})$ is the required one. Notice that it is applied to the RHS of the transition.

Rule SCOPE) By hypothesis we have a transition of the form:

$$\Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Gamma\pi \vdash [\![P'\pi]\!]^F_{SHR}$$

where π is a substitutive effect of ϕ and either $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \Gamma$ or there exists $y \in \Gamma$ such that $\Lambda(y) = (\tau, \langle\rangle)$ and $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \Gamma \setminus \{y\}$.

We want to bind z. Since $z \in \mathrm{fn}([\![P]\!]^F_{SHR})$, then we can apply rule (res-M) to bind it. The synchronization $\Lambda\lfloor_{\setminus\{z\}}$ has the wanted form (notice that the only node on which a τ action was executed may be the one to be restricted). As far as the renaming π is concerned, because of the first side condition we have to choose a transition where the representative of the equivalence class of z is x (we always have such a transition, since any node in the class can be chosen as representative). Restricting the domain of π amounts to delete the binding $\{x/z\}$. Notice that there is no need to apply it in the SHR framework, since it has already been applied when the fusion has been computed. Also, no additional node is restricted in the

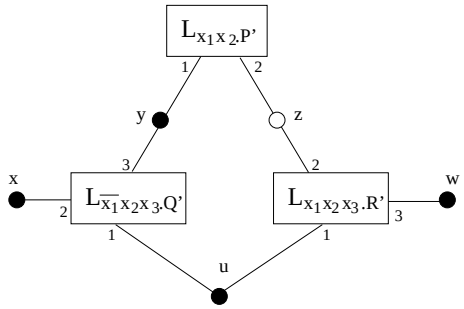

Figure 2.1: Fusion process translated into a graph.

RHS since the restricted node z does not appear there, thus the restriction can be garbage-collected using the axioms of structural congruence.

Rule PASS) This case is trivially proved by using SHR rule (res-M).

Rule OPEN) Here again rule (res-M) is used. The difference w.r.t. the previous rule is that now z appears in the free names of $\Lambda\downarrow_{\setminus\{z\}}$ and thus it is extruded and added to Γ_E.

Rule STRUCT) Changing the starting agent from P to Q with $P \equiv Q$ has no effect on the allowed transitions, since equivalent agents are mapped into equivalent graphs and the corresponding productions coincide (note that the transitions for graphs are defined directly on the equivalence classes). The same holds also for the result of the transition. □

The mapping can also be extended to deal with match and mismatch operators, since they can be distributed on sequential processes and embedded in the labels of edges. After that, however, the rules of MSHR must be extended to perform checks on equality or inequality of names. We will not deal further with this issue since we want to stick to the standard semantics of MSHR.

We present now the translation of a sample transition.

Example 2.1. *Let us consider the Fusion transition:*

$$(z)(yz.P|\overline{u}xy.Q|uzw.R) \xrightarrow{\{y=w\}} (yz.P|Q|R)\{x/z\}$$

The translation of the starting process into a graph term is:

$$\nu z\ L_{x_1x_2.P'}(y,z)|L_{\overline{x_1}x_2x_3.Q'}(u,x,y)|L_{x_1x_2x_3.R'}(u,z,w)$$

where P', Q' and R' are such that the processes labeling the edges are the ones from standard decompositions of the corresponding Fusion processes. For simplicity we also suppose that these are closed processes (otherwise the nodes corresponding to their free names should be added to the corresponding edges), thus we can choose $\Gamma = u, x, y, w$ as context. The resulting graph is in Figure 2.1.

Furthermore we have the following productions (the first one is an idle production, the other ones are from Definition 2.3):

$$\begin{aligned} y, z \vdash L_{x_1x_2.P'}(y,z) &\xrightarrow{\text{id}} y, z \vdash L_{x_1x_2.P'}(y,z) \\ u, x, y_1 \vdash L_{\overline{x_1}x_2x_3.Q'}(u,x,y_1) &\xrightarrow{(u,out_2,\langle x,y_1\rangle),\text{id}} u, x, y_1 \vdash [\![Q]\!]^F_{SHR} \\ u_1, z_1, w \vdash L_{x_1x_2x_3.R'}(u_1,z_1,w) &\xrightarrow{(u_1,in_2,\langle z_1,w\rangle),\text{id}} u_1, z_1, w \vdash [\![R]\!]^F_{SHR} \end{aligned}$$

Since the three productions have disjoint sets of names we can apply two times rule (par-M) to combine them, obtaining:

$$\begin{aligned} &y, z, u, x, y_1, u_1, z_1, w \vdash L_{x_1x_2.P'}(y,z) | L_{\overline{x_1}x_2x_3.Q'}(u,x,y_1) | L_{x_1x_2x_3.R'}(u_1,z_1,w) \\ &\qquad \xrightarrow{(u,out_2,\langle x,y_1\rangle),(u_1,in_2,\langle z_1,w\rangle),\text{id}} \\ &y, z, u, x, y_1, u_1, z_1, w \vdash L_{x_1x_2.P'}(y,z) | [\![Q]\!]^F_{SHR} | [\![R]\!]^F_{SHR} \end{aligned}$$

Now we have to merge corresponding names, using rule (merge-M) with renaming $\sigma = \{y_1/y, u_1/u, z_1/z\}$. This causes the synchronization between the two actions performed on u and u_1, since these are complementary actions. We get a renaming $\rho = \{x/z, y/w\}$ (but we could have chosen also different representatives for the equivalence classes). The obtained transition is:

$$\begin{aligned} &u, x, y, z, w \vdash L_{x_1x_2.P'}(y,z) | L_{\overline{x_1}x_2x_3.Q'}(u,x,y) | L_{x_1x_2x_3.R'}(u,z,w) \xrightarrow{(u,\tau,\langle\rangle),\{x/z,y/w\}} \\ &\qquad \to u, x, y \vdash (L_{x_1x_2.P'}(y,z) | [\![Q]\!]^F_{SHR} | [\![R]\!]^F_{SHR})\{x/z, y/w\} \end{aligned}$$

As a last step, we have to bind z using rule (res-M):

$$\begin{aligned} &u, x, y, w \vdash \nu z\ L_{x_1x_2.P'}(y,z) | L_{\overline{x_1}x_2x_3.Q'}(u,x,y) | L_{x_1x_2x_3.R'}(u,z,w) \xrightarrow{(u,\tau,\langle\rangle),\{y/w\}} \\ &\qquad \to u, x, y \vdash (L_{x_1x_2.P'}(y,z) | [\![Q]\!]^F_{SHR} | [\![R]\!]^F_{SHR})\{x/z, y/w\} \end{aligned}$$

Notice that if we would have chosen z as representative instead of x, then this step would not have been allowed. The transition is the desired one, since $\{y/w\}$ is a substitutive effect of $\{y = w\}$.

Figure 2.2 shows how the mapping allows to give a visual representation to Fusion computations (G_Q and G_R are the graphs given by $[\![Q]\!]^F_{SHR}$ and $[\![R]\!]^F_{SHR}$ respectively).

Until now, there is still a mismatch between the SHR framework, which allows to execute different synchronizations in different parts of the graph inside the same transition, and Fusion Calculus, which forces a strict interleaving among synchronizations.

In the following two sections we work to bridge that gap, in particular we introduce interleaving Milner SHR (Section 2.2), which is an SHR framework where interleaving is forced and which characterizes the image of the mapping from Fusion Calculus. On the opposite side, in Section 2.3 we present a concurrent semantics for Fusion Calculus, defined through the mapping, and accounting also for the missing transitions.

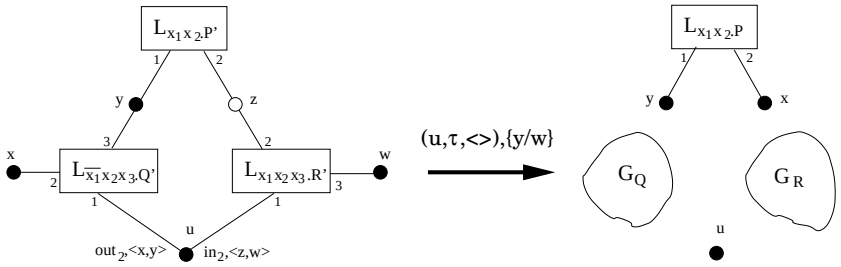

Figure 2.2: Graphical representation of a Fusion transition.

2.2 Interleaving Milner SHR

In this section we introduce interleaving Milner SHR (also referred as iMSHR), a subset of Milner SHR allowing only transitions which are "interleaving", i.e., where just one action can happen: either a communication action or a fusion action.

The main aim of the introduction of iMSHR is to have a strict correspondence with Fusion Calculus, and more in general to show how to deal with the problem of forcing interleaving in a framework where distribution and the absence of a centralized control push naturally toward concurrent semantics. The same approach can be applied also when different synchronization models are used, but we concentrate here on Milner synchronization since it is the one involved in the mapping from Fusion Calculus to SHR.

We give two characterizations of iMSHR, one as stand-alone framework, defined by restricting the allowed kind of productions w.r.t. standard SHR and by giving a suitable set of inference rules to compose them, and the other one as subset of MSHR. The two different characterizations are proved equivalent in Theorem 2.2.

In order to have a more direct connection with Fusion Calculus, we drop the τ action, using ϵ instead. Notice that this can not be done in the standard framework, where distinguishing τ from ϵ allows to recognize the nodes where a synchronization has already happened, and thus no further synchronization can be performed. Instead, in the interleaving framework, each transition contains just one synchronization, and this constraint can be imposed directly by the structure of the inference rules.

We give now the formal definition of interleaving transition.

Definition 2.4 (Interleaving transitions). *An SHR transition* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *is an* interleaving transition *iff it is of one of the two following kinds:*

communication transitions: *they have* $\Lambda(x) = (\epsilon, \langle\rangle)$ *for each* $x \in \Gamma$ *but (at most) one and they have* $\pi = id$*;*

fusion transitions: *they have* $\Lambda(x) = (\epsilon, \langle\rangle)$ *for each* $x \in \Gamma$.

The following inference rules are used to combine interleaving productions.

Definition 2.5 (Inference rules for interleaving Milner SHR). *The admissible behaviours of interleaving Milner SHR are defined by the following inference rules.*

$$\textit{(par-iM)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma \vdash G_1'}{\Gamma \vdash G_1|G_1' \xrightarrow{\Lambda,\pi} \Phi \vdash G_2|G_1'\pi}$$

$$\textit{(com-iM)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\mathrm{id}} \Phi \vdash G_2 \qquad \Gamma \vdash G_1' \xrightarrow{\Lambda',\mathrm{id}} \Phi' \vdash G_2'}{\Gamma \vdash G_1|G_1' \xrightarrow{\Lambda_\epsilon,\pi} \Phi'' \vdash \nu U\ (G_2|G_2')\rho}$$

where:

1. $\exists y \in \Gamma.\forall x \in \Gamma \setminus \{y\}.\Lambda(x) = \Lambda'(x) = (\epsilon, \langle\rangle) \wedge \Lambda(y) = (a, \vec{z}) \wedge \Lambda'(y) = (\overline{a}, \vec{z}')$
2. $\forall x \in \Gamma.\Lambda_\epsilon(x) = (\epsilon, \langle\rangle)$
3. $\rho = \mathrm{mgu}(\{\vec{z} = \vec{z}'\})$ *where* ρ *maps names to representatives in* Γ *whenever possible*
4. $\pi = \rho|_\Gamma$
5. $U = (\Phi \cup \Phi')\rho \setminus \Phi''$

$$\textit{(merge-iM)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2\sigma\rho}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

6. $\rho = \mathrm{mgu}(\{x\sigma = y\sigma | x\pi = y\pi\})$
7. $\Lambda'(z) = \begin{cases} (\Lambda(x))\sigma\rho & \text{if } x\sigma = z \wedge \Lambda(x) \neq (\epsilon, \langle\rangle) \\ (\epsilon, \langle\rangle) & \text{otherwise} \end{cases}$
8. $\pi' = \rho|_{\Gamma\sigma}$

$$\textit{(res-iM)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda|_\Gamma,\pi|_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

9. $(\exists y \in \Gamma.x\pi = y\pi) \Rightarrow x\pi \neq x$
10. $\mathrm{act}_\Lambda(x) = \epsilon$
11. $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda|_\Gamma), Z = \emptyset$ *otherwise*

$$\textit{(new-iM)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,\epsilon,\langle\rangle)\},\pi} \Phi, x \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$.

With respect to the rules in Definition 1.24, here rule (par-iM) allows to add only idle parts of the graph. Also, note that the two contexts Γ must be equal, but this is not a restriction since rule (new-iM) can be used to add names to contexts. Rule (com-iM) allows the reaction of two communication transitions to form a fusion transition. Rule (merge-iM) is used for non injective renamings: it is necessary when two tentacles of one edge performing an action are connected to the same node. Rules (res-iM) and (new-iM) are the standard ones, apart from the fact that τ is no more used.

Notation. *We write* $\mathcal{P} \Vdash_{iM} \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *iff* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *can be obtained from the productions in a set* $\mathcal{P}$ *of interleaving productions using interleaving Milner inference rules.*

The following lemma shows that the inference rules for iMSHR indeed produce interleaving transitions.

Lemma 2.5. *Let* $\mathcal{P}$ *be a set of interleaving productions. If* $\mathcal{P} \Vdash_{iM} \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *then* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *is an interleaving transition.*

Proof. By rule induction. More in details, rule (par-iM) preserves both Λ and π, thus it transforms communication transitions into communication transitions and fusion transitions into fusion transitions. Rule (com-iM) combines two communication transitions into a fusion one. Rule (merge-iM), (res-iM) and (new-iM) just preserve the kind of the transitions. □

We present now a simple application of interleaving Milner SHR to a case which has no relation with Fusion Calculus. Here the interleaving behaviour allows more control on the allowed computations.

Example 2.2. *Let us consider as example the problem of growing a ring graph, as done in the first part of Example 1.1. Instead of using the simple context free production used there, we use a couple of synchronizing communication productions:*

$$x, y \vdash R(x,y) \xrightarrow{(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x,z)|R(z,y)$$
$$x, y \vdash R(x,y) \xrightarrow{(x,\overline{a},\langle\rangle),\mathrm{id}} x, y \vdash R(x,y)$$

Let us consider as starting graph a four elements ring:

$$x \vdash \nu y, z, v\ R(x,y)|R(y,z)|R(z,v)|R(v,x)$$

We can take two suitably renamed instances of the productions and then apply rule (new-iM) two times for each production to get:

$$x,y,z,w \vdash R(x,y) \xrightarrow{(y,a,\langle\rangle),\mathrm{id}} x,y,z,w \vdash \nu t\ R(x,t)|R(t,y)$$
$$x,y,z,w \vdash R(y,z) \xrightarrow{(y,\overline{a},\langle\rangle),\mathrm{id}} x,y,z,w \vdash R(y,z)$$

Since the two transitions have the same context we can apply rule (com-iM) to get:

$$x,y,z,w \vdash R(x,y)|R(y,z) \xrightarrow{\mathrm{id}} x,y,z,w \vdash \nu t\ R(x,t)|R(t,y)|R(y,z)$$

We can then apply rule (par-iM) to add the idle part of the graph:

$$x,y,z,w \vdash R(x,y)|R(y,z)|R(z,v)|R(v,x) \xrightarrow{\mathrm{id}} \\ \to x,y,z,w \vdash \nu t\ R(x,t)|R(t,y)|R(y,z)|R(z,v)|R(v,x)$$

and finally rule (res-iM) (three times) to bind nodes y, z and w:

$$x \vdash \nu y,z,w\ R(x,y)|R(y,z)|R(z,v)|R(v,x) \xrightarrow{\mathrm{id}} \\ \to x,y,z,w \vdash \nu t,y,z,w\ R(x,t)|R(t,y)|R(y,z)|R(z,v)|R(v,x)$$

Note that using the interleaving inference rules we can have only transitions that add at most one new edge to the ring, while with the same productions but with the standard rules we can add many edges concurrently (in particular, each pair of adjacent edges can synchronize to add a new edge).

We now consider again the Fusion transition of Example 2.1, in order to emphasize the differences between the two sets of inference rules.

Example 2.3. *Let us consider the Fusion transition of Example 2.1:*

$$(z)(yz.P|\overline{u}xy.Q|uzw.R) \xrightarrow{\{y=w\}} (yz.P|Q|R)\{x/z\}$$

and the corresponding graph term:

$$\nu z\ L_{x_1x_2.P'}(y,z)|L_{\overline{x_1}x_2x_3.Q'}(u,x,y)|L_{x_1x_2x_3.R'}(u,z,w)$$

with context $\Gamma = u,x,y,w$ (see Figure 2.1).

Here we use as productions:

$$u,x,y \vdash L_{\overline{x_1}x_2x_3.Q'}(u,x,y) \xrightarrow{(u,out_2,\langle x,y\rangle)} u,x,y \vdash [\![Q]\!]^F_{SHR}$$
$$u,z,w \vdash L_{x_1x_2x_3.R'}(u,z,w) \xrightarrow{(u,in_2,\langle z,w\rangle)} u,z,w \vdash [\![R]\!]^F_{SHR}$$

We can use rule (new-iM) (by applying it twice to each production) to make the two contexts equal:

$$u,x,y,z,w \vdash L_{\overline{x_1}x_2x_3.Q'}(u,x,y) \xrightarrow{(u,out_2,\langle x,y\rangle)} u,x,y,z,w \vdash [\![Q]\!]^F_{SHR}$$
$$u,x,y,z,w \vdash L_{x_1x_2x_3.R'}(u,z,w) \xrightarrow{(u,in_2,\langle z,w\rangle)} u,x,y,z,w \vdash [\![R]\!]^F_{SHR}$$

We can then apply in sequence rules (com-iM), (par-iM) and (res-iM) to have respectively:

$$u, x, y, z, w \vdash L_{\overline{x_1}x_2x_3.Q'}(u, x, y) | L_{x_1x_2x_3.R'}(u, z, w) \xrightarrow{\{x/z, y/w\}} u, x, y \vdash (\llbracket Q \rrbracket^F_{SHR} | \llbracket R \rrbracket^F_{SHR})\{x/z, y/w\}$$

$$u, x, y, z, w \vdash L_{x_1x_2.P'}(y, z) | L_{\overline{x_1}x_2x_3.Q'}(u, x, y) | L_{x_1x_2x_3.R'}(u, z, w) \xrightarrow{\{x/z, y/w\}} u, x, y \vdash (L_{x_1x_2.P'}(y, z) | \llbracket Q \rrbracket^F_{SHR} | \llbracket R \rrbracket^F_{SHR})\{x/z, y/w\}$$

$$u, x, y, w \vdash \nu z\ L_{x_1x_2.P'}(y, z) | L_{\overline{x_1}x_2x_3.Q'}(u, x, y) | L_{x_1x_2x_3.R'}(u, z, w) \xrightarrow{\{y/w\}} u, x, y \vdash (L_{x_1x_2.P'}(y, z) | \llbracket Q \rrbracket^F_{SHR} | \llbracket R \rrbracket^F_{SHR})\{x/z, y/w\}$$

As expected, the derived transition is almost the same of Example 2.1 (see Figure 2.2), but it uses ϵ instead of τ on node u.

As shown in the example, the set of inference rules for iMSHR forces a different style of derivation w.r.t. the one for MSHR. First, just the non idle productions have to be considered, and instead of choosing instances with disjoint sets of names, instances with the same names must be chosen. Names appearing in just one of the productions must be added to the other one using rule (new-iM). Rule (merge-iM) must be used if a rewritten edge is attached many times to the same node. If the transition is the synchronization of two communication productions, then rule (com-iM) is used to combine them. Finally, the rest of the graph (which is idle) can be added using rules (par-iM) and/or (new-iM). Restrictions can be introduced as usual, using rule (res-iM).

We can state the following correspondence theorem between iMSHR and MSHR. Essentially, iMSHR is the restriction of MSHR to the case of just one edge performing a non idle production, or two edges synchronizing their communication productions on a shared node.

Theorem 2.2 (Correspondence between iMSHR and MSHR). *Given a set of interleaving productions $\mathcal{P}$ and a starting graph $\Gamma \vdash G_1$ we have $\mathcal{P} \Vdash_{iM} \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ iff we have $\mathcal{P} \Vdash_M \Gamma \vdash G_1 \xrightarrow{\Lambda',\pi} \Phi \vdash G_2$ where $\Lambda(x) = (\epsilon, \langle\rangle)$ if $\Lambda'(x) = (\tau, \langle\rangle)$ and $\Lambda(x) = \Lambda'(x)$ otherwise, and the second derivation uses either at most one non idle production or two communication productions which synchronize their non ϵ actions on a shared node.*

Proof. The proof is in two steps: first of all we have to prove that each iMSHR rule can be simulated using MSHR rules (apart from the possible exchange of ϵ for τ), and this will prove the forward implication. Note, in fact, that iMSHR inference rules can not generate other kinds of transitions (since these are exactly the interleaving transitions, and thanks to Lemma 2.5). For the backward implication we have to prove that all the transitions obtained with MSHR inference rules using either at most one non idle production or two communication productions which synchronize their non ϵ actions on a shared node can be generated by the iMSHR inference rules.

Let us consider the different iMSHR rules to prove the first part.

Rule (par-iM): let us consider a bijective renaming σ that maps all the names in Γ to names not in Γ. Using MSHR rules applied to idle productions we can easily derive an idle transition for the graph $\Gamma\sigma \vdash G_1'\sigma$. We can then apply rule (par-M) to compose it with the transition $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$, and then rule (merge-M) with the renaming σ^{-1} that is the inverse of σ. Notice that here $\rho = \pi$, and since it is idempotent we can suppose that it is applied only to G_1'. Also, no real synchronization is performed, since for each node we have two actions, and one of them (the one from the idle transition) is always ϵ. Thus Λ does not change. This proves the thesis.

Rule (com-iM): again, we have to take a renamed version with fresh names of the second transition, which exists thanks to Lemma 1.2. Again, this allows to apply rule (par-M) and then rule (merge-M) with the inverse renaming. By hypothesis the only two non trivial actions are performed on the same node, thus this is the only real synchronization that is performed by rule (merge-M). The only merges produced by ρ are those from the synchronization as required, since the two premises carry just identity renamings. The resulting Λ contains a τ as only non ϵ action, but this is replaced by ϵ too when moving to the interleaving framework.

Rule (merge-iM): this is just a subcase of MSHR rule (merge-M), when at most one non ϵ action exists (thus no real synchronization can happen).

Rules (res-iM) and (new-iM): these correspond exactly to the MSHR rules (res-M) and (new-M).

Let us consider the second implication. Let us consider the case of at most one non idle production first. Rule (par-M) can be applied in this case just to add an idle part of the graph, but nodes can be added in the iMSHR framework using rule (new-iM), and edges can then be added using rule (par-iM). MSHR rules (merge-M), (res-M) and (new-M) reduce to their instances (merge-iM), (res-iM) and (new-iM). We can consider now the case of two communication productions. In that case, applying rule (par-M) produces a non interleaving transition. To get back to the case of productions synchronizing on the same node, rule (merge-M) must be used. The total effect of these two rules is mimicked by the only rule (com-iM), with the only difference that first the two contexts must be made equal (but it is easy to see that Lemma 1.2 holds also for the iMSHR inference rules). After that we get a transition that is analogous to a fusion transition, apart for a τ on the node where the synchronization has been performed, and the above reasoning applies. □

Now we can consider again the case of Fusion Calculus, and show that iMSHR characterizes the image of the mapping. First of all notice that all the Fusion productions are interleaving productions. We can restate here, in the new framework, the correctness result of Theorem 2.1.

Theorem 2.3 (Correctness). *Let $\mathcal{P}_F$ be the set of Fusion productions. For each Fusion process P and each $\Gamma \supseteq \mathrm{fn}([\![P]\!]^F_{SHR})$ if $P \xrightarrow{\alpha} P'$ and $\Gamma \cap \mathrm{bn}(\alpha) = \emptyset$ then:*

- *if α is a communication action, $\mathcal{P}_F \Vdash_{iM} \Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda_\alpha, id} \Gamma, \Gamma_E \vdash [\![P']\!]^F_{SHR}$ where $\Lambda_\alpha(x) = [\![\alpha]\!]^F_{SHR}$ if x is the subject of α, $\Lambda_\alpha(x) = (\epsilon, \langle\rangle)$ otherwise and $\Gamma_E = \mathrm{bn}(\alpha)$;*

- *if α is a fusion action, for each substitutive effect π of α, $\mathcal{P}_F \Vdash_{iM} \Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Gamma\pi \vdash [\![P'\pi]\!]^F_{SHR}$ where $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \Gamma$.*

Proof. The proof is by induction on the derivation of $P \xrightarrow{\alpha} P'$, and it just composes the correctness result of Theorem 2.1 with the proof of the correspondence between iMSHR and MSHR of Theorem 2.2. □

The main improvement obtained thanks to iMSHR is that now we can have also a completeness result.

Theorem 2.4 (Completeness). *Let $\mathcal{P}_F$ be the set of Fusion productions. For each Fusion process P if $\mathcal{P}_F \Vdash_{iM} \Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Phi \vdash G$ for some $\Gamma \supseteq \mathrm{fn}([\![P]\!]^F_{SHR})$ and the transition is not idle then there exists P' such that:*

- *either $P \xrightarrow{\alpha} P'$, $\Lambda(x) = [\![\alpha]\!]^F_{SHR}$ if x is the subject of α, $\Lambda(x) = (\epsilon, \langle\rangle)$ otherwise, $\pi = id$, $[\![P']\!]^F_{SHR} = G$ and $\Phi = \Gamma, \Gamma_E$ where $\Gamma_E = \mathrm{bn}(\alpha)$;*

- *or $P \xrightarrow{\phi} P'$, π is a substitutive effect of ϕ, $\Lambda(x) = (\epsilon, \langle\rangle)$ for each x in Γ, $[\![P'\pi]\!]^F_{SHR} = G$ and $\Phi = \Gamma\pi$.*

Proof. The proof is by induction on the derivation of $\Gamma \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Phi \vdash G$.

Production) If the transition is a production, since it is not idle by hypothesis, it must be one of the productions defined in Definition 2.3. For each of those productions to exist, the Fusion process P must have a particular structure: in particular it must be a sum containing a summand of the form $\alpha.P'$. Thus the required Fusion transition exists thanks to Fusion rules PREF and SUM.

Rule (par-iM)) Let us consider the case of a fusion transition, which falls in the second case of the theorem (the case of communication transitions, which falls in the first case, is simpler). We know that $\Gamma \vdash G_1|G'_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2|G'_1\pi$ with $G_1|G'_1 = [\![P]\!]^F_{SHR}$. Thus P must be of the form $P_1|P'_1$ with $[\![P_1]\!]^F_{SHR} = G_1$ and $[\![P'_1]\!]^F_{SHR} = G'_1$. From the hypothesis of rule (par-iM) there is a transition of the form $\Gamma \vdash [\![P_1]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$. By inductive hypothesis we have a transition $P_1 \xrightarrow{\phi} P_2$ and π is a substitutive effect of ϕ, $\Lambda(x) = (\epsilon, \langle\rangle)$ for each x in Γ, $[\![P_2\pi]\!]^F_{SHR} = G_2$ and $\Phi = \Gamma\pi$. Using Fusion rule PAR we have $P_1|P'_1 \xrightarrow{\phi} P_2|P'_1$, which is the required transition since $[\![(P_2|P'_1)\pi]\!]^F_{SHR} = [\![P_2\pi]\!]^F_{SHR}|[\![P'_1\pi]\!]^F_{SHR} = G_2|G'_1\pi$.

Rule (com-iM)) By inductive hypothesis we have two transitions of the form $P_1 \xrightarrow{\alpha} P_2$ and $P'_1 \xrightarrow{\alpha'} P'_2$ with $[\![P_1]\!]^F_{SHR} = G_1$, $[\![P_2]\!]^F_{SHR} = G_2$, $[\![P'_1]\!]^F_{SHR} = G'_1$ and $[\![P'_2]\!]^F_{SHR} = G'_2$. Also, α and α' must be an input and an output, with the same arity and performed on the same name. Let us consider first the case where the two

actions are free actions. We can apply Fusion rule COM to get $P_1|P_1' \xrightarrow{\phi} P_2|P_2'$. This is the Fusion transition that corresponds to the conclusion of rule (com-iM) since $[\![P_1|P_1']\!]^F_{SHR} = G_1|G_1'$, π is a substitutive effect of ϕ and $[\![(P_2|P_2')\pi]\!]^F_{SHR} = (G_2|G_2')\rho$ since $\pi = \rho$ thanks to Lemma 2.4. For the same reason U is empty.

In order to prove the general case, we have to show that any transition that can be derived using iMSHR inference rules can also be derived by applying rule (com-iM) before any application of rule (res-iM). More in general, we have to show that all the applications of rule (res-iM) can be moved to the end of the proof without changing the result. Since the proof of this fact is simple, but long, we present just one interesting case as example. We consider the application of a rule (res-iM) restricting one of the parameters, let us call it x, of the first premise of rule (com-iM). If (res-iM) is applied first, then x appears in the label of the premise, and no restriction for x is added to the final graph. When applying rule (com-iM) x is neither in Γ nor in the final label, thus it is in U and the restriction is added. Also, it is not used as representative, thus if it is in a non trivial equivalence class it is just substituted. By applying rule (com-iM) first, no restriction is added since x is free in the LHS, but when applying rule (res-iM), since x no more appears in the label, then the restriction is added also to the RHS. Also, only transitions where x is not a representative of a non trivial equivalence class in π can be chosen (and, if it is not a representative, it is substituted). Thus the two derivations produce the same result.

Rule (merge-iM)) One can simply prove by rule induction on Fusion derivations that given a renaming σ and a transition $P \xrightarrow{\alpha} P'$, then there is also a transition $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. This is enough to prove the thesis for communication transitions, where $\pi = \mathrm{id}$ and thus also $\pi' = \rho = \mathrm{id}$. To prove the theorem also for fusion transitions we have to show that ρ is a substitutive effect of $\phi\sigma$, that is $\rho = \mathrm{mgu}(\phi\sigma)$ thanks to Lemma 2.3. By definition $\mathrm{mgu}(\phi\sigma) = \mathrm{mgu}(\{x\sigma = y\sigma | x = y \in \phi\}) = \mathrm{mgu}(\{x\sigma = y\sigma | x\pi = y\pi\})$ where the second step is from the definition of substitutive effect. Since this is exactly the definition of ρ then the thesis holds.

Rule (res-iM)) Different cases have to be considered here. If the restricted name z is not a name appearing in the label, then $Z = \{z\}$ and this corresponds to an application of rule PASS. Notice that in that case the restriction of the iMSHR observation has no effect on the corresponding Fusion label. Let us consider now the case of z appearing in the label. In any case, z can not be the subject of the non trivial synchronization performed by the transition (if any exists), since the action performed on z must be ϵ. Thus, for communication transitions, it can just be one of the parameters. In that case, $Z = \emptyset$, z is added to Γ_E and an extrusion must be performed in the Fusion setting. This is done by rule OPEN. The last case is the one dealing with a fusion transition, where z is merged. In that case z can not be a representative, and a binding $\{x/z\}$ appears in π. Similarly, $x = z$ appears in ϕ, but this equality is removed by going to $\phi \setminus z$, and the corresponding binding is deleted from π by the operation of domain restriction, thus preserving the

Fusion	MSHR	Fusion	MSHR
Process	Graph	Transition	Interleaving trans.
Sequential process	Edge	Parallel comp.	Parallel comp.
Name	Node	Scope	Restriction
Prefix execution	Production	0	*Nil*

Table 2.1: Comparison between Fusion and MSHR.

correspondence between the two labels. In the Fusion setting the binding is applied to the final process, but this is not required in the SHR setting, where π has already been applied.

Rule (new-iM)) Here the same Fusion transition corresponds to both the premise and the conclusion, the only thing that changes is in fact the chosen context Γ. □

We end this section with a simple schema (Table 2.1) summarizing the correspondence between Fusion Calculus and MSHR. The correspondence is built on the isomorphism that maps parallel composition into parallel composition and scope into restriction. Thus a process is translated into a graph whose edges stand for sequential processes and whose nodes are names. From a dynamic point of view, prefix executions give rise to productions, and whole Fusion transitions become interleaving MSHR transitions.

2.3 A concurrent semantics for Fusion Calculus

In this section we present a simple application of the mapping from Fusion Calculus to MSHR, namely we derive essentially for free a concurrent operational semantics for Fusion Calculus. This semantics is obtained by mapping Fusion processes into graphs, and then computing their behaviour using MSHR inference rules instead of the iMSHR ones. In particular, we consider here all the transitions instead of just the interleaving ones. This approach will be used also later, since it provides a simple way to export results from SHR to Fusion Calculus.

From a different point of view, this semantics is an approach to bridge the gap between MSHR and Fusion Calculus which is dual w.r.t. the one of the previous section: instead of constraining MSHR, here we generalize Fusion Calculus.

The semantics that we present can be considered a set semantics, but the term set is not referred to the representation of the state (which is however a multiset of edges) as in [Eng93], but to the structure of the labels. In fact, our observation includes a set of performed actions. There is no need to use multisets of actions since each action has a different subject. This happens because we force interleaving inside nodes.

Most of the concurrent semantics presented in the literature for process calculi are built on top of a sequential semantics, and they are based on a suitable equiva-

lence between computations that makes equal all the computations that differ only on the ordering of some independent transitions. This can be done directly at the level of process calculi [BC94, CFM90], or obtained via a mapping into a framework with a developed concurrency theory such as Petri Nets [BG95] or graph transformation [MP95].

Our approach instead adds further transitions to the normal LTS. Each of these transitions concurrently executes many independent actions. This approach allows to analyze standard computational steps (with more complex labels) instead of computations, and it allows to distinguish between nondeterminism and parallelism.

Here we present just a simple semantics, straightforwardly derived from MSHR semantics, and we give some insights on its main features. Some variations of this semantics, together with a detailed analysis of the effect of this semantics on the observational properties of Fusion processes will be presented in Section 9.1, in the part of the thesis devoted to the observational semantics.

Definition 2.6 (Concurrent operational semantics). *The* concurrent operational semantics for Fusion Calculus *is given by the LTS with Fusion processes as states, SHR labels and transition relation $\rightarrow_c$ defined by:*

$$P \xrightarrow{\Lambda,\pi}_c P' \text{ iff } \mathcal{P}_F \Vdash \mathrm{fn}([\![P]\!]^F_{SHR}) \vdash [\![P]\!]^F_{SHR} \xrightarrow{\Lambda,\pi} \Gamma \vdash G \text{ and } [\![P']\!]^F_{SHR} = G$$

where $\mathcal{P}_F$ is the set of Fusion productions.

An easy corollary of Theorem 2.1 guarantees that this concurrent semantics indeed extends the usual semantics, in the sense that for each standard transition we have a corresponding transition in the concurrent LTS.

Corollary 2.1 (Preservation of transitions). *For each Fusion process P if $P \xrightarrow{\alpha} P'$ then:*

1. *if α is a communication action, $P \xrightarrow{\Lambda_\alpha,\mathrm{id}}_c P'$ where $\Lambda_\alpha(x) = [\![\alpha]\!]^F_{SHR}$ if x is the subject of α, $\Lambda_\alpha(x) = (\epsilon, \langle\rangle)$ otherwise;*

2. *if α is a fusion action, for each substitutive effect π of α, $P \xrightarrow{\Lambda,\pi}_c P'\pi$ where either $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \mathrm{fn}(P)$ or there exists $y \in \mathrm{fn}(P)$ such that $\Lambda(y) = (\tau, \langle\rangle)$ and $\Lambda(x) = (\epsilon, \langle\rangle)$ for each $x \in \mathrm{fn}(P) \setminus \{y\}$.*

Proof. The proof is straightforward, given Theorem 2.1 and the definition of the concurrent LTS (Definition 2.6). □

As an example we analyze here the concurrent semantics of the Fusion process from Example 2.1.

Example 2.4.
Let us consider the Fusion process $(z)(yz.P|\overline{u}xy.Q|uzw.R)$ from Example 2.1. Using standard Fusion semantics we can have the two following computations:

$$\begin{array}{lll}(z)(yz.P|\overline{u}xy.Q|uzw.R) & \xrightarrow{\{y=w\}} & (yz.P|Q|R)\{x/z\} \\ & \xrightarrow{yx} & (P|Q|R)\{x/z\}\end{array}$$

$$\begin{array}{lll}(z)(yz.P|\overline{u}xy.Q|uzw.R) & \xrightarrow{(z)yz} & (P|\overline{u}xy.Q|uzw.R) \\ & \xrightarrow{\{x=z,y=w\}} & P|Q|R\end{array}$$

These are mimicked by the following computations in the concurrent LTS (remember that the labels contain an action for each free name of the process, and actions not written are of the form $(x, \epsilon, \langle\rangle)$ for some free name x).

$$\begin{array}{lll}(z)(yz.P|\overline{u}xy.Q|uzw.R) & \xrightarrow{(u,\tau,\langle\rangle),\{y/w\}}_c & (yz.P|Q|R)\{x/z, y/w\} \\ & \xrightarrow{(y,in_1,\langle x\rangle),\mathrm{id}}_c & (P|Q|R)\{x/z, y/w\}\end{array}$$

$$\begin{array}{lll}(z)(yz.P|\overline{u}xy.Q|uzw.R) & \xrightarrow{(y,in_1,\langle z\rangle),\mathrm{id}}_c & (P|\overline{u}xy.Q|uzw.R) \\ & \xrightarrow{(u,\tau,\langle\rangle),\{x/z,y/w\}}_c & (P|Q|R)\{x/z, y/w\}\end{array}$$

Also, different representatives can be chosen when computing the fusion renaming (but, in the first transition, z can not be a representative since it is bound in the starting process), and an idle transition to the process itself is always available.

In the concurrent semantics we can also execute both the actions in the same transition.

$$(z)(yz.P|\overline{u}xy.Q|uzw.R) \quad \xrightarrow{(u,\tau,\langle\rangle),(y,in_1,\langle x\rangle),\{x/z,y/w\}}_c \quad (P|Q|R)\{x/z, y/w\}$$

As shown by the example and by the correspondence theorem, there are many differences between the concurrent semantics and the standard semantics. Some of them are just syntactical, others have a more semantic meaning. Also, different concurrent semantics can be defined, removing the differences that are simply due to the structure of SHR labels and LTS.

We will analyze now those differences, and we will also come back to that topic later, when we will consider the observational semantics (see Section 9.1):

concurrent vs interleaving: this is clearly the main difference. We think that having a concurrent semantics can be useful, since it allows to analyze the degree of parallelism of a system. For instance, the two processes $xy.zw.0 +$

$zw.xy.0$ and $xy.0|zw.0$ have the same interleaving transitions, but the second one has also a transition corresponding to the concurrent execution of the two prefixes, emphasizing its higher degree of parallelism. Note that prefixes can be executed concurrently only if they have different subjects (but two prefixes with the same subject can clearly synchronize, if they are complementary).

located τ: in the concurrent semantics synchronizations on a free channel x produce a τ action on x, thus the channel on which the synchronization has been performed can be distinguished. The τ action performed on a bound channel is instead discarded by the rule for restriction. Having the τ on x is necessary in the concurrent semantics since the τ guarantees that no other action can be performed during the same transition on the same channel. This feature can also be added to the interleaving semantics, where distinguishing synchronizations performed on different channels can be useful for cost or performance reasons. We will come back to that topic later, when discussing different synchronization models (see Section 7.2).

renamings vs fusions: in the concurrent semantics a renaming π is used instead of a fusion ϕ. Renamings add to the corresponding fusions the choice of a representative for each equivalence class. However the different allowed choices can always be easily computed, thus the two approaches are equivalent (actually, this is the difference between Fusion Calculus and Update Calculus [PV97], but in the latter the choice of $\{x/y\}$ instead of $\{y/x\}$ can not be changed during the derivation, and this makes renamings and fusions no more equivalent). However, also when using fusions in the label, a substitutive effect must be chosen in order to apply the fusion to the resulting process.

application of fusion renamings: in the concurrent LTS the whole fusion in the label is applied to get the new process, instead of being applied only for the part concerning restricted names as done by the standard LTS. The choice done in the standard LTS is due to implementation reasons, since applying renamings at each step may be difficult. However this choice requires a particular definition of bisimulation that essentially applies the renaming (see Definition 1.16), and it produces, in our opinion, less intuitive computations. Notice for instance that in Example 2.4 the two computations in the standard LTS produce different final processes, while the results of the two corresponding computations in the concurrent LTS coincide.

free names: in the concurrent semantics labels contain an action for each free name, thus the whole set of free names is observed. This difference can be avoided by defining a slightly different concurrent semantics.

idle transitions: in the concurrent LTS idle transitions are available for each process. This difference can be avoided by defining a slightly different concurrent semantics.

extrusions: in the concurrent semantics extrusions are not explicitly marked in the label. This is usually unnecessary in the SHR framework, where names are tracked by judgements, thus extruded names are names not appearing in the judgement of the source graph. However adding this information to the label causes no problem, and this can be a valuable information to reason about the observable behaviour of processes (see Section 9.1).

other differences: there are other differences, for instance in the syntax of actions, but these differences have no real impact on the LTS since they are just due to the different syntaxes used in the two frameworks to carry the same information.

Chapter 3

Milner synchronization vs Hoare synchronization in SHR

In this chapter we present a detailed comparison between the expressiveness of Milner and Hoare synchronizations in the SHR framework. The only comparison in the literature between these two synchronization models we are aware of is [Hoa05, HH05], but it deals with the simple process calculi CCS [Mil82] and CSP [Hoa80], thus neglecting issues such as mobility and distribution. Also, the comparison deals with all the features of the two calculi, which are quite different, while we are mainly interested in analyzing the two synchronization models abstracting away from other differences that can make the comparison more difficult. We have chosen SHR as framework for that comparison for various reasons. First of all, SHR was already equipped with both the synchronization mechanisms. Also, SHR provides a very general framework for this comparison, allowing both mobility and multi-party synchronization. Finally, the links between Milner SHR and Fusion Calculus studied in the last chapter, and between Hoare SHR and Logic Programming to be presented in the next one make this comparison a key ring in the chain of mappings from Fusion Calculus to Logic Programming. Since SHR is mainly aimed at modeling topological reconfigurations of systems (networks, software architectures, processes, ...) specified via productions, our comparison will concentrate on which such reconfigurations can be specified using either Hoare or Milner synchronization model. In other words we are interested in analyzing the reachability relation in the LTS defined by a fixed set of productions in each synchronization model. In particular, we analyze under which conditions the two models are able to specify the same class of reconfigurations, and, when this is not the case, what can be done to bridge the expressiveness gap.

3.1 Expressiveness measures

As already said, we want to study the expressiveness of Hoare and Milner synchronization models in the SHR framework. In general, there exist different possible definitions of expressiveness, and different ways to measure it, concentrating on different aspects of the behaviour of the specified system. The choice of the best expressiveness measure is not unique, but it depends on the intended use of the model. Since we are mainly interested in expressing topological reconfigurations of systems, we choose as aspect to measure the class of reconfigurations that can be expressed by a fixed set of productions together with a synchronization model. It is also useful to fix a set of allowed computations, which may be smaller than the set of all the computations in the LTS. This choice essentially specifies some constraints that must be satisfied by computations, concerning, e.g., their lengths or the kind of rewritten graphs.

The concept of *behaviour* captures the above notion of expressiveness. Intuitively, the behaviour of a set of productions $\mathcal{P}$ on a graph $\Gamma \vdash G$ w.r.t. a synchronization model S is the set of graphs that are the results of "suitable" computations derived using synchronization model S and starting from graph $\Gamma \vdash G$. The choice of which computations are "suitable" determines the kind of behaviours of the system we concentrate on.

Definition 3.1 (Behaviour function).
The function $C\text{-behav}^S(\mathcal{P})(\Gamma \vdash G)$ *is the function that computes the set of graphs reachable from graph* $\Gamma \vdash G$ *using computations in the set* C *derived from the productions in set* $\mathcal{P}$ *using synchronization model* S.

Using the behaviour function we are now able to compare different models, thus defining an expressiveness preorder on the set of pairs (S, C) where S is a synchronization model and C is a set of computations.

Definition 3.2 (Expressiveness preorder $\succeq$). *Let* S_1 *and* S_2 *be synchronization models and let* C_1 *and* C_2 *be sets of computations. The* C_1*-expressiveness of* S_1 *is greater than the* C_2*-expressiveness of* S_2*, written* $(S_1, C_1) \succeq (S_2, C_2)$*, iff there exists a translation function* f *from sets of productions to sets of productions such that for each set of productions* $\mathcal{P}$ *and for each graph* $\Gamma \vdash G$ *the following equation holds:*

$$C_2\text{-behav}^{S_2}(\mathcal{P})(\Gamma \vdash G) = C_1\text{-behav}^{S_1}(f(\mathcal{P}))(\Gamma \vdash G)$$

Intuitively the above equation requires that reconfigurations that can be specified in (S_2, C_2) can be also specified in (S_1, C_1). The translation function f determines, given a set of productions $\mathcal{P}$, which is the set of productions $f(\mathcal{P})$ that specifies the same class of reconfigurations, but in (S_1, C_1) instead of in (S_2, C_2).

The following proposition proves that the expressiveness preorder $\succeq$ is indeed a preorder.

Proposition 3.1. $\succeq$ *is a preorder.*

Proof. Reflexivity is trivial using as translation function f the identity function. Transitivity is trivial too, using as translation function the composition of the two translation functions. □

Notation. *We use H to denote the Hoare synchronization model, and M for Milner one.*

For the moment we consider just three sets of computations, that are the following ones:

1: one-step computations, that is computations composed by one transition (which may also be idle);

***all*:** all possible finite computations, that is computations of any finite length (we consider only finite computations since a graph which is the result of the computation must always be found);

***max*:** maximal computations, that is computations of any finite length such that there is no transition from their final graph $\Gamma \vdash G$ to a different graph (there can be however transitions to the graph $\Gamma \vdash G$ itself).

We think that the above defined sets of computations are interesting for our aim, since set 1 analyzes reconfigurations that can be done in one step, set *all* considers all the possible reconfigurations and set *max* concentrates on the reconfigurations that are obtained by letting the system reconfigure until it reaches a stable (from a topological point of view) state.

Clearly other sets of computations or other measures can be considered, for instance to study the infinite behaviour of the system, but these are less relevant for our aim, and thus these are left for future work.

3.2 The expressiveness of HSHR and MSHR are not comparable

In this section we show that for each of the above defined sets of computations the expressiveness of Hoare and Milner synchronization models are different, but no one is greater than the other, i.e. for each set of computations $C_1, C_2 \in \{1, all, max\}$, both $(H, C_1) \not\succeq (M, C_2)$ and $(M, C_2) \not\succeq (H, C_1)$.

We first study some properties of LTSs of SHR useful for the comparison.

Definition 3.3 (Parallel closure). *We define the following partial order on transitions: $\Gamma \cup \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda', \pi} \Phi \cup \Gamma'\pi \vdash G_2 | G_1'\pi$ is greater than $\Gamma \vdash G_1 \xrightarrow{\Lambda, \pi} \Phi \vdash G_2$ iff $\Gamma' \vdash G_1'$ is a graph and $\Lambda = \Lambda'\restriction_\Gamma$. The partial order is extended to computations by*

requiring the ordering condition to hold between each pair of transitions in the same position in the two computations. A set of SHR computations is parallel closed *iff it is upward-closed under the above defined partial order.*

Intuitively, a set of SHR computations is parallel closed iff we can always add to the rewritten graph $\Gamma \vdash G_1$ another graph $\Gamma' \vdash G'_1$ which stays idle (up to possible fusions of nodes in $\Gamma \cap \Gamma'$ determined by π) and still have an allowed computation. Note that computations of different lengths are never comparable.

Proposition 3.2. *For each set of productions $\mathcal{P}$, the following two sets of MSHR computations derived using productions in $\mathcal{P}$ are parallel closed:*

- 1*, the set of one-step computations;*
- all*, the set of all finite computations.*

Proof. The proposition holds for set 1 since in MSHR it is always possible to use rule (par-M) to add an idle part to the graph (the idle transition can be easily derived by combining idle productions for the edges there, and adding isolated nodes using rule (new-M)). In order to apply rule (par-M), nodes must be chosen distinct w.r.t. the nodes in the main transition. The last step is to apply rule (merge-M) to merge common nodes. Since all the actions performed by the idle transition are of the form $(\epsilon, \langle\rangle)$, rule (merge-M) can be applied, and the synchronizations on the nodes from the main transition are just preserved.

The second part follows by induction since the same approach can be applied to each transition in the computation. □

Intuitively, the above proposition holds because Milner synchronization allows any number of idle participants. The same is not true for Hoare synchronization, because there is a universal quantification on the participants connected to the node where the synchronization is performed, and all those participants are forced to join the synchronization by performing an action which, in general, is not $(\epsilon, \langle\rangle)$.

We can apply the above developed theory to prove a first gap of expressiveness between HSHR and MSHR.

Theorem 3.1. $(M, C_1) \not\geq (H, C_2)$ *for each* $C_1 \in \{1, all\}$ *and each* $C_2 \in \{1, all, max\}$.

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash D'(x)$$

For $C_2 \in \{1, all\}$ we have:
C_2- behav$^H(\mathcal{P})(x \vdash D(x)) = \{x \vdash D(x), x \vdash D'(x)\}$
C_2- behav$^H(\mathcal{P})(x \vdash D(x)|D(x)) = \{x \vdash D(x)|D(x), x \vdash D'(x)|D'(x)\}$
For max-expressiveness instead we have:
max- behav$^H(\mathcal{P})(x \vdash D(x)) = \{x \vdash D'(x)\}$

max- $\mathrm{behav}^H(\mathcal{P})(x \vdash D(x)|D(x)) = \{x \vdash D'(x)|D'(x)\}$
since the idle transitions do not lead to stable states.

Since, for each choice of $C_2 \in \{1, all, max\}$, these HSHR behaviours are not parallel closed, the thesis follows from Proposition 3.2. □

For max-expressiveness of MSHR the above approach can not be applied, since Proposition 3.2 can not be generalized to the set max of maximal computations, as shown by the following example.

Example 3.1. *Let us consider the set of productions $\mathcal{P}$ generated by the only production:*

$$x \vdash D'(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash D(x)$$

We have the two following MSHR behaviours:
max- $\mathrm{behav}^M(\mathcal{P})(x \vdash D(x)) = \{x \vdash D(x)\}$
max- $\mathrm{behav}^M(\mathcal{P})(x \vdash D(x)|D'(x)) = \{x \vdash D(x)|D(x)\}$
that show that the set max of maximal MSHR computations is not parallel closed. Intuitively this is possible since adding a new part to the graph may allow new transitions, which must be taken in order to reach a stable state. Thus the computations required by the parallel closure condition exist in the LTS, but they may not belong to set max.

Thus we need a slightly more refined approach to deal with the case of max-expressiveness.

Theorem 3.2. *$(M, max) \not\succeq (H, C)$ for each $C \in \{1, all, max\}$.*

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash nil$$

For each $C \in \{1, all\}$ we have:
C- $\mathrm{behav}^H(\mathcal{P})(x \vdash D(x)) = \{x \vdash D(x), x \vdash nil\}$
C- $\mathrm{behav}^H(\mathcal{P})(x \vdash D(x)|D'(x)) = \{x \vdash D(x)|D'(x)\}$
For max-expressiveness instead we have:
max- $\mathrm{behav}^H(\mathcal{P})(x \vdash D(x)) = \{x \vdash nil\}$
max- $\mathrm{behav}^H(\mathcal{P})(x \vdash D(x)|D'(x)) = \{x \vdash D(x)|D'(x)\}$
Suppose that we can obtain these behaviours with max MSHR computations. By parallel closure from the first case, a transition from $x \vdash D(x)|D'(x)$ to $x \vdash D'(x)$ should exist. If $x \vdash D'(x)$ has no outgoing transitions to different graphs, then we have a contradiction, since it is not in the behaviour. Otherwise by parallel closure also $x \vdash D(x)|D'(x)$ must have outgoing transitions to different graphs, and so it can not be in the behaviour for maximal computations, as it is. This proves the thesis. □

We prove now that a similar expressiveness gap exists also in the other direction. To this end we will not consider parallel closure, but the effect of the restriction operator on the allowed transitions.

Notice that, if all nodes are free, Milner synchronization can not force productions to be executed together, i.e., each production can always be applied in isolation. Hence, restriction is fundamental for constraining the behaviour of components using Milner synchronization (and this is not a surprise, since even in CCS, which is the simplest used framework based on Milner synchronization, restriction is necessary to reach Turing equivalence [CHM94]).

Notice that in HSHR instead restriction just performs hiding of part of the observation, i.e., no allowed transition can be forbidden by adding restriction operators. This is formalized by the following proposition.

Proposition 3.3. *Given a set of productions $\mathcal{P}$, if $\mathcal{P} \Vdash_H \Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ then $\mathcal{P} \Vdash_H \Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu Z\ G_2$ where Λ', π' and Φ' are subsets of Λ, π and Φ respectively, and $Z = \Phi \setminus \Phi'$.*

This result can be used to prove the desired expressiveness gap.

Theorem 3.3. *$(H, C_1) \not\geq (M, C_2)$ for each $C_1, C_2 \in \{1, all, max\}$.*

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash D'(x)$$

For each $C_2 \in \{1, all\}$ we have:
C_2- behav$^M(\mathcal{P})(x \vdash D(x)) = \{x \vdash D(x), x \vdash D'(x)\}$
C_2- behav$^M(\mathcal{P})(\emptyset \vdash \nu x\ D(x)) = \{\emptyset \vdash \nu x\ D(x)\}$
For max-expressiveness instead we have:
max- behav$^M(\mathcal{P})(x \vdash D(x)) = \{x \vdash D'(x)\}$
max- behav$^M(\mathcal{P})(\emptyset \vdash \nu x\ D(x)) = \{\emptyset \vdash \nu x\ D(x)\}$
Since these behaviours do not satisfy Proposition 3.3, then they can not be obtained using Hoare synchronization (with any set of computations). □

We have thus proved that in the general case we have an incomparability result for all the sets of computations of interest. Next section presents instead some equivalence results, obtained by constraining the class of allowed graphs.

3.3 Reconciling Hoare and Milner synchronizations

Until now we have shown that the expressiveness of HSHR and MSHR are quite different. We concentrate now on a particular class of graphs, and on the computations

internal to that class, and we show that in this particular case MSHR can simulate HSHR (when the same set of computations is used in both the frameworks). To get a similar result in the opposite direction a suitable translation of graphs is needed.

We restrict our attention to *closed 2-shared graphs*. We remember that a graph is closed when its set of free names is empty. The following definition introduces 2-shared graphs.

Definition 3.4 (2-shared graphs). *A graph $\Gamma \vdash G$ is a* 2-shared graph *iff for each node x occurring in G there are exactly two tentacles in G attached to x.*

In practice, each non isolated node in a 2-shared graph can be attached two times to just one edge, or it can be attached one time each to two edges (isolated nodes occur in Γ but not in G). Note that the constraint applies also to bound nodes (remember however that bound nodes can not be isolated, since in this case they are garbage-collected).

This class of graphs has been chosen because of the results presented in the previous section. In particular, parallel closure is not meaningful for 2-shared graphs (since adding additional edges in parallel changes the number of tentacles attached to common nodes), thus Theorem 3.1 and Theorem 3.2 can not be applied. Also, for closed graphs Theorem 3.3 can not be applied, since there is no free node to bind.

Even if this class of graphs is quite simple, it is worthwhile to study it. In fact, it shows some interesting features of the two synchronization models and it is a first step towards the more general results in the next section.

Since we are interested in computations internal to this class of graphs, we have to check that transitions preserve the two required properties. Closedness is always preserved, as shown by the following proposition.

Proposition 3.4. *Let $\mathcal{P}$ be a set of productions. If $\mathcal{P} \Vdash_S \emptyset \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ with $S \in \{H, M\}$ then $\Phi = \emptyset$.*

Proof. Notice that both Λ and π are empty. Thus, from Definition 1.21, Γ_Λ is empty too, and $\Phi = \emptyset\pi \cup \Gamma_\Lambda$ is empty. □

The 2-shared property is not preserved in general, as can be easily seen (consider for instance the ring-to-star reconfiguration in Example 1.1). We will present a syntactic condition on productions, and we will show that all the computations using only productions satisfying this condition preserve the 2-shared property. However, the definition of the syntactic condition and the proof that it is enough to guarantee the preservation of the 2-shared property are quite complex and technical. We think that for most readers it is enough to know that such a condition exists, and that all the constructions that we will show use productions satisfying it. For a deeper understanding, but avoiding however some technical difficulties, just the case of productions with $\pi = \mathrm{id}$ can be considered, since it is enough in most the cases (but the case of $\pi \neq \mathrm{id}$ must be considered in the proof).

Let us consider the application of a production to one edge: when the rewritten edge is removed, all the nodes attached to it lose one attached tentacle (two if the edge was attached two times to the same node). Thus when inserting the graph in the RHS of the production, the same number of tentacles attached to those nodes must be provided, and two tentacles must be attached to each new node. However, new attached tentacles are provided also when nodes are merged, since merging two nodes with one attached tentacle each produces a node with two attached tentacles. Essentially, attached tentacles are resources that must be preserved, but to correctly count them, not only actual attached tentacles must be considered, but also occurrences of nodes as parameters in Λ (which cause merges when synchronization is performed) or in π must be accounted for, since they guarantee that further tentacles will be provided by the environment.

We give now a formal way to count how many times a node is actually used in a transition, dealing with both the left part of the transition, and the more complex right part.

Definition 3.5 (Left counting). *Given a transition $T = \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ the* left occurrences *of a node x in T are the occurrences of x in G, i.e. the tentacles in G attached to x. The* left counting *of x in T is the number of the left occurrences of x in T.*

Definition 3.6 (Right counting). *Given a transition $T = \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ the* right occurrences *of a node x in T are the occurrences of x in G', i.e., the tentacles in G' attached to x, plus the occurrences of x as parameter in Λ, plus, for each node y such that $\{x/y\}$ is in π, the occurrence of y in the LHS if it is exactly one. The* right counting *of x in T is the number of the right occurrences of x in T.*

A transition is connection-preserving iff for each node in the destination graph its left counting equals (almost always) its right counting.

Definition 3.7 (Connection-preserving transitions).
A transition $T = \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ is connection-preserving *iff for each node x occurring in T:*

- *if x occurs in both $\Gamma \vdash G$ and in $\Phi \vdash G'$ with left counting 0 or 1, then its right counting equals its left counting;*
- *if x occurs in both $\Gamma \vdash G$ and in $\Phi \vdash G'$ with left counting 2, then its right counting is either 0 or 2;*
- *if x occurs in $\Phi \vdash G'$ but not in $\Gamma \vdash G$, then its right counting is either 0 or 2.*

Proposition 3.5. *Let $\mathcal{P}$ be a set of connection-preserving productions and $\emptyset \vdash G$ a closed 2-shared graph. If $\mathcal{P} \Vdash_S \emptyset \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ with $S \in \{H, M\}$ then $\Phi \vdash G'$ is a closed 2-shared graph.*

Proof. Clearly, $\Phi \vdash G'$ is closed thanks to Proposition 3.4. Also, Λ and π are empty. We will prove that in each step of the derivation of the transition, the intermediate transition is connection-preserving.

Since the derived transition has empty Λ and π, then the only summand of the right counting for a node x that can be non zero is the number of occurrences of x in G'. Since the LHS is a 2-shared graph and the transition is connection-preserving, this number of occurrences must be either 0 or 2. Thus the thesis will follow.

Notice that no derivation for a transition with left counting at most 2 can have a premise with left counting more than 2, thus we have to deal only with transitions with left counting at most 2. Also, no rule can change the left counting of a bound node, thus left countings for bound nodes must be 2 (bound nodes with left counting 0 are garbage-collected).

We will prove the above property by rule induction on the derivation (we use suffix S for rules, where S must be instantiated with either H or M).

Productions) By hypothesis.

Rule (par-S)) The theorem holds, since each node x occurs in just one of the premises, and thus its left and right countings in the conclusion are equal to the ones in the premise where it occurs.

Rule (merge-S)) We can consider the simpler case of the merge of just two nodes, i.e. a renaming $\sigma = \{x/y\}$ since any renaming σ' can be decomposed as a composition of such renamings, and the application of rule (merge-S) with renaming σ' produces the same transition as the one derived by applying rule (merge-S) for each renaming in the decomposition.

By definition, both x and y must occur in Γ. If one of them has left counting 0, and if it occurs in $\Phi \vdash G'$, then it also has right counting 0, and nothing happens when merging it with the other one. Since the sum of the two left countings is at most 2, the only other possible case is that both x and y have left counting 1.

We will consider the application of σ first, and then the effects of the synchronization. Notice that σ is applied to both the left and the right graphs, and to the synchronization Λ. Also, it transforms the renaming π into a new renaming $\pi'' = \mathrm{mgu}(\{z\sigma = w\sigma | z\pi = w\pi\})$. Apart from the effect on π, it makes y to disappear from the transition, while the countings for x become each the sum of the corresponding previous countings for x and y. The same happens also if we consider the effect on π, since it merges the equivalence classes of x and y according to the relation $z \mathcal{R} w$ iff $z\pi = w\pi$, but thus the parts dealing with π of the two right countings are just added (remember that y disappears).

Now we have to consider the effect of the synchronization. For each pair of matched parameters z and w with $z \neq w$, their right countings are decreased by 1, since they disappear from Λ. Clearly, their right countings in the premise are at least 1. We will consider different cases according to these right countings.

Both 1: z and w occurred in the premise just in Λ and thus in Φ, and also their left countings were 1. The only possibility is that they are both in Γ, since neither

bound nodes nor newly created nodes can have left counting 1. Thus the renaming $\{z/w\}$ (or $\{w/z\}$) is added to π, and this gives exactly a contribution of 1 to the right counting for z, while w disappears from $\Phi \vdash G'$.

1 and 2: the first node (suppose that it is z) must be in Γ, and we can suppose that it is chosen as representative, while w disappears. The other old occurrence of w gives the required contribution to the right counting of z in the result. Notice that, also if both the nodes are in Γ and the renaming $\{z/w\}$ is added to π, since w has not left counting 1, it does not contribute to the right counting of z.

Both 2: as in the previous case if at least one of the two nodes is in Γ. Otherwise no renaming is added to π, two occurrences are deleted from Λ, but the other two become occurrences of the node chosen as representative, giving 2 as right counting as required.

If $z = w$ instead, the two occurrences simply disappear, thus we have in the conclusion a left counting 2 and a right counting 0 (since the previous right counting was 2). The case of many matchings among the same nodes (such as $(z, w), (z, w)$) reduces to the one above since we consider matchings one by one.

Rule (res-S)) This rule clearly does not influence the countings of nodes different from the bound node x. Also for x however the left counting is not changed, and for the right counting neither the part dealing with the graph nor the one dealing with Λ are changed. The part dealing with π also does not change, since x is not a representative if it is part of a non trivial equivalence class. This proves the thesis.

Rule (new-S)) Trivial. □

We present now some examples of connection-preserving productions, and of transitions derived from them.

Example 3.2. *Let us consider the simple production:*

$$x, y \vdash C(x, y) \xrightarrow{(x,a,\langle y\rangle),\mathrm{id}} x, y \vdash C(x)$$

This production is connection-preserving since both x and y have countings 1, in particular the right counting of y is 1 since y occurs in Λ. Using Milner synchronization we can combine this production with:

$$x, z \vdash C(x, z) \xrightarrow{(x,\overline{a},\langle z\rangle),\mathrm{id}} x, z \vdash C(x)$$

to derive:

$$x, y, z \vdash C(x, y)|C(x, z) \xrightarrow{(x,\tau,\langle\rangle),\{z/y\}} x, z \vdash C(x)|C(x)$$

Here both the countings for x are equal to 2. Also, z has left counting 1, and right counting 1 too since $\{z/y\} \subseteq \pi$ and y has left counting 1. Finally, y does not occur in $x, z \vdash C(x)|C(x)$, thus its right counting is not important.

We can compose the above transition with an idle production for graph $x, y \vdash C(y, z)$ *(notice that idle productions are always connection-preserving) obtaining:*

$$x, y, z \vdash C(x, y)|C(x, z)|C(y, z) \xrightarrow{(x,\tau,\langle\rangle),\{z/y\}} x, z \vdash C(x)|C(x)|C(z, z)$$

Now all the countings are equal to 2*. In particular, notice that the left counting of* y *does not contribute to the right counting of* z*. Also, both the graphs are 2-shared graphs, and adding restrictions for all the nodes produces exactly a transition between closed 2-shared graphs.*

We consider now a slightly different example to show a particular case: let us consider the composition of the first production with the following production.

$$x, y \vdash C(x, y) \xrightarrow{(x,\overline{a},\langle y\rangle),\mathrm{id}} x, y \vdash C(x)$$

The resulting transition is:

$$x, y \vdash C(x, y)|C(x, y) \xrightarrow{(x,\tau,\langle\rangle),\mathrm{id}} x, y \vdash C(x)|C(x)$$

Here x *has both countings* 2*, while* y *has left counting* 2 *and right counting* 0*, but this is again a connection-preserving transition.*

We will show in Section 3.4 (Theorem 3.7) that connection-preserving productions are expressive enough to simulate general Hoare transitions via a translation of graphs.

From now to the end of Section 3.4 we will concentrate on transitions between closed 2-shared graphs. In particular we will consider suitable restrictions of the sets of computations 1, *all* and *max*.

Notation. *We denote with* $2s{-}1$*,* $2s{-}all$ *and* $2s{-}max$ *the subsets of the sets of computations* 1*, all and max respectively containing only transitions between closed 2-shared graphs.*

Next theorem shows that for these sets of computations Milner synchronization can be used to simulate Hoare synchronization.

Theorem 3.4. $(M, C) \succeq (H, C)$ *for each* $C \in \{2s{-}1, 2s{-}all, 2s{-}max\}$.

Proof. We have just to consider the synchronizations that are performed on nodes with two attached tentacles, since nodes with zero attached tentacles are garbage-collected since they are bound. Hoare synchronization allows a transition only if the same action is performed by both the edges, and it merges the corresponding parameters.

With Milner synchronization the two actions must be complementary. The effect performed on the parameters is equal to the one performed by Hoare synchronization. For each Hoare action a (different from ϵ) we introduce a pair of complementary actions a and $\overline{a}$, with the same arity of the Hoare action.

The translation function f maps each set of productions to the set including, for each Hoare production, productions with the same structure and with all the possible combinations of actions and coactions for each Hoare action (ϵ actions are preserved). Thus each production with n non ϵ actions gives rise to 2^n productions.

All the transitions allowed by the Hoare model are allowed also by the Milner one, in fact for each pair of synchronizing actions we can always complement one of them, and form a pair of complementary actions. No more transitions are possible, since in the Milner model all the edges must use actions that correspond to the same Hoare action. Notice that action ϵ has the same behaviour in both the models. □

We present a simple example to clarify the theorem above.

Example 3.3. *Let us consider the set of productions generated by:*

$$x, y \vdash R(x, y) \xrightarrow{(x,a,\langle\rangle),(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y)$$

This production in HSHR has a similar effect to the first production of Example 1.1, that is it allows to grow a ring graph, but here all the edges replicate themselves at each step, thus the size of the ring doubles at each step. We can have for instance the transition (between closed 2-shared graphs):

$$\emptyset \vdash \nu x, y\ R(x, y)|R(x, y) \ \rightarrow\ \emptyset \vdash \nu x, y, z, w\ R(x, z)|R(z, y)|R(x, w)|R(w, y)$$

where on both x and y we have a synchronization between two actions a.

In order to have the same effect using MSHR, the following four productions can be used:

$$\begin{aligned}
x, y \vdash R(x, y) &\xrightarrow{(x,a,\langle\rangle),(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y)\\
x, y \vdash R(x, y) &\xrightarrow{(x,\overline{a},\langle\rangle),(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y)\\
x, y \vdash R(x, y) &\xrightarrow{(x,a,\langle\rangle),(y,\overline{a},\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y)\\
x, y \vdash R(x, y) &\xrightarrow{(x,\overline{a},\langle\rangle),(y,\overline{a},\langle\rangle),\mathrm{id}} x, y \vdash \nu z\ R(x, z)|R(z, y)
\end{aligned}$$

For instance we can choose for the first edge two actions a, and this requires the second edge to perform two actions $\overline{a}$. This can be done by using the first and the fourth production respectively.

Here two of the productions above are enough, however in more complex cases all of them may be necessary.

This proves that, for closed 2-shared graphs, Hoare synchronization is equal to Milner synchronization where the distinction between actions and coactions is dropped. In that case, Milner synchronization is strictly more expressive than Hoare synchronization and the additional expressiveness is given exactly by the asymmetry, as shown by the following theorem for $2s-1$-expressiveness and $2s-all$-expressiveness of Hoare synchronization.

Theorem 3.5.
$(H, C_1) \not\succeq (M, C_2)$ *for each* $C_1 \in \{2s{-}1, 2s{-}all\}$ *and* $C_2 \in \{2s{-}1, 2s{-}all, 2s{-}max\}$.

Proof. Let us consider the graph $\emptyset \vdash \nu x\ D(x)|D(x)$. This is a closed 2-shared graph and it is symmetric. Using Hoare synchronization, for any choice of production for the left edge, the same production can always be applied in the same transition also to the right edge, and the result is again a symmetric graph. In general, given a symmetric graph, for each choice of productions for the first half of the graph, if a transition exists, then also a transition that preserves the symmetry exists.

Let us consider the set of productions $\mathcal{P}$ generated by:

$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle),\mathrm{id}} x \vdash D'(x)$$
$$x \vdash D(x) \xrightarrow{(x,\overline{a},\langle\rangle),\mathrm{id}} x \vdash D(x)$$

For each $C_2 \in \{2s{-}1, 2s{-}all\}$ we have:
C_2- behav$^M(\mathcal{P})(\emptyset \vdash \nu x\ D(x)|D(x)) = \{\emptyset \vdash \nu x\ D(x)|D(x), \emptyset \vdash \nu x\ D'(x)|D(x)\}$
For $2s{-}max$-expressiveness instead we have:
$2s{-}max$- behav$^M(\mathcal{P})(\emptyset \vdash \nu x\ D(x)|D(x)) = \{\emptyset \vdash \nu x\ D'(x)|D(x)\}$
Since these behaviours contain no symmetric final graph for the choice of applying a non idle production to the first edge, then these behaviours can not be obtained using HSHR and any set of computations that, if it contains a computation, contains also all the computations of the same length. Notice that both the sets of computations $2s{-}1$ and $2s{-}all$ satisfy this assumption since the productions are connection-preserving (thus all the generated computations satisfy the 2-shared condition). □

Whether the above result holds also for $2s{-}max$-expressiveness is still an open problem. The above approach does not work since the set of computations $2s{-}max$ does not enjoy the required closure property (there can be two computations of the same length, one reaching a stable state and one reaching a non stable state).

We show in particular that the above behaviour can be specified using HSHR and $2s{-}max$ computations.

Example 3.4. *We show here that the particular behaviour shown in the proof of Theorem 3.5 can also be specified using* $2s{-}max$ *computations and HSHR. Let us consider the set of productions* $\mathcal{P}$ *generated by:*

$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle),\mathrm{id}} x \vdash D'(x)$$
$$x \vdash D(x) \xrightarrow{(x,a,\langle\rangle),\mathrm{id}} x \vdash D(x)$$
$$x \vdash D'(x) \xrightarrow{(x,b,\langle\rangle),\mathrm{id}} x \vdash D(x)$$

Starting from the graph $\emptyset \vdash \nu x\ D(x)|D(x)$ *we can have either transitions to* $\emptyset \vdash \nu x\ D(x)|D(x)$ *itself, or to either* $\emptyset \vdash \nu x\ D'(x)|D(x)$ *or* $\emptyset \vdash \nu x\ D'(x)|D'(x)$. *However the only graph that has no outgoing transitions is* $\emptyset \vdash \nu x\ D'(x)|D(x)$. *Thus*

we have:
$2s{-}max\text{-}\mathrm{behav}^H(\mathcal{P})(\emptyset \vdash \nu x\ D(x)|D(x)) = \{\emptyset \vdash \nu x\ D'(x)|D(x)\}$

The trick is that each computation leading to a symmetric graph can be extended to obtain an infinite computation, thus only asymmetric graphs appear in the $2s{-}max$ *behaviour.*

It is not clear whether such a technique can be extended to deal also with multiple concurrent synchronizations and mobility, or if more complex behaviours can be obtained using Milner synchronization but not using Hoare synchronization.

To bridge the gap between HSHR and MSHR we consider now a different form of simulation, that uses a translation for graphs. Intuitively, we want to simulate a computation by translating the starting graph, performing a corresponding computation derived using another synchronization model (or belonging to another set of computations), and then translating back the result. First of all we need a syntax to denote the forward and backward translations.

Definition 3.8 (Forward and backward translations). *The forward and backward translations are two functions* $[\![-]\!], [\![-]\!]^{-1} : Graphs \to Graphs$ *such that for each allowed graph* $\Gamma \vdash G$ *we have* $[\![[\![\Gamma \vdash G]\!]]\!]^{-1} = \Gamma \vdash G$.

Note that we may have $[\![[\![\Gamma \vdash G]\!]^{-1}]\!] \neq \Gamma \vdash G$. Also, we will consider only graphs to be translated where some particular edge labels do not occur, since they are introduced by the translation and they have a particular behaviour. This is not a strong restriction, since edge labels can be coherently changed without altering the behaviour.

Notation. *We denote with* $[\![-]\!]$ *and* $[\![-]\!]^{-1}$ *also the trivial extensions of* $[\![-]\!]$ *and* $[\![-]\!]^{-1}$ *to sets of graphs.*

The new form of simulation allows to define a new expressiveness preorder, which we call T-expressiveness preorder (where T stands for translation) and which we denote as $\succsim$.

Definition 3.9 (T-expressiveness preorder $\succsim$). *Let* S_1 *and* S_2 *be synchronization models and let* C_1 *and* C_2 *be sets of computations. We say that* (C_1, S_1) *can simulate* (C_2, S_2)*, written* $(C_1, S_1) \succsim (C_2, S_2)$ *iff there exists a translation function* f *from sets of productions to sets of productions and forward and backward translations* $[\![-]\!]$ *and* $[\![-]\!]^{-1}$ *such that for each set of productions* $\mathcal{P}$ *and for each allowed graph* $\Gamma \vdash G$ *the following equation holds:*

$$C_2\text{-}\mathrm{behav}^{S_2}(\mathcal{P})(\Gamma \vdash G) = [\![C_1\text{-}\mathrm{behav}^{S_1}(f(\mathcal{P}))([\![\Gamma \vdash G]\!])]\!]^{-1}$$

The main difference w.r.t. Definition 3.2 is that here the forward and backward translations are used.

The following proposition proves that the T-expressiveness preorder $\succsim$ is indeed a preorder.

Lemma 3.1. *$\succsim$ is a preorder.*

Proof. Reflexivity is trivial using identity functions both for the translation function f and for the forward and backward translations $[\![-]\!]$ and $[\![-]\!]^{-1}$. Transitivity is trivial too, by composing the translation functions, the forward translations and the backward translations. □

The expressiveness preorder $\succeq$ is included in the T-expressiveness preorder $\succsim$, as proved by the following proposition.

Proposition 3.6. $\succeq \subseteq \succsim$.

Proof. It is enough to choose identity functions as forward and backward translations. □

In general different kinds of forward and backward translations can be used. However, for our aims, translations based on the concept of *amoeboid* are enough. Amoeboids are particular graphs whose behaviour simulates the behaviour of nodes. In particular, an amoeboid with an interface Int (which is a set of nodes) such that $|Int| = n$ is used to simulate a node where n tentacles are attached. We present here the general definitions of amoeboids and the corresponding forward and backward translations, that will be instantiated in this and in the next section to model different kinds of synchronizations.

Notation. *When a particular kind of amoeboids is used, we will label the forward and backward translations with the kind of the amoeboids, e.g. $[\![-]\!]_H$ is the forward translation that uses Hoare amoeboids.*

Definition 3.10 (Amoeboids). *We suppose that the ranked set of edge labels LE is partitioned into two subsets, the subset LE_N of labels for normal edges, and the subset LE_{Am} of labels for amoeboid edges. Let Int be a set of nodes. An* amoeboid *$Am(Int)$ with interface Int is any connected graph whose edges are amoeboid edges, and whose set of nodes can be partitioned into the interface Int and the auxiliary nodes in Aux. Nodes in Int are attached to just one tentacle of one amoeboid edge and free. Nodes in Aux are attached to two tentacles of amoeboid edges and bound.*

Note that $\mathrm{fn}(Am(Int)) = Int$.

We need slightly different features for amoeboids modeling free nodes (which are used just inside productions, since we will always deal only with closed graphs), and amoeboids modeling bound nodes. We call them free amoeboids and bound amoeboids respectively.

Definition 3.11 (Free and bound amoeboids). *Each kind K of amoeboids has two subkinds,* free amoeboids *Kf and* bound amoeboids *Kb. We require that by joining* 3 *bound amoeboids of kind Kb by merging pairs of nodes of their respective interfaces we get a bound amoeboid of kind Kb. We require that by joining a free*

amoeboid of kind Kf and a bound amoeboid of kind Kb by merging two nodes of their respective interfaces we get a bound amoeboid of kind Kb. Free amoeboids with a singleton as interface are always composed only by the node in the interface itself.

Intuitively, the forward translation just takes each node x and it replaces x with an amoeboid with a suitable interface. To do that, a node for each tentacle attached to x must be added. A node for x is preserved in free amoeboids. Then the inverse translation just removes the amoeboids in the graph and it inserts the nodes they stand for at their place.

In order to formally define the forward translation, we have to introduce first a standard decomposition for graphs. A weaker form of decomposition would be enough here (dealing with graphs instead of with graph terms), but we introduce this form since it will be used also later (see Section 8.1).

Definition 3.12 (Standard decomposition for graphs). *Given a graph $\Gamma \vdash G$, its* standard decomposition *has the form:*

$$\Gamma \vdash \nu X\ \hat{G}\sigma_G$$

where $\hat{G}$ is a parallel composition of edges (or nil if there are no edges), left associated and in a canonical order and where names not in Γ are canonical ones, and they appear in a fixed order in $\hat{G}$, each name in $\hat{G}$ occurs at most once, $\Gamma \cap X = \emptyset$, $X \subseteq \mathrm{fn}(\hat{G}\sigma_G)$, restrictions are in a fixed order, σ_G is an idempotent renaming such that all the bindings in σ_G are of the form $\{x/y\}$ with y occurring in $\hat{G}$ after x, $x \in \Gamma \cup X$, and $\nu X\ \hat{G}\sigma_G \equiv G$.

Essentially the above decomposition chooses a particular graph term for the graph, and it ensures that each name occurs in $\hat{G}$ at most once (as done for Fusion Calculus agents by Definition 1.18). Notice that we suppose to have ordered sequences of canonical names and of canonical labels.

Notation. *In the examples we may use suggestive names instead of always the same sequence of canonical names to improve the readability.*

The following lemma ensures that each graph can be uniquely decomposed, and that two graphs can have the same standard decomposition iff they are isomorphic.

Lemma 3.2. *Let $\Gamma \vdash G$ and $\Gamma' \vdash G'$ be two graphs. There exists a unique standard decomposition $\Gamma \vdash \nu X\ \hat{G}\sigma_G$ of $\Gamma \vdash G$. Also, the standard decompositions of $\Gamma \vdash G$ and $\Gamma' \vdash G'$ are equal iff $\Gamma \vdash G$ and $\Gamma' \vdash G'$ are isomorphic.*

Proof. First, we have to show that the decomposition exists. In graph $\Gamma \vdash G$ all the restrictions can be moved to the outside, obtaining $\Gamma \vdash \nu X_1\ G_1$ where G_1 is a parallel composition of edges. Then we can make the names in X_1 canonical and disjoint w.r.t. names in Γ using α-conversion. Also, restrictions for names not occurring in G_1 can be deleted. After that the set of restrictions is the desired one. We can now

fix the order of restrictions, the order of edges, and enforce left associativity. Finally we can decompose G_1 as $\hat{G}\sigma_G$ as follows. For each occurrence of a name x in G_1 but the first one, the occurrence is replaced by the next unused canonical name y, and the renaming $\{x/y\}$ is added to σ_G.

As required each decomposition is structurally congruent to the original graph.

No two different standard decompositions can be structurally congruent since axioms can not change labels of edges, free names or connections among the edges, and all the other elements are fixed by the decomposition. Thus two decompositions are equal iff the starting graphs are structurally congruent and so isomorphic. □

We can now formalize the forward translation for closed graphs. Closed graphs are enough for our purposes.

Definition 3.13 (Forward translation). *For each set of nodes Int let $Am_K(Int)$ be an amoeboid of kind K with interface Int. Let $\emptyset \vdash G$ be a closed graph and $\emptyset \vdash \nu X\ \hat{G}\sigma_G$ be its standard decomposition. The* forward translation *of $\emptyset \vdash G$ using amoeboids of kind K is:*

$$[\![\emptyset \vdash G]\!]_K = \emptyset \vdash \nu Y\ \hat{G} \mid \prod_{x \in X} Am_{Kb}(\{y | y \in \mathrm{fn}(\hat{G}) \wedge y\sigma_G = x\})$$

where Y contains all the free names in the graph term. We also allow the forward translation to add bound amoeboids with empty interface.

Last possibility is necessary to have a correspondence between a computation and its translation, since these amoeboids are created when two extremities of an amoeboid are merged.

We need an auxiliary definition in order to define the inverse translation too.

Definition 3.14 (Standard-amoeboid decomposition). *Given a graph $\Gamma \vdash G$, its* standard-amoeboid decomposition *has the form:*

$$\Gamma \vdash \nu Y\ \ \mathrm{std}(G) \mid \mathrm{amoeb}(G)$$

where $\mathrm{std}(G)$ *is a parallel composition of standard edges and* $\mathrm{amoeb}(G)$ *is a parallel composition of amoeboid edges and $G \equiv \nu Y\ \ \mathrm{std}(G) \mid \mathrm{amoeb}(G)$.*

We can now define the inverse translation for closed graphs. Closed graphs are enough for our purposes.

Definition 3.15 (Inverse translation). *Given a closed graph $\emptyset \vdash G$, let $\cong_G$ be the equivalence relation on nodes in (a term representing) G defined by: $x \cong_G y$ iff x and y are connected in* $\mathrm{amoeb}(G)$. *Let σ be a substitutive effect of $\cong_G$. The* inverse translation *of $\emptyset \vdash G$ is:*

$$[\![\emptyset \vdash G]\!]^{-1} = \emptyset \vdash \nu X\ \ \mathrm{std}(G)\sigma$$

where X contains all the nodes in $\mathrm{std}(G)\sigma$.

Next lemma proves that the translations based on amoeboids are correct forward and backward translations for closed graphs, i.e., they satisfy the condition from Definition 3.8. Note that this result does not depend on which kind of amoeboid is used.

Lemma 3.3. *For each closed graph $\emptyset \vdash G$ containing only standard edges, $[\![[\![\emptyset \vdash G]\!]]\!]^{-1} = \emptyset \vdash G$.*

Proof. The edge part of G is clearly preserved since the forward translation adds amoeboid edges that are removed by the backward translation, while standard edges are always preserved. As far as nodes are concerned note that two nodes are connected by an amoeboid in $[\![\emptyset \vdash G]\!]$ (and thus merged by σ in the inverse translation) iff they correspond to instances of the same node in the starting graph. Since all the nodes are α-convertible their names do not matter, and the above proved preservation of connections guarantees that the node part is preserved. □

We present here a simple example of the translations, at the abstract level defined so far. More concrete examples will be presented later, when the framework will be instantiated on particular kinds of amoeboids.

Example 3.5. *Let us consider the graph $\emptyset \vdash \nu x, y, z\ C(x, y)|C(y, z)$. Its standard decomposition is $\emptyset \vdash \nu x, y, z, n_1\ C(x, y)|C(n_1, z)\{y/n_1\}$. Thus one of its forward translations (the one without added bound amoeboids with empty interface) is:*

$$[\![\emptyset \vdash \nu x, y, z\ C(x, y)|C(y, z)]\!] = \\ = \emptyset \vdash \nu x, y, z, n_1\ C(x, y)|C(n_1, z)|Am(x)|Am(y, n_1)|Am(z)$$

In order to compute the backward translation we have to consider the standard-amoeboid decomposition, which has $Y = \{x, y, z, n_1\}$, $\mathrm{std}(G) = C(x, y)|C(n_1, z)$ and $\mathrm{amoeb}(G) = Am(x)|Am(y, n_1)|Am(z)$. Thus the substitutive effect σ is e.g. $\{n_1/y\}$, and the result of the translation is:

$$[\![\emptyset \vdash \nu x, y, z, n_1\ C(x, y)|C(n_1, z)|Am(x)|Am(y, n_1)|Am(z)]\!]^{-1} = \\ = \emptyset \vdash \nu x, z, n_1\ C(x, n_1)|C(n_1, z)$$

which is equal to the starting graph up to α-conversion.

We have to extend the forward translation also to productions, since the productions that we want to use in the simulating framework are the translations of the productions used in the simulated framework, plus the productions for amoeboids. We will present the translation for productions at a less formal level w.r.t. previous definitions, since its formalization is quite complex, while the ideas behind it are simpler and technicalities are similar to the ones emerged while translating graphs.

Definition 3.16 (Translation of productions).
Let $P = x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G$ *be a production. The forward translation of* P *using amoeboids of kind* K *is:*

$$[\![P]\!]_K = x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda',\mathrm{id}} \Phi' \vdash \nu X\ G'|G_{Am}|G_\pi$$

where Λ' *and* G' *are obtained by replacing each right occurrence of each name* x *but the first one with a fresh name. Let* σ *be the renaming mapping each fresh name to the corresponding name. Then* G_{Am} *contains amoeboids of kind* K *connecting exactly the names merged by* σ *(free amoeboids for nodes in* Φ'*, and bound amoeboids for the other ones). Also,* G_π *contains bound amoeboids connecting the nodes merged by* π*, but for representatives the name of the corresponding occurrence is used instead of the name of the node. Finally,* X *binds nodes not in* Φ'*.*

The following example illustrates the definition.

Example 3.6. *We show now some examples of the translation of productions, with still unspecified amoeboids. We start with a simple connection-preserving production.*

$$\begin{aligned}&[\![x, y \vdash C(x, y) \xrightarrow{(x,a,\langle y\rangle),\mathrm{id}} x, y \vdash \nu z\ C(x, z)|D(z)]\!]_K = \\ &\quad = x, y \vdash C(x, y) \xrightarrow{(x,a,\langle y\rangle),\mathrm{id}} x, y \vdash \nu z, z_1\ C(x, z)|D(z_1)|Am_{Kb}(z, z_1)\end{aligned}$$

Amoeboids for x *and* y *are not shown since they are just the nodes* x *and* y *respectively. We show now a non connection-preserving production, requiring explicit free amoeboids:*

$$\begin{aligned}&[\![x, y \vdash C(x, y) \xrightarrow{(x,a,\langle y\rangle),\mathrm{id}} x, y \vdash C(x, y)]\!]_K = \\ &\quad = x, y \vdash C(x, y) \xrightarrow{(x,a,\langle y_1\rangle),\mathrm{id}} x, y, y_1 \vdash \nu y_2\ C(x, y_2)|Am_{Kf}(y, y_1, y_2)\end{aligned}$$

We conclude with a production where $\pi \neq \mathrm{id}$*:*

$$\begin{aligned}&[\![x, y, z \vdash F(x, y, z) \xrightarrow{(x,a,\langle y\rangle),\{y/z\}} x, y \vdash C(x, y)|D(y)]\!]_K = \\ &\qquad = x, y, z \vdash F(x, y, z) \xrightarrow{(x,a,\langle y_1\rangle),\mathrm{id}} \\ &\qquad \to x, y, z, y_1 \vdash \nu y_2, y_3, y_4\ C(x, y_2)|D(y_3)|Am_{Kf}(y, y_1, y_2, y_3, y_4)|Am_{Kb}(y_4, z)\end{aligned}$$

Note that we have a bound amoeboid for π*, connecting the occurrence* y_4 *of* y *with* z*.*

Now that we have defined the general framework, we can present the first application of amoeboids. We define $2M$ amoeboids, which are amoeboids useful to implement Milner synchronization on top of Hoare synchronization in the case of closed 2-shared graphs. Since we deal with 2-shared graphs only, it is enough to have bound amoeboids with either 0 or 2 nodes in their interface. Free amoeboids are required only with singletons as interfaces, thus they are simple nodes.

Definition 3.17 ($2Mb$ amoeboids). *A* $2Mb$ amoeboid *with two nodes in its interface is a chain composed by an odd number of edges of rank* 2 *labeled with* L *(for link). A* $2Mb$ *amoeboid with empty interface is either empty or it is a ring composed by an even number of* L *edges.* L *edges have, as only non idle productions, one production of the following form for each action* a *of arity* k*:*

$$x, y \vdash L(x, y) \xrightarrow{(x,a,\vec{x}),(y,\overline{a},\vec{y}),\mathrm{id}} \\ \rightarrow x, y, \mathrm{n}(\vec{x}), \mathrm{n}(\vec{y}) \vdash L(x, y) | \prod_{i \in \{1,\ldots,k\}} L(\vec{x}[i], \vec{y}[i])$$

where $\vec{x}$ *and* $\vec{y}$ *are vectors of new names such that* $|\vec{x}| = |\vec{y}| = k$*.*

The idea behind this definition is that an action a performed on one element of the interface of a $2Mb$ amoeboid is matched by an action $\overline{a}$ performed on the other node attached to the first edge in the chain (note that here coactions are normal Hoare actions). Then the other edges are in pairs, thus they change the action from $\overline{a}$ to a and back to $\overline{a}$, without influencing the allowed synchronizations. Nodes to be merged are connected via L edges instead, and this explains why chains of arbitrary length are required: the length increases when nodes should be merged. Note also that $2Mb$ amoeboids satisfy the condition for bound amoeboids in Definition 3.11.

Next lemma shows that complex $2Mb$ amoeboids behave exactly as simple L edges.

Lemma 3.4.
Let $Am_{2Mb}(\{x, y\})$ *be a* $2Mb$ *amoeboid with interface* $\{x, y\}$*.* $Am_{2Mb}(\{x, y\})$ *has, as only non idle transitions, one transition of the following form for each action* a *of arity* k*:*

$$x, y \vdash Am_{2Mb}(\{x, y\}) \xrightarrow{(x,a,\vec{x}),(y,\overline{a},\vec{y}),\mathrm{id}} \\ \rightarrow x, y, \mathrm{n}(\vec{x}), \mathrm{n}(\vec{y}) \vdash Am_{2Mb}(\{x, y\}) | \prod_{i \in \{1,\ldots,k\}} Am_{2Mb}(\{\vec{x}[i], \vec{y}[i]\})$$

where $\vec{x}$ *and* $\vec{y}$ *are vectors of new names such that* $|\vec{x}| = |\vec{y}| = k$*.*

Proof. The proof is by induction on the number n of L edges in the amoeboid, by keeping in mind that n is odd, thus $n = 1$ is the base case and the inductive step must prove that the property holds for $n + 2$ if it holds for n. □

We can now instantiate the general framework using $2M$ amoeboids.

Example 3.7. *Let us consider as closed 2-shared graph a (closed) ring of 3 elements:*

$$\emptyset \vdash \nu x, y, z \ R(x, y) | R(y, z) | R(z, x)$$

Its standard decomposition is:

$$\emptyset \vdash \nu x, y, z \ (R(x, y) | R(n_1, z) | R(n_2, n_3))\{y/n_1, z/n_2, x/n_3\}$$

and thus one of its forward translations is:

$$\emptyset \vdash \nu x, y, z, n_1, n_2, n_3, z_1, z_2$$
$$R(x,y)|R(n_1,z)|R(n_2,n_3)|L(x,n_3)|L(y,n_1)|L(z,z_1)|L(z_1,z_2)|L(z_2,n_3)$$

where we have used the most simple form for the amoeboids (which are all bound) for nodes x and y and a more complex form, that is a chain of three edges instead of just one edge, for the amoeboid for node z.

We will consider now a pair of simple productions that allow to enlarge the ring when two connected edges agree by performing a pair of complementary actions. We present the productions together with their forward translations.

$$[\![x, y \vdash R(x,y) \xrightarrow{(x,\epsilon,\langle\rangle),(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu w\ R(x,w)|R(w,y)]\!]_{2M} =$$
$$= x, y \vdash R(x,y) \xrightarrow{(x,\epsilon,\langle\rangle),(y,a,\langle\rangle),\mathrm{id}} x, y \vdash \nu w, w_1\ R(x,w)|R(w_1,y)|L(w,w_1)$$
$$[\![x, y \vdash R(x,y) \xrightarrow{(x,\overline{a},\langle\rangle),(y,\epsilon,\langle\rangle),\mathrm{id}} x, y \vdash R(x,y)]\!]_{2M} =$$
$$= x, y \vdash R(x,y) \xrightarrow{(x,\overline{a},\langle\rangle),(y,\epsilon,\langle\rangle),\mathrm{id}} x, y \vdash R(x,y)$$

Using MSHR and the two original productions we can derive the transition:

$$\emptyset \vdash \nu x, y, z\ R(x,y)|R(y,z)|R(z,x) \rightarrow$$
$$\rightarrow \emptyset \vdash \nu x, y, z, w\ R(x,w)|R(w,y)|R(y,z)|R(z,x)$$

where the synchronization is performed on node y between action a from $R(x,y)$ and action $\overline{a}$ from $R(y,z)$.

Similarly, in HSHR the translation of the graph can be rewritten using the translation of the productions together with the auxiliary production for L edge on edge $L(y,n_1)$ and idle productions on other edges. The derived transition is:

$$\emptyset \vdash \nu x, y, z, n_1, n_2, n_3, z_1, z_2$$
$$R(x,y)|R(n_1,z)|R(n_2,n_3)|L(x,n_3)|L(y,n_1)|L(z,z_1)|L(z_1,z_2)|L(z_2,n_3) \rightarrow$$
$$\emptyset \vdash \nu x, y, z, n_1, n_2, n_3, z_1, z_2, w, w_1$$
$$R(x,w)|R(w_1,y)|L(w,w_1)|R(n_1,z)|R(n_2,n_3)|$$
$$L(x,n_3)|L(y,n_1)|L(z,z_1)|L(z_1,z_2)|L(z_2,n_3)$$

Now the synchronization of the two actions a and $\overline{a}$ on y and n_1 respectively is mediated by the actions a and $\overline{a}$ performed on the same nodes by the production for $L(y,n_1)$.

Notice that by applying the backward translation to the final graph of the HSHR transition we get the final graph of the MSHR transition.

Next theorem shows that the behaviour of the above example can be generalized to any transition, thus ensuring that for $2s-1$ and $2s-all$ computations, HSHR can simulate MSHR.

Theorem 3.6. *$(H, C) \succsim (M, C)$ for each $C \in \{2s{-}1, 2s{-}all\}$.*

Proof. We have to prove that for each computation in (M, C), a corresponding computation can be done in (H, C), and that in (H, C) no more computations exist. It is enough to prove that there is a bijective correspondence between the transitions.

Let us consider a choice of productions for the edges of the starting graph. We will show that the corresponding Milner transition is built using the same productions on the corresponding edges of the translated graph, but using also productions for the amoeboids. For each node x in the starting graph (which is closed and 2-shared), there is a corresponding bound amoeboid in the translated graph. In Milner synchronization the two attached edges must perform either two ϵ actions, or two complementary actions. The same possibilities are offered by the $2Mb$ amoeboid for x, as proved by Lemma 3.4.

Nodes to be merged because of the synchronization are connected using bound amoeboids instead. We have to prove that the backward translation of the final graph is indeed the wanted one. The standard edges in the graph are the wanted ones by definition. The only thing that we have to show is that two nodes are connected iff they are instances of the same node in the final graph of the Milner transition, and that the connections are provided by bound amoeboids.

Two nodes should be connected either if they were connected in the source graph, or they were connected in the RHS of the production, or they are merged by a synchronization or by π. It is easy to see that in all the cases they are connected using amoeboids. Note also that the connection is always provided by a bound amoeboid. In fact preserved connections are bound amoeboids by hypothesis, connections created in the RHS for bound nodes are given by bound amoeboids and connections created for nodes free in the productions are given by free amoeboids, thus by connecting them with the existing bound amoeboids we get again bound amoeboids. Finally, amoeboids for π and the ones created by the synchronization are bound amoeboids, which connect pairs of bound amoeboids, thus the result is again a bound amoeboid. Note that, if two extremities of an amoeboid are connected by another amoeboid, then a bound amoeboid with empty interface is created. However, this is deleted by the backward translation.

Finally, notice that the result is a transition between closed 2-shared graphs since all the used productions are connection-preserving. Thus the built computation is indeed in the desired class. □

The same result does not hold for the set of computations $2s{-}max$, since bound amoeboids with empty interface can be created during the computation, and they are always able to replicate themselves. Thus, there can be no stable state after one of these has been created, and this does not allow to have the desired maximal computations. To be precise, these computations exist, and their image along the inverse translation is the wanted one, but they are not maximal.

We conclude here our treatment of closed 2-shared graphs, and in the next session we will get back to the case of general closed graphs.

3.4 MSHR vs HSHR with general closed graphs

In the previous section we have built two simulations, from MSHR to HSHR and viceversa, at the price of considering only closed 2-shared graphs and of using the forward and backward translations. Now we want to exploit the second technique to analyze the relationships between MSHR and HSHR when general closed graphs are rewritten. In particular, we drop the 2-shared condition. To do that however we rely heavily on the work done in the previous section, by reducing this more general case to the 2-shared case by using a new kind of amoeboids.

We introduce suitable sets of computations for our aim.

Notation. *We denote with $c{-}1$, $c{-}all$ and $c{-}max$ the subsets of the sets of computations* 1, *all and max respectively containing only transitions between closed graphs.*

We start by analyzing Hoare synchronization. By reducing Hoare synchronization for general closed graphs to Hoare synchronization for closed 2-shared graphs, we can then exploit Theorem 3.4 to implement HSHR using MSHR.

As said, we introduce amoeboids of a new kind, called H (for Hoare) amoeboids. Intuitively H amoeboids perform the broadcast of the chosen action to all the elements in the interface. Note that now we need amoeboids with interfaces of arbitrary size. However we build them using just two kinds of edges.

Definition 3.18 (H amoeboids). *An H* amoeboid *(either free or bound) with interface Int is an amoeboid with interface Int composed by edges labeled with H (for Hoare) of rank* 3 *and C (for closing) of rank* 1*. The above edges have, as only non idle productions, one production of the following form for each action a of arity k:*

$$x, y, z \vdash H(x,y,z) \xrightarrow{(x,a,\vec{x}),(y,a,\vec{y}),(z,a,\vec{z}),\mathrm{id}} \\ \to x, y, z, \mathrm{n}(\vec{x}), \mathrm{n}(\vec{y}), \mathrm{n}(\vec{z}) \vdash H(x,y,z) | \, \|_{i\in\{1,\ldots,k\}} H(\vec{x}[i], \vec{y}[i], \vec{z}[i])$$

$$x \vdash C(x) \xrightarrow{(x,a,\vec{x}),\mathrm{id}} x, \mathrm{n}(\vec{x}) \vdash C(x) | \, \|_{i\in\{1,\ldots,k\}} C(\vec{x}[i])$$

where $\vec{x}$, $\vec{y}$ and $\vec{z}$ are vectors of new names such that $|\vec{x}| = |\vec{y}| = |\vec{z}| = k$.

Such an amoeboid imposes the same action to be executed on each node and it creates a copy of itself for each set of corresponding names, which are thus connected in the resulting graph.

Next lemma characterizes the allowed transitions for H amoeboids.

Lemma 3.5. *Let $Am_H(Int)$ be an H amoeboid with interface Int. $Am_H(Int)$ has, as only non idle transitions, one transition of the following form for each action a of arity k:*

$$Int \vdash Am_H(Int) \xrightarrow{\{(x,a,\vec{x})|x\in Int\},\mathrm{id}} \\ \to Int, \bigcup_{x\in Int} \mathrm{n}(\vec{x}) \vdash Am_H(Int) | \, \|_{i\in\{1,\ldots,k\}} Am_H(\{\vec{x}[i] | x \in Int\})$$

where we have a vector $\vec{x}$ of new names with $|\vec{x}| = k$ for each $x \in Int$.

Proof. The proof is by induction on the number of edges in the amoeboid, by adding edges in such a way that at each step the considered graph is connected. □

Next theorem proves the desired simulation result.

Theorem 3.7.
$(H, C_{2s}) \gtrsim (H, C_c)$ for each $(C_{2s}, C_c) \in \{(2s{-}1, c{-}1), (2s{-}all, c{-}all)\}$.

Proof. The proof is similar to the one of Theorem 3.6, using Lemma 3.5 instead of Lemma 3.4. The only difference is that now both the nodes in the simulated framework and the amoeboids in the simulating framework require that all the connected edges perform the same action, and all the corresponding nodes (which may be more than 2) are connected using amoeboids. □

This result can be used together with Theorem 3.4 to get a translation from Hoare synchronization to Milner synchronization for any closed graph.

As for Theorem 3.6, using amoeboids it is not possible to preserve maximal computations, thus the result does not hold for them.

Let us see now an example of the simulation.

Example 3.8. *Let us consider the following production, taken from Example 1.1:*

$$x, y \vdash R(x, y) \xrightarrow{(x,r,\langle w\rangle)(y,r,\langle w\rangle),\mathrm{id}} x, y, w \vdash S(y, w)$$

The forward translation of the production using H amoeboids is:

$$x, y \vdash R(x, y) \xrightarrow{(x,r,\langle w\rangle)(y,r,\langle w_1\rangle)} x, y, w, w_1, w_2 \vdash C(x), S(y, w_2), H(w, w_1, w_2)$$

Notice that this is a connection-preserving production. Let us consider a closed ring of four elements:

$$\emptyset \vdash \nu x, y, z, v\ R(x, y)|R(y, z)|R(z, v)|R(v, x)$$

Its forward translation using H amoeboids is:

$$\begin{aligned}
&[\![\emptyset \vdash \nu x, y, z, v\ R(x, y)|R(y, z)|R(z, v)|R(v, x)]\!]_H = \\
&\quad = \emptyset \vdash \nu x, x_1, x_2, y, y_1, y_2, z, z_1, z_2, v, v_1, v_2 \\
&\qquad R(x, y)|R(y_1, z)|R(z_1, v)|R(v_1, x_1)| \\
&\qquad H(x, x_1, x_2)|C(x_2)|H(y, y_1, y_2)|C(y_2)|H(z, z_1, z_2)|C(z_2)|H(v, v_1, v_2)|C(v_2)
\end{aligned}$$

The transition in Figure 3.1 can be derived using the production above and the auxiliary productions for H amoeboids. In the figure, X is the set containing all the nodes in the resulting graph.

Finally, the resulting graph can be translated using the backward translation, and the result is:

$$\emptyset \vdash \nu x, y, z, v, w\ S(y, w)|S(z, w)|S(v, w)|S(x, w)$$

as desired.

$$
\begin{aligned}
\emptyset \vdash\ & \nu x, x_1, x_2, y, y_1, y_2, z, z_1, z_2, v, v_1, v_2 \\
& R(x,y)|R(y_1,z)|R(z_1,v)|R(v_1,x_1)| \\
& H(x,x_1,x_2)|C(x_2)|H(y,y_1,y_2)|C(y_2)|H(z,z_1,z_2)|C(z_2)|H(v,v_1,v_2)|C(v_2) \rightarrow \\
\emptyset \vdash\ & \nu X\ C(x)|S(y,w_3)|H(w_1,w_2,w_3)|C(y_1)|S(z,w_6)|H(w_4,w_5,w_6)| \\
& C(z_1)|S(v_1,w_9)|H(w_7,w_8,w_9)|C(v_1),S(x_1,w_{12}),H(w_{10},w_{11},w_{12})| \\
& H(x,x_1,x_2)|H(w_1,w_2,x')|C(x_2)|C(x')|H(y,y_1,y_2)|H(w_4,w_5,y')|C(y_2)|C(y')| \\
& H(z,z_1,z_2)|H(w_7,w_8,z')|C(z_2)|C(z')|H(v,v_1,v_2)|H(w_{10},w_{11},v')|C(v_2)|C(v')
\end{aligned}
$$

Figure 3.1: Translation of a HSHR transition into the 2-shared framework.

Now we want to apply the same approach to Milner synchronization, i.e. we want to find amoeboids that allow to simulate computations between general closed graphs based on Milner synchronization by using only 2-shared graphs. The amoeboids for Milner synchronization, called M (for Milner) amoeboids, are essentially routers that create a path from an action to the corresponding coaction. We have here, as for H amoeboids, the need for amoeboids with interface of arbitrary size.

Definition 3.19 (M amoeboids). *An M* amoeboid *(either free or bound) with interface Int is an amoeboid with interface Int composed by edges labeled with M (for Milner) of rank 3, L (for link) of rank 2 and I (for inactive) of rank 1. The edges labeled by M and L have respectively, as only non idle productions, one production of the following form for each action a of arity k and each permutation of the nodes in the interface:*

$$
\begin{aligned}
& x,y,z \vdash M(x,y,z) \xrightarrow{(x,a,\vec{x}),(y,\overline{a},\vec{y}),(z,\epsilon,\langle\rangle),\mathrm{id}} \\
& \qquad \rightarrow x,y,z,\mathrm{n}(\vec{x}),\mathrm{n}(\vec{y}) \vdash M(x,y,z)|\prod_{i\in\{1,\ldots,k\}} L(\vec{x}[i],\vec{y}[i]) \\
& x,y \vdash L(x,y) \xrightarrow{(x,a,\vec{x}),(y,\overline{a},\vec{y}),\mathrm{id}} \\
& \qquad \rightarrow x,y,\mathrm{n}(\vec{x}),\mathrm{n}(\vec{y}) \vdash L(x,y)|\prod_{i\in\{1,\ldots,k\}} L(\vec{x}[i],\vec{y}[i])
\end{aligned}
$$

where $\vec{x}$ and $\vec{y}$ are vectors of new names such that $|\vec{x}| = |\vec{y}| = k$.

For edges labeled by I only idle productions exist.

Intuitively, an action a on x is matched by a coaction $\overline{a}$ again on x done by the amoeboid. Thus on another node y an action a is performed by the amoeboid, and it is matched by a coaction $\overline{a}$, as required for Milner synchronization. Adding other edges does not change the behaviour.

The absence of productions for edges labeled by I guarantees that actions and coactions are originated only by standard edges. Notice however that a synchronization may happen between edges forming a cycle in the amoeboid. However such a

synchronization has no effect on the outside, and it just creates a bound amoeboid with empty interface, which is discarded by the backward translation.

Note that the behaviour of edges labeled with L is the same of the ones used in $2Mb$ amoeboids. Also, these edges are not strictly necessary here since $L(x,y)$ and $\nu z\ M(x,y,z)|I(z)$ have the same behaviour, but they simplify the presentation.

We are not able to prove a reduction theorem similar to Theorem 3.7, since there is here an additional problem: many independent synchronizations may be allowed inside an amoeboid during one transition, but this is not allowed in standard Milner synchronization. In particular, this happens when the pairs of interacting elements of the interface are connected by disjoint paths inside the amoeboid. Adding a blocking action is not enough to solve this problem for symmetry reasons.

Thus graphs translated using M amoeboids can simulate any allowed Milner synchronization, but they may allow also other transitions. This is shown by the example below.

Example 3.9. *Let us consider the following M amoeboid, which has $\{x,y,z,w\}$ as interface:*

$$x,y,z,w \vdash \nu v\ M(x,y,v)|M(v,z,w)$$

As required the amoeboid can provide pairs of complementary actions on any pair of nodes, such as in the transition:

$$x,y,z,w \vdash M(x,y,v)|M(v,z,w) \xrightarrow{(x,a,\langle\rangle),(w,\overline{a},\langle\rangle),\mathrm{id}} x,y,z,w \vdash M(x,y,v)|M(v,z,w)$$

thus allowing to synchronize an action at x with the corresponding coaction at w. However also the transition:

$$\begin{aligned} x,y,z,w \vdash M(x,y,v)|M(v,z,w) &\xrightarrow{(x,a,\langle\rangle),(y,\overline{a},\langle\rangle),(z,b,\langle\rangle),(w,\overline{b},\langle\rangle),\mathrm{id}} \\ &\to x,y,z,w \vdash M(x,y,v)|M(v,z,w) \end{aligned}$$

is allowed. This transition synchronizes two (action,coaction) pairs, the first one based on actions a and $\overline{a}$ performed at x and y respectively, and the other one based on actions b and $\overline{b}$ performed at z and w respectively.

The usual lemma describing the transitions allowed by the amoeboids in this case guarantees only the existence of the desired transitions, but others exist too.

Lemma 3.6. *Let $Am_M(Int)$ be an M amoeboid with interface Int. $Am_M(Int)$ has at least one transition of the following form for each action a of arity k:*

$$\begin{aligned} Int \vdash Am_M(Int) &\xrightarrow{(x,a,\vec{x}),(y,\overline{a},\vec{y}),\mathrm{id}} \\ &\to Int, \mathrm{n}(\vec{x}), \mathrm{n}(\vec{y}) \vdash Am_M(Int)|\ \|_{i\in\{1,\ldots,k\}} Am_M(\{\vec{x}[i],\vec{y}[i]\}) \end{aligned}$$

where $\vec{x}$ and $\vec{y}$ are vectors of new names with $|\vec{x}| = |\vec{y}| = k$.

Proof. The proof is by induction on the number of edges in the amoeboid, by adding edges in such a way that at each step the considered graph is connected. □

We thus have the following partial correctness result.

Theorem 3.8. *For each pair of sets of computations* $(C_c, C_{2s}) \in \{(c{-}1, 2s{-}1), (c{-}all, 2s{-}all)\}$ *there exists a translation function* f *from sets of productions to sets of productions such that for each set of productions* $\mathcal{P}$ *and for each closed graph* $\emptyset \vdash G$ *the following inclusion holds:*

$$C_c\text{-behav}^M(\mathcal{P})(\emptyset \vdash G) \subseteq [\![C_{2s}\text{-behav}^M(f(\mathcal{P}))([\![\emptyset \vdash G]\!]_M)]\!]^{-1}$$

Proof. The proof is similar to the one of Theorem 3.7, using Lemma 3.6 instead of Lemma 3.5. The only difference is that now we are only able to prove that the required interactions can be simulated, but not to prove that no other interaction is possible. □

The other inclusion holds, e.g., for amoeboids connecting at most 3 nodes, since in that case we can have at most one synchronization. Notice that this theorem can be composed with Theorem 3.6 to have an implementation of Milner synchronization using Hoare synchronization, at least in the case of nodes shared by at most 3 tentacles.

As usual maximal computations are not preserved.

We show now that the problem of guaranteeing interleaving inside amoeboids of the above seen kind can not be solved.

Theorem 3.9. *Let* $\mathcal{G} \subseteq$ *Graphs contain for each* $n \in \mathbb{N}$ *at least a graph with* n *nodes in its interface and let it be closed under composition of graphs by joining them by merging pairs of nodes in their respective interfaces, or by connecting them using graphs with two nodes in their interface taken from a fixed set. Then the maximum* k *such that all* $\Gamma \vdash G \in \mathcal{G}$ *allow only (HSHR or MSHR) transitions where at most* k *actions on the interface are not* ϵ*, if it exists is* 0.

Proof. We consider first the case where the graphs are connected by merging nodes. Suppose that such a k exists and it is not 0 and take a graph with more than k nodes in its interface, and a transition where k of the actions are not ϵ. Take a node where action ϵ is executed. By connecting two such graphs by merging these two nodes, we get a graph which allows at least $2k$ non ϵ actions on its interface. This gives a contradiction.

If graphs are connected using another graph, just note that the idle transition for that graph can be used to derive a transition like the one above, obtaining again a contradiction. □

This proves that we can not have a set of amoeboids for Milner synchronization (since this requires $k = 2$), since the closure property is needed to model mobility.

Notice in fact that if we want to model transitions without mobility we can use, e.g., amoeboids with a tree structure whose leaves are the elements in the interface of the amoeboid and whose roots check that the resulting action is a τ. When mobility is allowed, the tree shape can not be preserved.

To summarize, we have proved that Hoare and Milner synchronizations have a different expressive power, and that Milner synchronization is more expressive.

Chapter 4

Mapping HSHR into Logic Programming

In this chapter we analyze the relationships between HSHR and Logic Programming, which is used, as said in the introduction, as a mechanism for modeling concurrent interacting systems, which are represented as goals. Interaction is performed using the unification engine, and shared variables are used to communicate information. We introduce two variations of Logic Programming: Synchronized Logic Programming (SLP) and Logic Programming with hiding. The former introduces a mechanism of transactions inside Logic Programming, in the style of zero-safe nets [BM00], to allow more control on the execution of Logic Programming steps. In particular, in SLP we can force different atoms to synchronize before being allowed to evolve further. The latter instead adds to Logic Programming an operator of variable hiding, similar to the one found in Fusion Calculus or in SHR. We will show that this is related to the usual practice of restricting the computed answer substitution to the variables in the starting goal, and to anonymous variables as can be found in some implementations of Prolog, such as, e.g., Sicstus Prolog [Lab95]. The two presented variations of Logic Programming are actually orthogonal and they can be combined into SLP with hiding. As final result we present a mapping form HSHR into SLP with hiding, thus further extending the chain of mappings whose starting point is Fusion Calculus. As for the first step, the correspondence result is stated at the level of LTS, and here too the correspondence is essentially an injective homomorphism, showing the strict relation between the two frameworks. We will briefly discuss some generalizations of HSHR that allow to extend the correspondence to larger subsets of Logic Programming with hiding.

4.1 Synchronized Logic Programming

In this section we present the first variation of Logic Programming, called Synchronized Logic Programming (SLP).

We have introduced SLP because Logic Programming allows for many execution strategies and for complex interactions. In some cases, one wants more control on the order of execution of different SLD-steps, in order to have a more synchronous behaviour. This is obtained by grouping the SLD-steps, and forcing all the SLD-steps in the same group to be executed before the execution of the SLD-steps in next group can begin. Essentially, this is obtained by introducing transactions, which are called big-steps in this setting. The approach is similar to the zero-safe nets approach [BM00] for Petri nets. In particular, function symbols are considered resources, and they must be produced and consumed inside the same transaction. A transaction can thus end only when the goal contains just predicates applied to variables. No atom created inside a transaction can be rewritten inside the same transaction. If some SLD-steps are performed, but there is no follow-up leading the transaction to a successful completion, then the computation is not allowed (from an operational point of view, backtracking must be performed). A computation is thus a sequence of big-steps.

We start by introducing goal-graphs, which represent the states of the system when there are no ongoing transactions. The name goal-graph has been chosen since goal-graphs are the result of the translation of HSHR graphs into SLP.

Definition 4.1 (Goal-graphs). *A* goal-graph *is a goal which contains no function symbols.*

Note that a goal-graph can not contain constant symbols too, since constant symbols are just function symbols whose arity is 0.

Programs in SLP are called synchronized programs, to stress the introduction of the synchronization mechanism.

Definition 4.2 (Synchronized programs). *A* synchronized program *is a finite set of synchronized rules, i.e. clauses such that:*

1. *the body of each rule is a goal-graph;*
2. *the head of each rule has the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol and each t_i, $i \in \{1, \ldots, n\}$ is either a term obtained by applying a function symbol to a tuple of variables, or it is a variable;*
3. *if the head of a rule has the form $p(t_1, \ldots, t_n)$ and there is a variable x and an index $i \in \{1, \ldots, n\}$ such that $x = t_i$, then x occurs also in the body of the rule.*

Next example clarifies the above constraints, while their motivations will be clear when we explain the synchronization mechanism.

Example 4.1. *We show here some clauses, and we discuss whether they are synchronized rules or not.*

$q(f(x), y) \leftarrow p(x, y)$	*synchronized rule;*
$q(f(x), g(y)) \leftarrow p(x, f(y))$	*not synchronized because* $p(x, f(y))$ *is not a goal-graph;*
$q(g(f(x)), g(y)) \leftarrow p(x, y)$	*not synchronized because it contains nested functions;*
$r(g(x), z, g(y)) \leftarrow p(x, x)$	*not synchronized because* z *is an argument of* r *and it does not occur in the body.*

Next definition formalizes the concept of big-step.

Definition 4.3 (Big-steps). *Given a synchronized program* P *and a goal-graph* G_1*, a* big-step *allowed by program* P *starting from* G_1 *is a SLD-computation* $P \Vdash G_1 \xrightarrow{\theta'}{}^{*} G_2$ *such that:*

- *both* G_1 *and* G_2 *are goal-graphs;*
- *clauses are never applied to atoms introduced by previous steps.*

We denote this big-step with:

$$P \Vdash G_1 \overset{\theta}{\Rightarrow} G_2$$

where $\theta = \theta'|_{\mathrm{n}(G_1)}$*. We will omit* $P \Vdash$ *if* P *is clear from context.*

A SLP computation is a sequence of big-steps.

Definition 4.4 (Synchronized Logic Programming computation). *A* SLP computation *is a sequence of big-steps such that for each* i *the final goal-graph of big-step* i *is the starting goal-graph of big-step* $i + 1$*.*

Notation. *We denote a SLP computation as* $G_1 \overset{\theta}{\Rightarrow}{}^{*} G_2$*, where* θ *is the composition of the substitutions computed by the different big-steps, in the same order of their appearance in the sequence, and the empty substitution if the sequence is empty.*

As we have already said, a big-step is a transaction composed by steps corresponding to the application of different clauses. Each clause specifies how an atom in the goal evolves. The evolution involves creating new atoms, and function symbols are used as a synchronization mechanism. When a function symbol f appears in the head of a clause, the unification procedure causes it to be substituted for some variable x. Thus occurrences of f will appear in each atom containing x. In order to derive a big-step, a goal-graph must be reached, but this is possible only if each f will disappear from the goal. This may happen only if all those symbols f are unified with other symbols f from other clauses (the last condition in Definition 4.2 ensures that f can not disappear by being unified with a variable x which is then discarded). From another point of view, atoms have to agree on which function symbol to unify with each variable. This clearly reminds Hoare synchronization, and in fact this is the key for the translation from HSHR to SLP. Note that if a

synchronized rule does not satisfy condition 1 in Definition 4.2 then it can not be used in a successful big-step since there is no way to delete the function symbol in the body from the resulting goal. The second condition forbids structured actions (see the end of this chapter for a discussion on how to release it).

We present now a simple example of big-step to show how the synchronization mechanism works. More complex examples will be presented later, after having introduced hiding and having presented the mapping from HSHR into SLP with hiding.

Example 4.2. *Let us consider a simple system with a producer $p(x, y)$, which is able to produce a resource f with two arguments on either channel x or channel y. Similarly, there is a consumer $c(z, w)$ that can take from f its second argument (notice that since the synchronization has no polarity, i.e. there is no real difference between input and output, then the difference between the producer and the consumer is minimal) and connect to it. Finally, we have two routers r, propagating the synchronization from x to z and from y to w.*

Thus the system has the form:

$$p(x, y), r(x, z), r(y, w), c(z, w)$$

The behaviour of the system is specified by the program P composed by the following synchronized rules:

$$\begin{aligned} p(f(x, n), y) &\leftarrow p(x, y) \\ p(x, f(y, n)) &\leftarrow p(x, y) \\ r(f(x, n), f(y, n)) &\leftarrow r(x, y) \\ c(f(x, n), y) &\leftarrow c'(x, y, n) \\ c(x, f(y, n)) &\leftarrow c''(x, y, n) \end{aligned}$$

Let us consider a sample big-step (we present its SLD-steps in the most intuitive order, however they can be performed in any order, provided that at the end a goal-graph is reached, and no steps from other big-steps are executed in between).

We first produce a resource, using e.g. the first clause, obtaining the step:

$$p(x, y), r(x, z), r(y, w), c(z, w) \xrightarrow{\{f(x',n)/x, y/y'\}} p(x', y), r(f(x', n), z), r(y, w), c(z, w)$$

Now we have a function symbol in the predicate representing the first router, and we must delete it:

$$\begin{aligned} p(x', y), r(f(x', n), z), r(y, w), c(z, w) &\xrightarrow{\{x''/x', n'/n, f(z', n')/z\}} \\ &\rightarrow p(x'', y), r(x'', z'), r(y, w), c(f(z', n'), w) \end{aligned}$$

Finally, the resource can be consumed:

$$\begin{aligned} p(x'', y), r(x'', z'), r(y, w), c(f(z', n'), w) &\xrightarrow{\{z''/z', n'/n'', w'/w\}} \\ &\rightarrow p(x'', y), r(x'', z''), r(y, w'), c'(z'', w', n') \end{aligned}$$

Now we have no more function symbols, thus the big-step is finished. The substitution θ obtained as observation for this big-step is:

$$\theta = \{f(x'', n')/x, f(z'', n')/z, w'/w\}$$

Note that in θ we have neither the substitutions for the variables in the clauses (such as $\{y/y'\}$) nor the ones for the intermediate results (such as $\{z''/z'\}$).

As shown, SLP is a suitable framework for modeling interacting systems with mobility, and its strong connections with Logic Programming allow to easily implement it. For instance, a toy interpreter allowing to perform SLP computations has been presented in [Lan02].

We move now to the second promised variation of Logic Programming, namely Logic Programming with hiding.

4.2 Adding an hiding operator to Logic Programming

In the usual practice of modeling interacting distributed systems, such as GC systems, frequently *local resources* have to be modeled. A local resource is a resource that is known only to a small part of the system. In particular, only components in that part can access it, and they are aware of events generated by it. From a naming point of view, the resource has a local name, and scoping rules guarantee that this is kept distinct w.r.t. other syntactically equal names of resources which are global or local to other parts of the system.

From a modeling point of view, these aspects are usually managed by adding an operator of variable (or channel, or name, ...) *hiding* or restriction. Consider for instance the scope operator in Fusion Calculus, or restriction in SHR. Such an operator declares a resource as local, it fixes its scope, and via α-conversion it guarantees that its name is kept distinct w.r.t. names of other resources (unless they are explicitly merged).

The term "hiding" refers to the fact that observations of events performed on that variable are removed from the global observation (i.e., from the label of the transition in a LTS semantics).

In the Logic Programming world, this concept has usually been neglected because Logic Programming has not been used as a model for interacting systems. This feature becomes however very important for our aims. To cover this aspect, we extend Logic Programming with a ν operator of variable hiding, taking inspiration from the ν operator of HSHR (and in fact, the mapping from HSHR to SLP with hiding will map the ν operator of HSHR into the ν operator of SLP with hiding).

We will show that this operator allows to further understand some features of traditional Logic Programming. In particular, it allows to compute the computed

(AL1) $(G_1, G_2), G_3 = G_1, (G_2, G_3)$	(AL2) $G_1, G_2 = G_2, G_1$	(AL3) $G, \Box = G$
(AL4) $\nu x\ \nu y\ G = \nu y\ \nu x\ G$	(AL5) $\nu x\ G = G$ if $x \notin \mathrm{fn}(G)$	
(AL6) $\nu x\ G = \nu y\ G\{y/x\}$ if $y \notin \mathrm{fn}(G)$		
(AL7) $\nu x\ G_1, G_2 = G_1, \nu x\ G_2$ if $x \notin \mathrm{fn}(G_1)$		

Table 4.1: Axioms for goals with hiding.

answer substitution restricted to the (free) variables of the starting goal in a step-by-step way. Also we show that hidden variables are strictly connected to anonymous variables, which can be found in some implementations of Prolog, such as, e.g., Sicstus Prolog [Lab95].

We start by extending the syntax of clauses and goals (see Definition 1.26).

Definition 4.5 (Syntax for clauses and goals with hiding). *Clauses C and goals G with hiding are defined by the following grammar:*

$$\begin{aligned} C &::= A \leftarrow G \\ G &::= \nu x\ G \mid G, G \mid A \mid \Box \end{aligned}$$

where A is a logic atom (i.e., a predicate), x is a variable, "," is the AND conjunction and $\Box$ is the empty goal. The hiding operator ν is a binder, thus free and bound variables can be defined as usual for a goal G. We require that each free variable in the body G of the clause $A \leftarrow G$ occurs also in A. The hiding operator has lower priority w.r.t. AND conjunction.

We consider goals with hiding up to the axioms in Table 4.1.

We have chosen to state the properties of goals in the table as axioms so that they induce a congruence which is essentially identical to the structural congruence on graph terms.

Notation. *To have a uniform presentation, variables will be called names in this context and we use the same notations used, e.g., for Fusion Calculus names. In particular, we define as usual the functions* $\mathrm{n}(-)$ *and* $\mathrm{fn}(-)$ *that compute the sets of names and of free names respectively.*

Axioms (AL1), (AL2) and (AL3) define respectively the associativity, commutativity and identity over $\Box$ for AND composition. Axioms (AL4) and (AL5) state that names can be hidden only once and in any order. Axiom (AL6) defines α-conversion of a goal w.r.t its hidden names. Axiom (AL7) defines the interaction between hiding and AND composition.

Note that axioms (AL4) and (AL5) allow to write just $\nu\{x_1, \ldots, x_n\}$ instead of $\nu x_1 \ldots \nu x_n$. We will drop the braces if no confusion will arise. Note also that function fn is well-defined on equivalence classes.

The following lemma holds on goals with hiding.

Lemma 4.1. *Let G be a goal with hiding. We can always write the goal in the form:*

$$G = \nu X\ A_1, \ldots, A_n$$

where X is a set of names and $A_1, \ldots, A_n$ are atoms.

Proof. Goal G can be written in this form using the axioms to extend the scope of hidden names, and using α-conversion to avoid name clashes. □

Computations on goals with hiding are very similar to standard SLD-computations, but in addition one must track which variables are hidden. Intuitively, a variable is hidden after a SLD-step iff it was either hidden before or it is new, and it has not appeared in the observable part of the substitution (in other words, it has not been extruded).

We present here some auxiliary definitions useful to formalize this idea.

Definition 4.6 (Sets of ancestors). *Given a substitution θ and a variable x the* set of ancestors *of x w.r.t. θ is:*

$$\theta^{-1}(x) = \{y | x \in \mathrm{n}(y\theta)\}$$

.

Definition 4.7 (Sets of origins). *Let G be a goal with hiding and θ a substitution. For each variable x we define the* set of origins *of x w.r.t. θ and G as:*

$$\mathrm{orig}_{\theta,G}(x) = \theta^{-1}(x) \cap \mathrm{fn}(G)$$

We can now extend the semantics of Logic Programming to goals and programs with hiding.

Definition 4.8 (SLD-steps with hiding). *Let P be a program with hiding. The allowed* SLD-steps with hiding *are given by the LTS defined by the following inference rules:*

$$\frac{H \leftarrow \nu X\ B_1, \ldots, B_k \in P \qquad \theta = \mathrm{mgu}(\{A = H\rho\})}{P \Vdash A \xrightarrow{\theta|_{\mathrm{n}(A)}}_r (\nu X\ B_1, \ldots, B_k)\rho\theta} \quad \textit{atomic goal}$$

where ρ is an injective renaming such that $\mathrm{n}(H\rho)$ contains only fresh names and A is an atom.

$$\frac{P \Vdash G \xrightarrow{\theta}_r F}{P \Vdash G, G' \xrightarrow{\theta}_r F, G'\theta} \quad \textit{conjunctive goal}$$

$$\frac{P \Vdash G \xrightarrow{\theta}_r F}{P \Vdash \nu x\ G \xrightarrow{\theta'}_r \nu X'\ F} \quad \textit{goal with hidden variables}$$

where:

- $\theta' = \theta|_{\setminus\{x\}}$;
- $X' = \{y \in \mathrm{fn}(F) \,|\, \mathrm{orig}_{\theta,G}(y) \subseteq \{x\}\}$.

We require that, when an hidden variable x occurs in a term substituted for a free variable, the representative chosen for x by the mgu is a fresh name of the computation.

The observed substitution is called the restricted mgu.

The condition on the mgu amounts to say that when a name is extruded (and this happens when it appears in a term substituted for a free name), a fresh name is chosen. If this condition would be dropped then substitutions performed on this name will affect substitutions on other occurrences of the same name, which correspond to semantically different names.

A SLD-computation with hiding is a sequence of SLD-steps with hiding.

Definition 4.9 (SLD-computations with hiding). *A* SLD-computation with hiding *is a sequence of SLD-steps with hiding allowed by some program P and starting from some goal G. If G' is the final goal then we write $P \Vdash G \xrightarrow{\theta}_r{}^* G'$ where θ is obtained by composing the restricted mgus $\theta_1, \ldots, \theta_n$ as follows. Let:*

- $\theta'_1 = \theta_1$,
- $\theta'_i = \{t_i\theta_i/x_i | t_i/x_i \in \theta'_{i-1}\} \cup \theta_i|_{\setminus \mathrm{n}(\theta'_{i-1})}$ *for each* $i \in \{2, \ldots, n\}$.

Then $\theta = \theta'_n$.

We show here a simple example of SLD-computation with hiding.

Example 4.3. *Let P be the program defined by the following clauses with hiding:*

$$\begin{array}{rcl} p(x,y) & \leftarrow & \nu z\ q(x,y,z), r(z) \\ q(x,x,z) & \leftarrow & p(x,z) \\ r(f(x,y)) & \leftarrow & p(x,y) \end{array}$$

Starting from the goal with hiding:

$$\nu x\ p(x,y)$$

we can perform the following SLD-computation with hiding:

$$\begin{array}{l} \nu x\ p(x,y) \xrightarrow{y'/y}_r \\ \qquad \to_r \nu x', z'\ q(x',y',z'), r(z') \xrightarrow{x''/y'}_r \\ \qquad \to_r \nu z''\ p(x'',z''), r(z'') \to_r \\ \qquad \to_r \nu x''', y'''\ p(x'', f(x''', y''')), p(x''', y''') \end{array}$$

The restricted substitution θ corresponding to the whole computation is $\{x''/y\}$.

Note that, even for goals without hiding operator, the observed substitution is not the standard composition of the mgus of the different steps, but it is restricted to the names in the starting goal, as proved by the following theorem.

Theorem 4.1. *Let G be a goal without hiding operator and P a program with hiding. Let P_{free} be a program without hiding obtained by removing from P all the ν operators, and choosing for previously bound names corresponding free names which were not already used in the same clause. If $P \Vdash G \xrightarrow{\theta}_r{}^* G'$ then $P_{free} \Vdash G \xrightarrow{\theta_{free}}{}^* G'_{free}$ where $\theta = \theta_{free}|_{\mathrm{n}(G)}$ and $G' = \nu X\ G'_{free}$ for some set of names X.*

Proof. The proof is by induction on the length of the computation, and each step requires a rule induction. For the base step consider that, as far as G' is concerned, the only difference between the two sets of inference rules is that the one for computations with hiding adds some hiding operators. Notice also that in steps with hiding any fresh name can be chosen for bound variables (when they are extruded, or when hiding operators are removed to have the correspondence with the framework without hiding). However since there are no hidden names in the starting goal, all these names are introduced by clauses, thus the desired fresh names can be chosen by tuning the renaming ρ. This implies the second part of the thesis, since the inductive step is trivial.

Let us prove in detail the first part. The base step is trivial. Let us consider the inductive step. We have to prove the thesis for a computation of i steps. The bindings in the restricted mgu θ'_i concern only free variables of G_i. The first part of the composed substitution (see Definition 4.9) updates bindings in θ'_{i-1}, which concern names in the starting goal by inductive hypothesis. All the required updates are made, since all the variables therein are free. The second part of the built substitution concerns free names in G_i which are not in $\mathrm{n}(\theta'_{i-1})$. These names must be in G_1 since the only other free names in G_i are introduced by applying substitutions to free names, and in this case the names are in $\mathrm{n}(\theta'_{i-1})$. This proves the thesis. □

Thus adding the hiding operator allows to compute the computed answer substitution restricted to the names in the starting goal. When hiding operators are added to the starting goal too, substitutions performed on hidden variables are deleted from the computed substitution, as proved by the following theorem.

Theorem 4.2. *Let G be a goal with hiding, P a program with hiding and X a set of variables. We have $P \Vdash G \xrightarrow{\theta}_r{}^* G'$ iff $P \Vdash \nu X\ G \xrightarrow{\theta|_{\setminus X}}{}^* \nu X'\ G'$ where $X' = \{x \in \mathrm{fn}(G') | \ \mathrm{orig}_{\theta,G}(x) \subseteq \{X\}\}$.*

Proof. We prove the theorem for a single step first, by induction on the number of variables in X, and for whole computations later, by induction on the number of steps. Let us start with the forward implication. The first part of the proof is trivial.

The first part of the proof is also the base case of the second part. Let us consider the inductive case. Suppose we have a computation $P \Vdash G \xrightarrow{\theta'}_r{}^* G'' \xrightarrow{\theta''}_r G'$. By inductive hypothesis we have a computation $P \Vdash \nu X\ G \xrightarrow{\theta'|_{\setminus X}}_r{}^* \nu X''\ G''$ where $X'' = \{x \in \mathrm{fn}(G'') | \, \mathrm{orig}_{\theta',G}(x) \subseteq \{X\}\}$. Using the first part we also have $P \Vdash \nu X''\ G'' \xrightarrow{\theta''|_{\setminus X''}}_r{}^* \nu X'\ G'$ where $X' = \{x \in \mathrm{fn}(G') | \, \mathrm{orig}_{\theta'',G''}(x) \subseteq \{X''\}\}$. The composition of the substitutions is the wanted one, since the bindings deleted from the second part concern variables deleted from the first part. Also, variables in X' are the desired ones, because of the definition of set of origins.

The backward implication is trivial, since the desired computation is obtained just by removing the steps dealing with hiding. □

This is exactly what happens in Sicstus Prolog [Lab95] when a variable is declared as anonymous (also, the variable is anonymous since its name is not meaningful because of α-conversion). Summarizing, the hiding operator allows to track which variables are important in a step-by-step way.

4.3 Mapping HSHR into SLP with hiding

In this section we present a mapping from HSHR into SLP with hiding, that is into the combination of the two variations of Logic Programming presented in the previous sections. This allows to complete the comparison among the chosen models for GC systems.

In very broad terms, the correspondence is:

- graphs correspond to goal-graphs;
- HSHR productions correspond to synchronized clauses;
- HSHR computations are modeled by synchronized computations.

As we will see shortly, the correspondence can be stated at the level of LTS, it covers the full HSHR and it is quite simple. This highlights the strong connection between the two frameworks, in particular between Hoare synchronization and unification. The introduction of the synchronization mechanism of SLP into Logic Programming is fundamental to show this connection, since it forces a "lock-step" behaviour that corresponds to the one of HSHR, where each transition is derived by using one production for each edge, and each edge performs one action on each node in the interface. The introduction of the hiding operator is less fundamental, since HSHR without restriction exists [HIM00] and it corresponds to SLP without hiding. However, the chosen definition of the hiding operator in Logic Programming is motivated and justified by the possibility of extending the mapping to the whole setting.

We start the technical description of the mapping by defining the translation of graphs into goals. As announced, goal-graphs are obtained. The translation defines essentially an isomorphism between graphs and goal-graphs, mapping edge labels into predicates, nodes to names, restriction into hiding, parallel composition into AND composition and *nil* into $\Box$.

The only mismatch is that there is no correspondent in goal-graphs for the context Γ of a graph judgement $\Gamma \vdash G$. The only information that is lost because of this mismatch is the set of isolated nodes, since all the other nodes occur in G. In many cases isolated nodes are not important (e.g., since they can not create edges, and they are garbage-collected if bound), thus we think that this is not a big loss. When describing the mapping we will thus use graphs without mentioning context Γ. Notice however that the choice for clauses of fresh names done by SLP rules correspond to choose names not present in the context Γ of the corresponding graph.

For simplicity, we suppose that the set of nodes in HSHR coincides with the set of names in SLP (otherwise we need a bijective translation function). We do the same for edge labels and names of predicates.

Definition 4.10 (Translation from graphs to goal-graphs). *The translation operator $[\![-]\!]^{SHR}_{SLP}$ is defined on graphs as:*

$$\begin{aligned}
[\![L(x_1,\ldots,x_n)]\!]^{SHR}_{SLP} &= L(x_1,\ldots,x_n)\\
[\![G_1|G_2]\!]^{SHR}_{SLP} &= [\![G_1]\!]^{SHR}_{SLP}, [\![G_2]\!]^{SHR}_{SLP}\\
[\![\nu x\ G]\!]^{SHR}_{SLP} &= \nu x\ [\![G]\!]^{SHR}_{SLP}\\
[\![nil]\!]^{SHR}_{SLP} &= \Box
\end{aligned}$$

Lemma 4.2 (Correspondence of judgements and goal-graphs). *The translation operator $[\![-]\!]^{SHR}_{SLP}$ defines an isomorphism between graphs up to isolated nodes and goal-graphs.*

Proof. The proof is straightforward observing that the operator $[\![-]\!]^{SHR}_{SLP}$ defines an isomorphism between representatives of graphs and representatives of goal-graphs and the structural congruence on the two structures is the same up to the translation. $\Box$

The translation also maps HSHR productions to synchronized clauses with hiding. We suppose to have function symbols for all HSHR actions, with corresponding arities.

Definition 4.11 (Translation from productions to clauses). *The translation operator $[\![-]\!]^{SHR}_{SLP}$ is defined on HSHR productions as:*

$$[\![L(x_1,\ldots,x_n) \xrightarrow{\Lambda,\pi} G]\!]^{SHR}_{SLP} = L(a_1(x_1\pi,\vec{y}_1),\ldots,a_n(x_n\pi,\vec{y}_n)) \leftarrow [\![G]\!]^{SHR}_{SLP}$$

if $\Lambda(x_i) = (a_i,\vec{y}_i)$ for each $i \in \{1,\ldots,n\}$.

Notice that all the clauses obtained via the translation are synchronized clauses. In particular, predicate L has never simple variables as arguments, and this trivially satisfies the third condition of Definition 4.2. This ensures that all the images of the edges (in the same connected component) evolve in a lock-step way, as it happens in SHR.

The idea of the translation is that the condition given by an action $(x, a, \vec{y})$ is represented by using the term $a(x\pi, \vec{y})$ as argument in the position that corresponds to x. Notice that in this term a is a function symbol and π is a renaming. During unification, x will be bound to a renamed version of that term and, when other instances of x are found, the corresponding term must contain the same function symbol (as required by Hoare synchronization) in order to be unifiable. Furthermore the corresponding tuples of names are unified. Since x will disappear from the goal (it is substituted, and substitutions are idempotent) another name is required to represent the node that corresponds to x. The first argument of a is used to this purpose. If two nodes are merged by π then their successors are the same as required.

Observe that there is no need to translate all the productions, but translating only the generators (included, however, idle productions) is enough. In fact clauses with the required fresh names are automatically generated by the inference rule for atomic goals in Definition 4.8.

We concentrate now on the relationships between HSHR transitions and SLP big-steps. The observable substitution of the big-step contains information on Λ and π. Thus given a transition we can associate to it a substitution θ. We have different choices for θ according to where we map variables. In fact in HSHR nodes are mapped to their representatives according to π, while, in SLP, θ can not do the same since it is idempotent. The possible choices of fresh names for the variables change by an injective renaming the result of the big-step.

Definition 4.12 (Substitution associated to a transition). *Let $T = G \xrightarrow{\Lambda,\pi} G'$ be a HSHR transition. The substitution θ_ρ associated to T via the injective renaming ρ is:*

$$\theta_\rho = \{a(x\pi\rho, \vec{y}\rho)/x | x \in \mathrm{fn}(G) \wedge \Lambda(x) = (a, \vec{y})\}$$

We present now a lemma, which, even if quite technical, is interesting since it shows the synchronization conditions that are required in order to perform a big-step. We concentrate on big-steps where each argument of the head predicate of each used clause is a function symbol, since this is the kind of clauses built by the translation of HSHR productions. The lemma is fundamental to prove (but not to understand) the correspondence theorems 4.3 (correctness) and 4.4 (completeness).

Lemma 4.3. *Let $\nu X\ A_1, \ldots, A_n$ be a goal-graph with hiding.*

We want to characterize the big-step derived by unifying the synchronized clause $H_i \leftarrow B_i$ with atom A_i for each $i \in \{1, \ldots, n\}$. As a notational convention we use $x_{i,1}, \ldots, x_{i,n_i}$ to denote the arguments of A_i and $a_{i,j}(x'_{i,j}, \vec{y}'_{i,j})$ to denote the j-th argument of H_i.

Let $\rho = \mathrm{mgu}(\{x'_{i,j} = x'_{p,q}, \vec{y}'_{i,j} = \vec{y}'_{p,q} | i,p \in \{1,\ldots,n\}, j \in \{1,\ldots,n_i\}, q \in \{1,\ldots,n_p\}, x_{i,j} = x_{p,q}\})$.

There is a big-step with hiding of the form:

$$\nu X\ A_1,\ldots,A_n \stackrel{\theta}{\Rightarrow}_r \nu X'\ G_1,\ldots,G_n$$

iff $\forall i,p \in \{1,\ldots,n\}.\forall j \in \{1,\ldots,n_i\}.\forall q \in \{1,\ldots,n_p\}.x_{i,j} = x_{p,q} \Rightarrow a_{i,j} = a_{p,q}$. *Furthermore we have:*

- $\theta = \{a_{i,j}(x'_{i,j}\rho, \vec{y}'_{i,j}\rho)/x_{i,j} | i \in \{1,\ldots,n\}, j \in \{1,\ldots,n_i\}, x_{i,j} \notin X\}$;
- $G_1,\ldots,G_n = (B_1,\ldots,B_n)\rho$;
- $X' = \{x \in \mathrm{fn}(G_1,\ldots,G_n) | \mathrm{orig}_{\theta,\nu X\ A_1,\ldots,A_n}(x) \neq \emptyset\}$.

The big-step is determined up to injective renamings by the choice of the clauses and of the atoms they are applied to.

Proof. We will prove a more general result by induction on the number of "considered" atoms, that is we consider chains of computations of increasing length, where at step m considered atoms are the ones that have been expanded in steps $1,\ldots,m-1$. Without loss of generality (thanks to associativity and commutativity of AND conjunction), atoms are considered in numeric order, i.e. at step m atoms $A_1,\ldots,A_{m-1}$ have already been considered, and atom A_m becomes considered.

We will prove that a computation of the form:

$$\nu X\ A_1,\ldots,A_n \xrightarrow{\theta^r_m}_r{}^*\ \nu X'_m\ G_1,\ldots,G_n$$

where clauses are never applied to atoms introduced by previous steps, where atoms $A_1,\ldots,A_m$ have been considered, and where the atoms generated by them do not contain function symbols exists iff:

$$\forall i,p \in \{1,\ldots,m\}.\forall j \in \{1,\ldots,n_i\}.\forall q \in \{1,\ldots,n_p\}.x_{i,j} = x_{p,q} \Rightarrow a_{i,j} = a_{p,q}$$

and that furthermore:

- $G_1,\ldots,G_n = (B_1,\ldots,B_m)\rho_m, (A_{m+1},\ldots,A_n)\theta_m$;
- $\rho_m = \mathrm{mgu}(\{x'_{i,j} = x'_{p,q}, \vec{y}'_{i,j} = \vec{y}'_{p,q} | i,p \in \{1,\ldots,m\}, j \in \{1,\ldots,n_i\}, q \in \{1,\ldots,n_p\}, x_{i,j} = x_{p,q}\})$;
- $\theta^r_m = \{a_{i,j}(x'_{i,j}\rho_m, \vec{y}'_{i,j}\rho_m)/x_{i,j} | i \in \{1,\ldots,m\}, j \in \{1,\ldots,n_i\}, x_{i,j} \notin X\}$;
- $\theta_m = \theta^r_m \cup \rho_m$;
- $X'_m = \{x \in \mathrm{fn}(G_1,\ldots,G_n) | \mathrm{orig}_{\theta^r_m,\nu X\ A_1,\ldots,A_n}(x) \neq \emptyset\}$.

Note that if all the atoms are considered then we obtain the thesis.

Base case, $m = 1$) The computation has the form:

$$\nu X\ A_1, \ldots, A_n \xrightarrow{\theta_1^r}_r \nu X'_1\ (B_1, A_2, \ldots, A_n)\theta_1$$

where θ_1 is a mgu of $\{A_1 = H_1\}$ and θ_1^r is the restriction of θ_1 to the free names in $\nu X\ A_1, \ldots, A_n$. This computation exists iff:

$$\theta_1 = \text{mgu}(\{x_{1,j} = a_{1,j}(x'_{1,j}, \vec{y}'_{1,j}) | j \in \{1, \ldots, n_1\}\})$$

exists. Note that if $x_{1,j} = x_{1,q} \nRightarrow a_{1,j} = a_{1,q}$ then the mgu does not exist.

If the condition is satisfied instead we have:

$$\begin{aligned}\theta_1 = \text{mgu}(&\{x_{1,j} = a_{1,j}(x'_{1,j}, \vec{y}'_{1,j}) | j \in \{1, \ldots, n_1\}\} \cup \\ &\cup \{a_{1,j}(x'_{1,j}, \vec{y}'_{1,j}) = a_{1,q}(x'_{1,q}, \vec{y}'_{1,q}) | j, q \in \{1, \ldots, n_1\}, x_{1,j} = x_{1,q}\}) = \\ = \text{mgu}(&\{x_{1,j} = a_{1,j}(x'_{1,j}, \vec{y}'_{1,j}) | j \in \{1, \ldots, n_1\}\} \cup \\ &\cup \{x'_{1,j} = x'_{1,q}, \vec{y}'_{1,j} = \vec{y}'_{1,q} | j, q \in \{1, \ldots, n_1\}, x_{1,j} = x_{1,q}\})\end{aligned}$$

Note that the last part is exactly $\text{eqn}(\rho_1)$, thus ρ_1 is its mgu.

Thus we have:

$$\begin{aligned}\theta_1 = \text{mgu}(&\{x_{1,j} = a_{1,j}(x'_{1,j}, \vec{y}'_{1,j}) | j \in \{1, \ldots, n_1\}\} \cup \text{eqn}(\rho_1)) \\ &= \text{mgu}(\{x_{1,j} = a_{1,j}(x'_{1,j}\rho_1, \vec{y}'_{1,j}\rho_1) | j \in \{1, \ldots, n_1\}\} \cup \text{eqn}(\rho_1))\end{aligned}$$

This set of equations is in solved form, thus it has the same structure as θ_1. This proves the first part since this mgu exists. Also, ρ_1, θ_1 and its restriction θ_1^r to the free names in $\nu X\ A_1, \ldots, A_n$ have the wanted form. The structure of the body is as desired, as follows from the observation that $\theta_1 \lfloor_{\text{n}(B_1)} = \rho \lfloor_{\text{n}(B_1)}$. The last part follows from Theorem 4.2.

Inductive case, $m \Rightarrow m + 1$) Assume that:

$$\nu X\ A_1, \ldots, A_n \xrightarrow{\theta_{m+1}^r}_r{}^* \nu X'_{m+1}\ G_1, \ldots, G_n$$

is a SLD-computation with hiding where clauses are never applied to atoms introduced by previous steps, where atoms $A_1, \ldots, A_{m+1}$ are considered and where the atoms generated by them do not contain function symbols.

Let us take the subcomputation where only the first m atoms have been considered. By inductive hypothesis we have that this subcomputation exists iff:

$$\forall i, p \in \{1, \ldots, m\}. \forall j \in \{1, \ldots, n_i\}. \forall q \in \{1, \ldots, n_p\}. x_{i,j} = x_{p,q} \Rightarrow a_{i,j} = a_{p,q}$$

and that:

- $G_1, \ldots, G_n = (B_1, \ldots, B_m)\rho_m, (A_{m+1}, \ldots, A_n)\theta_m$;

- $\rho_m = \mathrm{mgu}(\{x'_{i,j} = x'_{p,q}, \vec{y}'_{i,j} = \vec{y}'_{p,q} | i,p \in \{1,\ldots,m\}, j \in \{1,\ldots,n_i\}, q \in \{1,\ldots,n_p\}, x_{i,j} = x_{p,q}\})$;
- $\theta^r_m = \{a_{i,j}(x'_{i,j}\rho_m, \vec{y}'_{i,j}\rho_m)/x_{i,j} | i \in \{1,\ldots,m\}, j \in \{1,\ldots,n_i\}, x_{i,j} \notin X\}$;
- $\theta_m = \theta^r_m \cup \rho_m$;
- $X'_m = \{x \in \mathrm{fn}(G_1,\ldots,G_n) | \, \mathrm{orig}_{\theta^r_m, \nu X \ A_1,\ldots,A_n}(x) \neq \emptyset\}$.

We consider now the atom A_{m+1}. There is a step with hiding of the form:

$$\nu X'_m \ (B_1,\ldots,B_m,A_{m+1},\ldots,A_n)\theta_m \xrightarrow{\sigma^r_{m+1}}_r \nu X'_{m+1} \ (B_1,\ldots,B_{m+1},A_{m+2},\ldots,A_n)\theta_m\sigma_{m+1}$$

where $\sigma_{m+1} = \mathrm{mgu}(\{A_{m+1}\theta_m = H_{m+1}\})$ (note that we can assume that θ_m is applied to B_{m+1} too since $\mathrm{dom}(\theta_m) \cap \mathrm{n}(B_{m+1}) = \emptyset$ since B_{m+1} has been renamed using fresh variables) and σ^r_{m+1} is the restriction of σ_{m+1} to the free variables of the starting goal of the step.

We have:

$$\begin{aligned}\sigma_{m+1} &= \mathrm{mgu}(\{A_{m+1}\theta_m = H_{m+1}\}) = \\ &= \mathrm{mgu}(\{x_{m+1,j}\theta_m = a_{m+1,j}(x'_{m+1,j}, \vec{y}'_{m+1,j}) | j \in \{1,\ldots,n_{m+1}\}\})\end{aligned}$$

For each j there are two possibilities: either the variable $x_{m+1,j}$ appears in already considered atoms or it does not. We call it an *old variable* in the first case and a *new variable* in the second one.

In the second case θ_m is the identity on that variable. In the first case instead the mgu exists iff $a_{m+1,j} = a_{p,q}$ where $a_{p,q}$ is the function symbol in the binding for $x_{m+1,j}$ in θ_m. A similar condition must be satisfied if $x_{m+1,j} = x_{m+1,k}$ for $j,k \in \{1,\ldots,n_{m+1}\}$.

Thus we will have:

$$\begin{aligned}\sigma_{m+1} &= \mathrm{mgu}(\{A_{m+1}\theta_m = H_{m+1}\}) = \\ &= \mathrm{mgu}(\{a_{p,q}(x'_{p,q}\rho_m, \vec{y}'_{p,q}\rho_m) = a_{m+1,j}(x'_{m+1,j}, \vec{y}'_{m+1,j}) | p \in \{1,\ldots,m+1\}, \\ &\quad q \in \{1,\ldots,n_p\}, j \in \{1,\ldots,n_{m+1}\}, x_{p,q} = x_{m+1,j}\} \cup \\ &\quad \cup\{x_{m+1,j} = a_{m+1,j}(x'_{m+1,j}, \vec{y}'_{m+1,j}) | j \in \{1,\ldots,n_{m+1}\}, x_{m+1,j} \text{ is new}\} = \\ &\quad = \mathrm{mgu}(\{x'_{p,q}\rho_m = x'_{m+1,j}, \vec{y}'_{p,q}\rho_m = \vec{y}'_{m+1,j} | p \in \{1,\ldots,m+1\}, q \in \{1,\ldots,n_p\}, \\ &\quad j \in \{1,\ldots,n_{m+1}\}, x_{p,q} = x_{m+1,j}\} \cup \\ &\quad \cup\{x_{m+1,j} = a_{m+1,j}(x'_{m+1,j}, \vec{y}'_{m+1,j}) | j \in \{1,\ldots,n_{m+1}\}, x_{m+1,j} \text{ is new}\})\end{aligned}$$

We can reorder it as:

$$\begin{aligned}&\mathrm{mgu}(\{x_{m+1,j} = a_{m+1,j}(x'_{m+1,j}, \vec{y}'_{m+1,j}) | j \in \{1,\ldots,n_{m+1}\}, x_{m+1,j} \text{ is new}\} \cup \\ &\qquad \cup\{x'_{p,q}\rho_m = x'_{m+1,j}, \vec{y}'_{p,q}\rho_m = \vec{y}'_{m+1,j} | p \in \{1,\ldots,m+1\}, q \in \{1,\ldots,n_p\}, \\ &\qquad j \in \{1,\ldots,n_{m+1}\}, x_{p,q} = x_{m+1,j}\})\end{aligned}$$

Note that the second part is a renaming that does not involve variables in the domain of the first part. Thus we can define the new renaming ρ_{m+1} as its mgu and put the set of equations in solved form:

$$\begin{aligned}\mathrm{mgu}(\{x_{m+1,j} = a_{m+1,j}(x'_{m+1,j}\rho_{m+1}, \vec{y}'_{m+1,j}\rho_{m+1}) \mid \\ j \in \{1,\ldots,n_{m+1}\}, x_{m+1,j} \text{ is new}\} \cup \mathrm{eqn}(\rho_{m+1}))\end{aligned}$$

Thus the mgu exists and the first part of the thesis is proved.

Let us consider the substitution $\theta_{m+1} = \theta_m\sigma_{m+1}$. Note that the renaming part of σ_{m+1} substitutes variables according to the equivalence between variables in H_{m+1} and representatives of the corresponding variables in A_{m+1} thus the composed substitution θ_{m+1} maps each variable to the representative of the equivalence class that is defined by the union of the two sets of equations as required. Furthermore bindings with complex terms (the first part of the substitution) have disjoint domains and thus the union of them is made. Bindings coming from σ_{m+1} have already the wanted representatives in the image, while to bindings in θ_m the renaming ρ_{m+1} is applied, mapping variables into the representatives of their equivalence classes. Thus θ_{m+1} has the wanted form.

We have:

$$(B_1,\ldots,B_{m+1},A_{m+2},\ldots,A_n)\theta_{m+1} = (B_1,\ldots,B_{m+1})\rho_{m+1},(A_{m+2},\ldots,A_n)\theta_{m+1}$$

as required since $(\mathrm{dom}(\theta_{m+1}) \setminus \mathrm{dom}(\rho_{m+1})) \cap \mathrm{fn}(B_1,\ldots,B_{m+1}) = \emptyset$.

The observed substitution θ^r_{m+1} is the restriction of θ_{m+1} to the free variables in the starting goal as required. Also, the set of hidden variables X'_{m+1} is the desired one thanks to Theorem 4.2.

Note that this result does not depend on the order of application of clauses and that after having chosen which clauses to apply and to which atoms it is determined up to an injective renaming (which depends on the choice of names for new variables and on the choice of representatives for the equivalence classes). □

We also need the following lemma dealing with the mgu of the union of sets of equations, taken from [Pal90].

Lemma 4.4. *Let θ_1 and θ_2 be idempotent substitutions. Then:*

$$\mathrm{mgu}(\mathrm{eqn}(\theta_1) \cup \mathrm{eqn}(\theta_2)) = \theta_1\,\mathrm{mgu}(\mathrm{eqn}(\theta_2)\theta_1)$$

Proof. See [Pal90]. □

We can now formally state the behavioural correspondence between HSHR and SLP with hiding, and prove a correctness and a completeness result.

Theorem 4.3 (Correctness). *Let $\mathcal{P}$ be a set of HSHR productions. Let P be the synchronized program obtained by translating the productions in $\mathcal{P}$ according to Definition 4.11. If:*

$$\mathcal{P} \Vdash_H G \xrightarrow{\Lambda,\pi} G'$$

then:

$$P \Vdash [\![G]\!]^{SHR}_{SLP} \overset{\theta_\rho}{\Rightarrow}_r T$$

for every ρ such that $x\rho$ is a fresh variable for each $x \in \mathrm{fn}(G)$. Furthermore θ_ρ is associated to $G \xrightarrow{\Lambda,\pi} G'$ and $T = [\![G']\!]^{SHR}_{SLP}\rho$. Finally, the big-step is obtained by applying the translation of the productions used in the HSHR derivation to the translation of the edges they were applied to.

Proof. The proof is by rule induction on the derivation of the HSHR transition.

Productions) Assume that:

$$L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} G \in \mathcal{P}$$

where $\Lambda(x_i) = (a_i, \vec{y}_i)$ for each $i \in \{1, \ldots, n\}$.

Then P contains the clause:

$$L(a_1(x_1\pi, \vec{y}_1), \ldots, a_n(x_n\pi, \vec{y}_n)) \leftarrow [\![G]\!]^{SHR}_{SLP}$$

Let us consider the goal:

$$[\![L(x_1, \ldots, x_n)]\!]^{SHR}_{SLP} = L(x_1, \ldots, x_n)$$

We can apply to it, using the rule for atomic goal, a renamed version of the above clause for each injective renaming ρ that maps each variable to a fresh one. The renamed clause:

$$L(a_1(x_1\pi, \vec{y}_1), \ldots, a_n(x_n\pi, \vec{y}_n))\rho \leftarrow [\![G]\!]^{SHR}_{SLP}\rho$$

unifies with the goal above with mgu $\theta_\rho = \{a_i(x_i\pi\rho, \vec{y}_i\rho)/x_i | i \in \{1, \ldots, n\}\}$. Note that $\theta_\rho = \theta_\rho\lfloor_{\mathrm{n}(L(x_1,\ldots,x_n))}$. By definition θ_ρ is associated to $L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} G$ as required. Observe also that the result of the computation is $[\![G]\!]^{SHR}_{SLP}\rho$ as required.

Note finally that we used as clause the translation of the production and that we applied it to the translation of the rewritten edge. This proves the thesis.

Rule (par-H)) By inductive hypothesis we have:

- $[\![G_1]\!]^{SHR}_{SLP} \overset{\theta_\rho}{\Rightarrow}_r T$ where θ_ρ is associated to $G_1 \xrightarrow{\Lambda,\pi} G_2$ and $T = [\![G_2]\!]^{SHR}_{SLP}\rho$;
- $[\![G'_1]\!]^{SHR}_{SLP} \overset{\theta'_{\rho'}}{\Rightarrow}_r T'$ where $\theta'_{\rho'}$ is associated to $G'_1 \xrightarrow{\Lambda',\pi'} G'_2$ and $T' = [\![G'_2]\!]^{SHR}_{SLP}\rho'$.

In both cases by inductive hypothesis we used as clauses the translations of the productions used in the proof of the HSHR transition applied to the translations of the edges on which the productions were applied.

Thanks to Lemma 4.3 the above big-steps exist iff $x_{i,j} = x_{p,q} \Rightarrow a_{i,j} = a_{p,q}$. Since the used variable sets are disjoint the same condition guarantees the existence of a big-step of the form:

$$[\![G_1|G'_1]\!]^{SHR}_{SLP} \xrightarrow{\theta_\rho\theta'_{\rho'}}{}_r^{\,*}\ T, T'$$

where we used as clauses the unions of the clauses used in the two smaller computations, applied to the same predicates.

We have that $\theta_\rho\theta'_{\rho'}$ is associated to $G_1|G'_1 \xrightarrow{\Lambda\cup\Lambda',\pi\cup\pi'} G_2|G'_2$.
We also have $\theta_\rho\theta'_{\rho'} = (\theta\theta')_{\rho\rho'}$ and thus:

$$T, T' = [\![G_2]\!]^{SHR}_{SLP}\rho, [\![G'_2]\!]^{SHR}_{SLP}\rho' = [\![G_2|G'_2]\!]^{SHR}_{SLP}\rho\rho'$$

as required.

Rule (merge-H)) By inductive hypothesis we have:

$$[\![G_1]\!]^{SHR}_{SLP} \overset{\theta_\varrho}{\Rightarrow}_r T$$

where θ_ρ is associated to $G_1 \xrightarrow{\Lambda,\pi} G_2$ and $T = [\![G_2]\!]^{SHR}_{SLP}\rho$ and we used as clauses the translations of the productions used in the proof of the HSHR transition applied to the translations of the edges on which the productions were applied.

Thanks to Lemma 4.3 this computation exists iff $x_{i,j} = x_{p,q} \Rightarrow a_{i,j} = a_{p,q}$ and:

- $\rho_l = \text{mgu}(\{x'_{i,j} = x'_{p,q}, \vec{y}'_{i,j} = \vec{y}'_{p,q} | i,j \in \{1,\ldots,n\},\ j \in \{1,\ldots,n_i\},\ q \in \{1,\ldots,n_p\}, x_{i,j} = x_{p,q}\}$;
- $\theta_\rho = \{a_{i,j}(x'_{i,j}\rho_l, \vec{y}'_{i,j}\rho_l)/x_{i,j} | i \in \{1,\ldots,n\}, j \in \{1,\ldots,n_i\}, x_{i,j} \notin X\}$;
- $T = \nu X'\ (B_1,\ldots,B_n)\rho_l$.

where $[\![G_1]\!]^{SHR}_{SLP} = \nu X\ A_1,\ldots,A_n$ and we used the naming conventions defined in Lemma 4.3, apart from the fact that we use ρ_l instead of the ρ therein to avoid confusion with ρ from θ_ρ. Thus here ρ maps each variable $x_{i,j}$ to $x'_{i,j}$.

We want to find a big-step that uses the same clauses as the previous one applied to the same atoms, but starting from goal $[\![G_1]\!]^{SHR}_{SLP}\sigma = [\![G_1\sigma]\!]^{SHR}_{SLP}$ instead of from $[\![G_1]\!]^{SHR}_{SLP}$. We will use an overline to denote the components of this new big-step. Thanks to Lemma 4.3 such a big-step exists iff $x_{i,j}\sigma = x_{p,q}\sigma \Rightarrow a_{i,j} = a_{p,q}$, but if $x_{i,j} = x_{p,q}$ then this holds by hypothesis, otherwise this is guaranteed by the applicability condition 1 of rule (merge-H).

We have to prove that $\overline{\theta}_{\overline{\rho}}$ is associated to $G_1\sigma \xrightarrow{\Lambda',\pi'} G_2\sigma\rho_g$ where ρ_g is used instead of ρ from rule (merge-H).

From Lemma 4.3 we have:

$$\overline{\theta}_{\overline{\rho}} = \{a_{i,j}(x'_{i,j}\overline{\rho_l}, \vec{y}'_{i,j\overline{\rho_l}})/x_{i,j}\sigma | i \in \{1,\ldots,n\}, j \in \{1,\ldots,n_i\}, x_{i,j}\sigma \notin X\}$$

where $\overline{\rho_l}$ maps each variable to the representative of the equivalence class given by:

$$\{x'_{i,j} = x'_{p,q}, \vec{y}'_{i,j} = \vec{y}'_{p,q} | i,p \in \{1,\ldots,n\}, j \in \{1,\ldots,n_i\}, q \in \{1,\ldots,n_p\},\ x_{i,j}\sigma = x_{p,q}\sigma\}$$

Since $\Lambda'(x_{i,j}\sigma) = (\Lambda(x_{i,j}))\sigma\rho_g$ we need to prove that $x'\overline{\rho_l} = y'\overline{\rho_l}$ iff $x\sigma\rho_g = y\sigma\rho_g$.

From the hypothesis and from Lemma 4.4 we have that $\sigma\rho_g = \mathrm{mgu}(\{\mathrm{n}_\Lambda(x) = \mathrm{n}_\Lambda(y) | x\sigma = y\sigma\} \cup \{x = y | x\pi = y\pi\} \cup \mathrm{eqn}(\sigma))$. The first and the third part are equal (adding primes) to the equations for $\overline{\rho_l}$. As far as the second part is concerned, nodes merged by π correspond to the same variables in the SLP setting, thus these equations become trivial. This proves that $\overline{\theta}_{\overline{\rho}}$ is associated to $G_1\sigma \xrightarrow{\Lambda',\pi'} G_2\sigma\rho_g$ as required.

By hypothesis and using Lemma 4.3 we have:

$$[\![G_2]\!]^{SHR}_{SLP}\rho = \nu X'\ (B_1, \ldots, B_n)\rho_l$$

The result of the new computation is:

$$\nu X'\ (B_1, \ldots, B_n)\overline{\rho_l}$$

Thanks to the properties of $\overline{\rho}$ we have:

$$\nu X'\ (B_1, \ldots, B_n)\overline{\rho_l} = \nu X'\ (B_1, \ldots, B_n)\rho_l\overline{\rho_l} = [\![G_2]\!]^{SHR}_{SLP}\rho\overline{\rho_l} = [\![G_2\sigma\rho_g]\!]^{SHR}_{SLP}\overline{\rho}$$

Rule (res-H)) By inductive hypothesis we have the following big-step:

$$[\![G_1]\!]^{SHR}_{SLP} \xrightarrow{\theta_\rho}_r{}^{*}\ T$$

where θ_ρ is associated to $G_1 \xrightarrow{\Lambda,\pi} G_2$ and $T = [\![G_2]\!]^{SHR}_{SLP}\rho$. Thanks to Theorem 4.2 we have:

$$[\![\nu x\ G_1]\!]^{SHR}_{SLP} \xrightarrow{\theta_\rho|_{\setminus\{x\}}}_r{}^{*}\ \nu Z\ T$$

where Z is the correct one for rule (res-H), but concerning primed variables. Thus $\theta_\rho\,|_{\setminus\{x\}}$ is associated to $\nu x\ G_1 \xrightarrow{\Lambda|_{\setminus\{x\}},\pi|_{\setminus\{x\}}} \nu Z\ G_2$ and $[\![\nu Z\ G_2]\!]^{SHR}_{SLP}\rho = \nu Z\ T$.

Rule (new-H)) By inductive hypothesis we have the following big-step:

$$[\![G_1]\!]^{SHR}_{SLP} \xrightarrow{\theta_\rho}_r{}^{*}\ T$$

where θ_ρ is associated to $G_1 \xrightarrow{\Lambda,\pi} G_2$ and $T = [\![G_2]\!]^{SHR}_{SLP}\rho$ but because $x \notin \mathrm{fn}(G_1)$ we have that θ_ρ is also associated to $G_1 \xrightarrow{\Lambda\cup\{(x,a,\vec{y})\},\pi} G_2$. The thesis follows. □

We present now the completeness result.

Theorem 4.4 (Completeness). *Let $\mathcal{P}$ be a set of productions. Let P be the synchronized program obtained by translating the productions in $\mathcal{P}$ according to Definition 4.11. If we can derive, applying a clause in P to each predicate in $[\![G]\!]^{SHR}_{SLP}$, a big-step of SLP with hiding of the form:*

$$[\![G]\!]^{SHR}_{SLP} \overset{\theta}{\Rightarrow}_r T$$

then there exist ρ, θ', Λ, π and G' such that $\theta = \theta'_\rho$ is associated to $G \xrightarrow{\Lambda,\pi} G'$. Furthermore $T = [\![G']\!]^{SHR}_{SLP}\rho$ and $\mathcal{P} \Vdash_H G \xrightarrow{\Lambda,\pi} G'$. Finally, the productions used in the HSHR derivation translate into the clauses used in the big-step and they are applied to the edges that translate into the predicates rewritten by them.

Proof. The translation $[\![G]\!]^{SHR}_{SLP}$ of graph G can be written in the form $\nu X\ A_1, \ldots, A_n$ where $A_1, \ldots, A_n$ are translations of single edges and X is a set of variables. We associate to each edge A_i an instance P_i of the HSHR production that corresponds to the clause used to rewrite it.

Let $L_1, \ldots, L_n$ be the LHSs of these productions. We choose the names for nodes in the following way: for each first occurrence of a variable in $A_1, \ldots, A_n$ we use the same name for the node in the corresponding position, while we use new names for all other occurrences. Note that there exists a renaming σ such that $(L_1, \ldots, L_n)\sigma = A_1, \ldots, A_n$ and that σ is idempotent. Note also that for each i all names in L_i are distinct. For each i we can choose P_i in such a way that for each $i, j, i \neq j$ we have $\mathrm{fn}(P_i) \cap \mathrm{fn}(P_j) = \emptyset$. As notation we use:

$$P_i = L_i \xrightarrow{\Lambda_i, \pi_i} R_i$$

Since all the productions have disjoint sets of names we can apply $n-1$ times rule (par-H) obtaining:

$$\|_{i\in\{1,\ldots,n\}} L_i \xrightarrow{\bigcup_{i\in\{1,\ldots,n\}} \Lambda_i, \bigcup_{i\in\{1,\ldots,n\}} \pi_i} \|_{i\in\{1,\ldots,n\}} R_i$$

Now we want to apply rule (merge-H) with renaming σ. We can do it since σ is idempotent. We have to verify that $x\sigma = y\sigma \Rightarrow \mathrm{act}_{\bigcup_{i\in\{1,\ldots,n\}} \Lambda_i}(x) = \mathrm{act}_{\bigcup_{i\in\{1,\ldots,n\}} \Lambda_i}(y)$. This happens thanks to Lemma 4.3. Thus we obtain a rule of the form:

$$A_1, \ldots, A_n \xrightarrow{\Lambda_{free}, \pi_{free}} G'_{free}$$

Finally we can apply rule (res-H) for each variable in X obtaining the transition:

$$\nu X\ A_1, \ldots, A_n \xrightarrow{\Lambda, \pi} G'$$

Thanks to Theorem 4.3 using program P we can derive the following big-step of SLP with hiding:

$$[\![\nu X\ A_1, \ldots, A_n]\!]^{SHR}_{SLP} \overset{\theta'_\rho}{\Rightarrow}_r T'$$

for every ρ that satisfies the freshness conditions. Furthermore θ'_ρ is associated to $\nu X\ A_1, \ldots, A_n \xrightarrow{\Lambda, \pi} G'$ and $T' = [\![G']\!]^{SHR}_{SLP}\rho$. Finally we used as clauses the translations of the productions used in the derivation of the HSHR transition. Note that $[\![\nu X\ A_1, \ldots, A_n]\!]^{SHR}_{SLP} = [\![G]\!]^{SHR}_{SLP}$. Since the result of a big-step is determined up to injective renamings by the starting goal and the used clauses (see Lemma 4.3) then we must have $\theta = \theta'_\rho\rho'$ and $T = T'\rho'$ for some injective renaming ρ'. Note that $\rho\rho'$ satisfies the freshness conditions (since variables generated in SLP are always fresh) and we also have $\theta = \theta''_{\rho\rho'}$. Thus we have that θ is associated to $G \xrightarrow{\Lambda, \pi} G'$ and that $T = T'\rho = [\![G']\!]^{SHR}_{SLP}\rho\rho'$. This proves the thesis. □

Note that the theorem above applies to big-steps where each atom in the goal is rewritten. This is needed since in SHR each edge is rewritten in each transition (using maybe an idle production), while in SLP some predicates may remain untouched. If the goal is the translation of a connected graph the condition is automatically enforced by the synchronization constraints. We can consider not rewritten predicates as corresponding to edges that perform idle transitions, but in this case the observation is empty instead of being $\{\epsilon(x')/x\}$, and variable x is preserved instead of being replaced by a fresh variable x'.

We show now an example to clarify the translation.

Example 4.4. *We present here the translation into SLP with hiding of the HSHR computation in Example 1.1.*

First of all, productions:

$$\begin{array}{lll} x, y \vdash R(x,y) & \xrightarrow{(x,\epsilon,\langle\rangle),(y,\epsilon,\langle\rangle),\mathrm{id}} & x, y \vdash \nu z\ R(x,z)|R(z,y) \\ x, y \vdash R(x,y) & \xrightarrow{(x,r,\langle w\rangle),(y,r,\langle w\rangle),\mathrm{id}} & x, y, w \vdash S(y,w) \end{array}$$

are translated into the corresponding clauses:

$$\begin{array}{rcl} R(\epsilon(x),\epsilon(y)) & \leftarrow & \nu z\ R(x,z), R(z,y) \\ R(r(x,w),r(y,w)) & \leftarrow & S(y,w) \end{array}$$

Also the translation:

$$R(\epsilon(x),\epsilon(y)) \leftarrow R(x,y)$$

of the idle production for $R(x,y)$ is used. We can thus have the SLP computation with hiding:

$$\begin{array}{l} R(x,x) \xrightarrow{\{\epsilon(x')/x\}}_r \nu y\ R(x',y), R(y,x') \xrightarrow{\{\epsilon(x'')/x'\}}_r \\ \qquad \nu y, z\ R(x'',y), R(y,z), R(z,x'') \xrightarrow{\epsilon(x''')/x''}_r \\ \qquad \nu y, z, v\ R(x''',y), R(y,z), R(z,v), R(v,x''') \xrightarrow{r(x'''',w)/x'''}_r \\ \qquad \nu y, z, v\ S(y,w), S(z,w), S(v,w), S(x'''',w) \end{array}$$

that corresponds to the HSHR computation from Example 1.1. Let us consider for instance the first big-step. It is associated to the HSHR transition $R(x,x) \xrightarrow{(x,\epsilon,\langle\rangle),\mathrm{id}} \nu y\ R(x,y)|R(y,x)$ with substitution $\theta_\rho = \{\epsilon(x')/x\}$ where $\rho = \{x'/x\}$. In fact $[\![\nu y\ R(x,y)|R(y,x)]\!]^{SHR}_{SLP}\rho = \nu y\ R(x',y), R(y,x')$, and this is the result of the big-step.

Note that to extend the correspondence to whole computations, the renamings ρ corresponding to the different big-steps must be composed.

We summarize the results of the comparison with a simple schema (Table 4.2).

HSHR	SLP	HSHR	SLP
Graph	Goal	Transition	Big-step
Edge	Atomic goal	Node	Variable
Parallel comp.	AND comp.	Restriction	Hiding
Nil	$\Box$	Synchronization	Unification
Production	Clause	Action	Function s.

Table 4.2: Comparison between HSHR and SLP with hiding.

Essentially the correspondence is given by the homomorphism between graphs and goals, with edges mapped to atomic goals, nodes to variables, parallel composition to AND composition, restriction to hiding and *nil* into $\Box$. Dynamically, HSHR transitions are modeled by big-steps, that are transactional applications of clauses which model productions. Finally, HSHR synchronization based on actions is modeled by unification of function symbols.

We conclude with some remarks on possible extensions of HSHR to mimic features of Logic Programming with hiding which are still outside the correspondence. First of all, translations of HSHR productions use only function symbols with arity at least 1, but we can allow constants too. An SLP constant corresponds in the HSHR framework to an action that consumes the node it is performed on. Since all the edges attached to the node have to synchronize by performing the action, all of them agree on deleting the node. For instance, let us consider an action a^* that corresponds to a Logic Programming constant. A simple production using this action is, e.g.:

$$x, y \vdash C(x, y) \xrightarrow{(x,a^*,\langle\rangle),\mathrm{id}} y \vdash C(y)$$

This allows to delete a free node, and this is impossible using standard SHR actions.

A more interesting extension is to allow nested functions in the head of the clauses. This corresponds to have structured actions in the HSHR setting. If we keep the constraint of having a "lock-step" behaviour then this is not a real extension, since during synchronization the complete structured action must be matched with an equal structured action performed by another edge. Thus if each structured action is substituted by a normal action with the same arguments then neither the allowed synchronizations nor their behaviour change. For instance, term $g(h(x, y), z)$ can be substituted by the term $f_{g(h(-,-),-)}(x, y, z)$. A more complex scenario is obtained if an edge produced in a transition is allowed to perform actions inside the same transition. In this case structured actions impose requirements that may be satisfied in part by one edge and in part by edges created by it, thus allowing more complex interactions. However in that case there is no guarantee of being able to compute a transition, since infinitely long transitions become possible. This happens when, while satisfying a pending constraint, the same (structured) action imposes a new

constraint to be satisfied. This situation can repeat infinitely many times.

These considerations, together with the introduction of the operator of hiding in Logic Programming, show that the comparison between HSHR and Logic Programming suggests many interesting improvements of the two analyzed frameworks.

Part II

Parametric synchronizations in a mobile framework

Roadmap to part II

In this part we introduce Synchronization Algebras with Mobility (SAMs for short), which are an abstract formalization of synchronization models extending Winskel's synchronization algebras [Win84] to cope with mobility and handling of local resources.

We start (Chapter 5) by defining SAMs, showing also how they can be used to express common synchronization policies such as message passing or priority communications. Then the relationships among different SAMs are analyzed using simple categorical tools, and categorical constructions are exploited to build complex SAMs from simpler ones.

The two following chapters present applications of SAMs to build modeling frameworks where the synchronization policy is not fixed, but it can be chosen according to the application in mind. This makes the modeling step easier, since this makes the synchronization primitives most suitable to model the system directly available, without need of implementing them. The two chosen frameworks are SHR (Chapter 6) and process calculi (Chapter 7).

In the field of SHR we present two applications. The former is called parametric SHR and it allows to combine productions using the synchronization policy specified in a fixed SAM. This allows to give a uniform presentation to HSHR and MSHR, but also to easily define new models simply by changing the used SAM. Properties can be proved in the parametric setting, ensuring that they hold in any instance (or in all the instances satisfying some constraints). The latter application is called SHR for heterogeneous systems and it exploits the formalization of synchronization policies as SAMs to allow many of them to coexist in the same model. In particular, SAMs are associated to nodes, and they rule the interactions therein. This allows to model heterogeneous systems, where interactions are based on different primitives (made available by different coexisting middlewares). Consider for instance a network with both message passing on wired links and broadcast on wireless ones. An important aspect is the dynamic management of SAMs, which includes the ability of having the SAM for a node to be decided at runtime, as result of a synchronization.

In the field of process calculi we introduce PRISMA Calculus, which is for Fusion Calculus what parametric SHR is for MSHR. We want to stress here the differences of the two kinds of applications. From a modeling point of view SHR provides a powerful and clean but "rigid" framework, since the separation of concerns (graph for the structure of the system, productions for specifying the behaviour) and the use of multi-party synchronization make the model more suggestive and expressive but also difficult to extend. Thus SHR is mainly used as an architectural model. Process calculi, at the contrary, have a simpler structure that can be easily extended by introducing new operators, and that makes them suitable to analyze the linguistic aspect of interaction. In other words, process calculi are suitable frameworks for experimenting new primitives. The choice of Fusion Calculus as starting point, even if not forced (we will discuss the changes required to use π-calculus instead), is the

most convenient for our aims. First of all, it allows an easy comparison with the SHR framework, thanks to the results in Chapter 2, concentrating on the difference between graph transformation and process calculi, not mixed to different design choices made inside the model. Furthermore the simple form of synchronization used in Fusion Calculus can be easily extended to cope with different synchronization models.

Chapter 5

Synchronization Algebras with Mobility

The comparison between HSHR and MSHR carried out in Chapter 3 suggests that the choice of the synchronization policy to be used in a modeling framework is a crucial choice, since different policies are more suitable to express in a simple way different coordination patterns. Also, implementing the primitives required to enforce a synchronization policy using primitives designed for enforcing a different policy can be quite complex, and this obfuscates the model of the system, since this will be mixed with the implementation of the required synchronization primitives. An example of this situation can be found in Chapter 3, for Hoare and Milner synchronization in SHR. A similar situation arises in [EM99], for broadcast and Milner synchronization in π-calculus style calculi. Also, the two synchronization models used in the literature for SHR (Hoare and Milner) claimed for a uniform presentation.

This chapter presents the work triggered by these considerations, which aims at building frameworks where the synchronization policy can be freely chosen. More in details, we introduce Synchronization Algebras with Mobility (SAMs for short), which are an abstract formalization of the concept of synchronization model, extending Winskel's synchronization algebras (SAs) to cope with mobility (expressed as name mobility) and handling of local resources. We show how common synchronization patterns can be modeled as SAMs. We also study the relationships among different SAMs by equipping the set of SAMs with a categorical structure. This allows to relate them, and to build complex SAMs by combining simpler ones using standard categorical constructions. Applications of SAMs in the field of SHR and of process calculi will be presented in the next chapters.

5.1 Definition of SAMs

In this section, we extend the concept of SA [Win84] to deal with certain aspects of synchronization that are necessary to model GC systems. In particular, we add to SAs mobility, handling of local resources and nondeterminism.

Technically, the introduction of mobility in SAs, which is the most challenging part of the extension, is realized having in mind name mobility, and in particular the approach used in SHR and Fusion Calculus. In particular, the fundamental operation allowed is the merge of two names to get a unique name. The choice, natural since SAMs have been firstly introduced to give a uniform presentation to HSHR and MSHR, has also some technical reasons. The main point is that this kind of primitive is symmetric, i.e., all the matched arguments are treated in the same way. This makes easier to extend the usual binary synchronization to general synchronizations where many parties interact. For instance, it is not clear how Hoare synchronization can be defined in a framework where the mobility primitive is asymmetric. We claim however that the SAM approach can be extended also to asymmetric forms of communication, such as message passing in the π-calculus style, at the price of adding some more complexity to the definition of SAMs. We will come back to this topic when describing PRISMA Calculus (see Section 7.2).

In general, we want to be able to express the synchronization among any number of components, each of them performing an action. Also, an action a can carry a tuple of parameters, whose length is determined by the arity $\mathrm{ar}(a)$ of the action. Actions from a multiset $[a_1, \ldots, a_n]$ can interact, and either they express compatible constraints, thus the system can perform a transition where these actions are executed, or they express incompatible constraints. Consider for instance the multiset of actions executed on a node by the edges connected to it in HSHR. If all the actions in the multiset are equal, then the transition is allowed, otherwise it is not. When the synchronization is performed, it affects the structure of the system, thus realizing mobility. Since mobility is modeled by merging names, which are the parameters of the actions, the SAM specifies an equivalence relation on them, which depends on the interacting actions. Actual names corresponding to equivalent formal parameters have to be merged.

In order to specify in a finite way the interactions among an unbound number of components a compositional approach is needed. In particular, we analyze the interactions between two components, combine the result with the action of a third component, and so on. In order to do that we require that the result of the synchronization of two actions is again an action with its tuple of parameters, which can thus be further composed. Also, this interaction defines an equivalence relation on the parameters of the two actions, which will then be extended with the contributions of other actions.

The SAM thus specifies how the arguments attached to the resulting action are computed (the component Mob in Definition 5.2) and which names are merged (the component $\doteq$ in Definition 5.2).

We can now start with some preliminary definitions, but first we introduce a notational shortcut.

Notation. *We denote with* $\underline{n}$ *the set* $\{1, \ldots, n\}$ *(where* $\underline{0} = \emptyset$*), while* id_n *is the identity function with domain* $\underline{n}$.

An action signature specifies the alphabet of actions, together with their arities.

Definition 5.1 (Action signature). *An* action signature $\mathbf{A}$ *is a triple* $(Act, \mathrm{ar}, \epsilon)$ *where* Act *is the set of actions,* $\epsilon \in Act$*, and* $\mathrm{ar} : Act \to \mathbb{N}$ *is the arity function satisfying* $\mathrm{ar}(\epsilon) = 0$.

Action ϵ above stands for "not taking part to the synchronization", as for standard SAs (see Definition 1.7).

A main ingredient in the formalization of SAMs is the action synchronization, which specifies an allowed pattern of interaction between two components, including the two synchronizing actions a and b and the results of the synchronization: (i) an outcoming action c, (ii) a communication function Mob that tells how the parameters of c are taken from those of a and b, (iii) an equivalence relation $\doteq$ on the parameters of a and b expressing fusions. Since actual parameters of actions are not known when the SAM is defined, Mob and $\doteq$ are defined according to the positions of the parameters in the tuples: for instance $\mathrm{Mob}(1) = \mathrm{inj}_2(3)$ means that the first parameter of c comes from the third parameter of the second action.

Definition 5.2 (Action synchronization).
Given an action signature $\mathbf{A} = (Act, \mathrm{ar}, \epsilon)$*, an* action synchronization *on* $\mathbf{A}$ *is a triple of the form* $(a, b, (c, \mathrm{Mob}, \doteq))$ *where* $a, b, c \in Act$*,* $\mathrm{Mob} : \underline{\mathrm{ar}(c)} \to \underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$ *and* $\doteq$ *is an equivalence relation on* $\underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$.

In order to ensure that the result of the synchronization of many actions does not depend on the order of composition (remember that we want the result to be determined just by the multiset of actions), we require the composition of action synchronizations to be associative and commutative. To do that we have to specify the combined effect of two action synchronizations, i.e. the effect of a synchronization among three components.

Definition 5.3 (Action synchronization composition).
Let $\alpha = (a_1, b_1, (c_1, \mathrm{Mob}_1, \doteq_1))$ *and* $\beta = (a_2, b_2, (c_2, \mathrm{Mob}_2, \doteq_2))$ *be action synchronizations where* $c_1 = a_2$*. The composition of* α *and* β *is the tuple* $(a_1, b_1, b_2, (c_2, \mathrm{Mob}_3, \doteq_3))$ *where the function* $\mathrm{Mob}_3 : \underline{\mathrm{ar}(c_2)} \to \underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *is defined by* $\mathrm{Mob}_3 = [\mathrm{Mob}_1, id_{\mathrm{ar}(b_2)}] \circ \mathrm{Mob}_2$*, and the equivalence relation* $\doteq_3$ *on* $\underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *is defined as the projection on the above specified domain of the minimal equivalence relation* R *on* $\underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(c_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *such that* $x \,\mathrm{R}\, y$ *iff* $x \doteq_1 y \vee x \doteq_2 y \vee \mathrm{Mob}_1(x) = y$.

A similar composition is defined when $c_1 = b_2$ *instead of* $c_1 = a_2$.

Figure 5.1: Equivalent action synchronizations.

An action synchronization relation is a set of action synchronizations satisfying suitable conditions.

Definition 5.4 (Action synchronization relation). *Given an action signature* $\mathbf{A} = (Act, \mathrm{ar}, \epsilon)$*, an* action synchronization relation *$ActSyn$ on* $\mathbf{A}$ *is a set of action synchronizations such that:*

1. $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActSyn \Rightarrow (c = \epsilon \Leftrightarrow a = b = \epsilon)$;
2. $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActSyn \Rightarrow (b, a, (c, \mathrm{Mob}', \doteq')) \in ActSyn$, *with* $\mathrm{Mob}'(x) = comp(\mathrm{Mob}(x))$ *for each* $x \in \underline{\mathrm{ar}(c)}$, *and* $x \doteq' y$ *iff* $comp(x) \doteq comp(y)$;
3. $(a, b, (c, \mathrm{Mob}, \doteq)), (c, d, (e, \mathrm{Mob}', \doteq')) \in ActSyn \Rightarrow \exists f \in Act, \exists (b, d, (f, \mathrm{Mob}'', \doteq'')), (a, f, (e, \mathrm{Mob}''', \doteq''')) \in ActSyn$ *such that composing* $(a, b, (c, \mathrm{Mob}, \doteq))$ *and* $(c, d, (e, \mathrm{Mob}', \doteq'))$ *gives the same result as composing* $(b, d, (f, \mathrm{Mob}'', \doteq''))$ *and* $(a, f, (e, \mathrm{Mob}''', \doteq'''))$.

Condition 1 is taken from standard SAs, and it states that action ϵ can arise only as combination of actions ϵ. This guarantees that no real action can disappear. Condition 2 extends commutativity of SAs, dealing also with parameters. In particular, for each action synchronization, another action synchronization modeling the symmetric interaction must exist. Last condition extends associativity, specifying that, when three actions a, b and d synchronize, the parameters of the resulting action e and the computed equivalence relation do not depend on the order of composition.

Having multiple action synchronizations for the same pair of interacting actions allows nondeterminism. In particular, the result of the synchronization is nondeterministically chosen among the allowed alternatives. An example of use of nondeterminism is described in Definition 5.11.

Action synchronizations $(a, b, (c, \mathrm{Mob}_1, \doteq))$ and $(a, b, (c, \mathrm{Mob}_2, \doteq))$ that differ only because $\mathrm{Mob}_1(n) \neq \mathrm{Mob}_2(n)$ but with $\mathrm{Mob}_1(n) \doteq \mathrm{Mob}_2(n)$ (see the example in Figure 5.1) are semantically equivalent. In fact, they will be isomorphic in the category of SAMs that we will define in Section 5.3.

As last step toward SAMs, we introduce a commonly used communication function and a related equivalence relation. The two definitions jointly define message passing, in the sense that they merge parameters in the same position (which are, in a synchronization between an input and an output, the actual value from the output and the receiving variable from the input) and they make the result available as parameter of the composed action. These can be used, e.g., to define the SAM for Milner synchronization (see Definition 5.8).

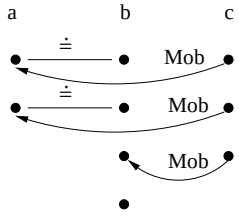

Figure 5.2: Graphical representation of an action synchronization.

Definition 5.5 (Communication function for message passing). *The communication function for message passing $MP_{i,j}$ with $i, j \in \mathbb{N}$ is the function from $\max(i,j)$ to (any superset of) $\underline{i} \uplus \underline{j}$ such that $MP_{i,j}(m) = \mathrm{inj}_1(m)$ if $m \leq i$, $\overline{MP_{i,j}(m)} = \mathrm{inj}_2(m)$ otherwise.*

Definition 5.6 (Equivalence relation for message passing). *The equivalence relation for message passing EQ_i with $i \in \mathbb{N}$ is the equivalence relation on any superset S of $\underline{i} \uplus \underline{i}$ given by $id_S \cup \{(\mathrm{inj}_1(m), \mathrm{inj}_2(m)) | m \leq i\}$.*

Example 5.1. *As an example, we draw in Figure 5.2 an action synchronization between two actions a and b of arity 2 and 4, giving an action c of arity 3, with* Mob $= MP_{2,3}$ *and* $\doteq = EQ_2$. *The first two parameters of c are obtained by merging the ones from a and b, while the third one is directly taken from b. Note that the fourth parameter of b is simply discarded.*

We can finally present the definition of SAM. We give here a very general definition, in order to introduce all the features needed for different applications (e.g., SHR, process calculi). However some of them can simply be disregarded in the simplest cases.

Definition 5.7 (Synchronization Algebras with Mobility).
A Synchronization Algebra with Mobility *over the action signature* $\mathbf{A}$ *is a quintuple* $(Id, \mathbf{A}, Init, Fin, ActSyn)$ *that includes an identifier Id taken from a suitable set of identifiers, two sets $Init, Fin \subseteq Act$ of* initial actions *and* final actions *respectively and an action synchronization relation $ActSyn$ on $\mathbf{A}$. We require:*

1. $\epsilon \in Init$;
2. $\forall i \in Init, a \in Act \setminus \{\epsilon\}, \forall (i, a, (c, \text{Mob}, \doteq)) \in ActSyn.\ c = a$, Mob $= MP_{\mathrm{ar}(i),\mathrm{ar}(a)}$, $\doteq \subseteq EQ_{\mathrm{ar}(a)}$;
3. $\forall a \in Act. \exists i \in Init. (i, a, (a, MP_{\mathrm{ar}(i),\mathrm{ar}(a)}, \doteq)) \in ActSyn$ *with* $\doteq \subseteq EQ_{\mathrm{ar}(a)}$.

The identifier Id is used to distinguish SAMs with the same structure but that can be composed in different ways (this will be used in SHR for heterogeneous systems, where different SAMs are used at the same time, and they are composed dynamically, see Section 6.2).

The set $Init$ of initial actions contains some trivial actions that can be executed by channels themselves, and they are a technical trick to deal with isolated nodes in the SHR framework. Condition 1 requires ϵ to be one of these actions. Condition 2 specifies that the synchronization of an initial action i with any non ϵ action a, if it is allowed, essentially preserves a and its parameters (some merges with parameters of i are possible, but these parameters carry no information, thus the merges are semantically irrelevant). Condition 3 requires that for each action a an action i able to synchronize with it exists. Note that condition 2 implies that for each action a at most one such initial action (different from ϵ) exists. The two conditions are required to have a behaviour for synchronization which does not depend on how it is computed, and in particular they are required to prove that bisimilarity is a congruence for parametric SHR (see Theorem 8.3 and Lemma 8.9).

Finally, the set Fin of final actions contains the actions that are considered complete, and which thus do not require any further interaction in order to be meaningful. From a technical point of view, these are the actions allowed on bound channels, and they allow to deal with local resources. For instance, in a π-calculus style synchronization, τ is a final action, while an input is not final since an output is needed to complete the interaction.

The examples in the next section will make the definition clearer.

5.2 A toolbox of SAMs

In this section we introduce different SAMs in order to show how they can be used to model interaction. We start by formalizing the synchronization models used in the previous chapters, and then we introduce some new ones. Further SAMs can be built using the composition techniques presented in next section.

Remark 5.1. *From now on, to simplify the presentation, we will not write explicitly the action synchronizations obtained by commutativity.*

We start by formalizing Milner synchronization, as it is used in MSHR and in Fusion Calculus. Also, Milner synchronization without parameters is CCS [Mil82] synchronization, and Milner synchronization where parameters of the input action are bound corresponds to π-calculus synchronization (as shown by the translation of π-calculus into Fusion Calculus in [Vic98]).

Definition 5.8 (Milner SAM). *Given a set of labels inp, the SAM* $Milner_{inp}$ *is given by:*

- $Act = \{\tau, \epsilon\} \cup \bigcup_{a \in inp} \{a, \overline{a}\}$;
- $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ *for each* $a \in inp$, $\mathrm{ar}(\tau) = 0$;
- $Init = \{\epsilon\}$;

- $Fin = \{\tau, \epsilon\}$;
- $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0)) \in ActSyn$ *for each* $\lambda \in Act$,
  $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp$.

The first action synchronization specifies that an action synchronizing with ϵ is just propagated, together with its parameters (as required since $\epsilon \in Init$). The second action synchronization formalizes the reaction of an action (input) and the corresponding coaction (output). Corresponding parameters are merged but not propagated, in fact the resulting action τ just says that a synchronization has been performed and it has arity 0.

Next definition formalizes Hoare SAM, that is the synchronization policy used in HSHR. Also, Hoare synchronization without parameters is used in CSP [Hoa80]. Here however different SAMs (determined by the label of each parallel composition operator) are used at the same time, forcing synchronization on some actions and interleaving among the others. Interleaved actions can be introduced in SAMs by allowing them to interact only with action ϵ. Many SAMs can be used together using techniques such as the ones presented in Section 6.2 for the SHR framework.

Definition 5.9 (Hoare SAM). *Given a set of labels inp, the SAM* $Hoare_{inp}$ *is given by:*

- $Act = Init = Fin = \{\epsilon\} \cup inp$;
- $(\lambda, \lambda, (\lambda, MP_{\mathrm{ar}(\lambda),\mathrm{ar}(\lambda)}, EQ_{\mathrm{ar}(\lambda)})) \in ActSyn$ *for each* $\lambda \in Act$.

The only (schema of) action synchronization in Hoare SAM models the agreement among the participants on the action to perform. During synchronization corresponding parameters are merged.

We introduce now a SAM for broadcast communication. This is the primitive used in $b\pi$-calculus [EM99], where it is combined with the name handling policy of π-calculus. We will come back to that topic in Section 7.2, when describing PRISMA Calculus.

Definition 5.10 (Broadcast SAM). *Given a set of labels inp, the SAM* Bdc_{inp} *is given by:*

- $Act = \{\epsilon\} \cup \bigcup_{a \in inp} \{a, \overline{a}\}$;
- $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ *for each* $a \in inp$;
- $Init = \{\epsilon\} \cup inp$;
- $Fin = \{\epsilon\} \cup \bigcup_{a \in inp} \{\overline{a}\}$;
- $(a, \overline{a}, (\overline{a}, MP_{\mathrm{ar}(a),\mathrm{ar}(\overline{a})}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp$,
  $(a, a, (a, MP_{\mathrm{ar}(a),\mathrm{ar}(a)}, EQ_{\mathrm{ar}(a)})) \in ActSyn$ *for each* $a \in inp$,
  $(\epsilon, \epsilon, (\epsilon, MP_{0,0}, EQ_0)) \in ActSyn$.

The main difference w.r.t. Milner SAM is that here an output can synchronize with more than one input, thus when synchronization is performed the result is the output itself, which can thus interact with further inputs. Notice also that two inputs can interact by merging their parameters, thus when an output is finally met, its parameters are merged with the ones of all the inputs. If no output is met then the resulting action is an input. Inputs do not belong to Fin, thus they are not allowed as final results of a synchronization. Notice that broadcast SAM forces all the available components to interact with an output, in fact they can not perform an action ϵ, since no action synchronization allows ϵ to interact with an output. Thus this SAM models secure broadcast, where a check is made to ensure that the broadcasted message is received by all the listeners.

Notice that, if one wants to have a multicast Mul_{inp}, where some listener may not synchronize with the output, it is enough to add the triple $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0))$ for each $\lambda \in Act$ to $ActSyn$.

Next SAM models a communication with priority, where different components produce messages with different priorities. In each transition just one component can act as receiver, and it receives only the message with the highest priority. We think that this SAM can be useful to model sensor networks, where sensors produce and send data, and the base station receives the most important data available at each time. From a technical point of view this SAM exemplifies the use of nondeterminism. Even if we can define this SAM w.r.t. a set of input actions with different arities (as done in the previous cases), for simplicity we concentrate on one input action *in* of arity 1.

Definition 5.11 (Priority SAM). *The SAM Pri for* communication with priority *is defined as follows:*

- $Act = \{in, \epsilon\} \cup \{(out, n) | n \in \mathbb{N}\} \cup \{(out+, n) | n \in \mathbb{N}\} \cup \{(out-, n) | n \in \mathbb{N}\}$*;*
- $\mathrm{ar}(\lambda) = 1$ *for all* $\lambda \in Act \setminus (\{\epsilon\} \cup \{(out+, n) | n \in \mathbb{N}\})$, $\mathrm{ar}(\lambda) = 0$ *otherwise;*
- $Init = \{\epsilon\}$*;*
- $Fin = \{\epsilon\} \cup \{(out+, n) | n \in \mathbb{N}\}$*;*
- $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0) \in ActSyn$ *for each* $\lambda \in Act$*,*
  $(in, (out, n), ((out+, n), MP_{0,0}, EQ_1)) \in ActSyn$ *for each* $n \in \mathbb{N}$*,*
  $(in, (out, n), ((out-, n), MP_{1,0}, EQ_0)) \in ActSyn$ *for each* $n \in \mathbb{N}$*,*
  $((out, n), (out, m), ((out, n), MP_{1,0}, EQ_0)) \in ActSyn$ *for each* $n, m \in \mathbb{N}$*,* $n \geq m$*,*
  $((out, n), (out-, m), ((out+, n), MP_{0,0}, EQ_1)) \in ActSyn$ *for each* $n, m \in \mathbb{N}$*,* $n \geq m$*,*
  $((out, n), (out-, m), ((out-, \max(n, m)), MP_{0,1}, EQ_0)) \in ActSyn$ *for each* n*,* $m \in \mathbb{N}$*,*
  $((out+, n), (out, m), ((out+, n), MP_{0,0}, EQ_0)) \in ActSyn$ *for each* $n, m \in \mathbb{N}$*,* $n \geq m$*.*

The above definition is a bit tricky. The basic idea is that the result of the synchronization of an action in and an action (out, n), representing an output with priority n, is determined by a guess: either we guess that n is the highest priority among all the outputs and thus we perform the merge on parameters and the result is $(out+, n)$ or we guess that an output with higher priority will be found later. In this case the input variable is propagated (while the parameter of the output is discarded) and the result is $(out-, n)$. The first guess, if wrong, is discarded when an output with higher priority is found, since $(out+, n)$ can not interact with outputs with priority higher than n. The second guess instead, if wrong, is discarded when the channel is bound, since $(out-, n) \notin Fin$. On free channels this kind of guess can never be wrong, since an output with higher priority may be provided by the environment.

A simpler approach could just merge the parameters of input and output and choose the parameter of the output with highest priority when two synchronize, without any guess. This alternative approach however is not sound, since there is no way of undoing the merge of the input parameter with the parameter of an output with low priority.

Here nondeterminism has been used for the guess but also, in a more intuitive way, to decide which output to propagate when two with the same priority interact.

Notice that some of the actions above (such as $(out-, n)$ or $(out+, n)$) are used as "intermediate results" during the computation of the effect of the synchronization, and in many scenarios one may suppose that components are not able to perform them autonomously, but that they can only emerge as results of a synchronization.

We move now to another kind of synchronization, which we call threshold synchronization. In threshold synchronization, like in multicast, a group of components receives data from a sender, but communication is allowed only if the number of receivers is above a threshold m. This synchronization can be used to enforce some security policy, e.g., when users in a group have jointly the right of accessing some information, but no one of them has enough rights on his own. As above, we consider just one basic input action $(in, 1)$ of arity 1.

Definition 5.12 (Threshold synchronization). *The SAM $Thre_m$ for* threshold synchronization *with threshold $m \in \mathbb{N}$ is defined as follows:*

- $Act = \{out, \epsilon\} \cup \{(in, n) | n \in \mathbb{N}\} \cup \{(in*, n) | n \in \mathbb{N}\}$;
- $\mathrm{ar}(\lambda) = 2$ *for all* $\lambda \in \{(in*, n) | n \in \mathbb{N}, n < m\}$, $\mathrm{ar}(\lambda) = 1$ *otherwise;*
- $Init = \{\epsilon\}$;
- $Fin = \{\epsilon\} \cup \{(in*, n) | n \in \mathbb{N}\}$;
- $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0)) \in ActSyn$ *for each* $\lambda \in Act$,
  $(out, (in, n), ((in*, n), MP_{1,1}, EQ_1)) \in ActSyn$ *for each* $n \in \mathbb{N}$, $n \geq m$,
  $(out, (in, n), ((in*, n), GAT, EQ_0)) \in ActSyn$ *for each* $n \in \mathbb{N}$, $n < m$,

$((in, n_1), (in, n_2), ((in, n_1 + n_2)), MP_{1,1}, EQ_1)) \in ActSyn$ *for each* $n_1, n_2 \in \mathbb{N}$,
$((in*, n_1), (in, n_2), ((in*, n_1 + n_2), MP_{1,1}, ALL)) \in ActSyn$ *for each* $n_1, n_2 \in \mathbb{N}$, $n_1 + n_2 \geq m$,
$((in*, n_1), (in, n_2), ((in*, n_1 + n_2), GAT, EQ_0)) \in ActSyn$ *for each* $n_1, n_2 \in \mathbb{N}$, $n_1 + n_2 < m$;

where the communication function GAT (for gathering) is defined by $GAT(1) = \mathrm{inj}_2(1)$ *and* $GAT(2) = \mathrm{inj}_1(1)$ *and the equivalence relation ALL declares that all the parameters are equivalent.*

The intuition behind SAM $Thre_m$ is that components can perform only the actions *out* and $(in, 1)$. During synchronization the inputs are merged (and n in (in, n) or $(in*, n)$ counts how many of them have been merged, while the $*$ denotes that the output has already been found) but, until m of them have joined, the parameter of the output is kept separate. Thus at least m processes have to do an input to make the piece of information in the output available. If they are not enough, synchronization is performed but no data exchange occurs.

Further SAMs can be obtained by composing the ones above using the categorical constructions described in next section.

5.3 Equipping SAMs with a categorical structure

In this section we want to analyze how different SAMs can be related and combined. This is done using basic constructions from category theory [AHS90]. In particular, we give a categorical structure to the set of SAMs, and we show that standard categorical constructions such as product and coproduct can be used to combine simple SAMs into complex ones.

In particular, we extend Winskel's definitions of categories of SAs (see Definition 1.8) by keeping into account all the features added by SAMs.

We start with the basic definition of asynchronous morphisms between action signatures. Synchronous morphisms are defined as a particular kind of asynchronous morphisms.

Definition 5.13 (Morphisms between action signatures).
Let $\mathbf{A}_A = (Act_A, \mathrm{ar}_A, \epsilon_A)$ *and* $\mathbf{A}_B = (Act_B, \mathrm{ar}_B, \epsilon_B)$ *be two action signatures. An* asynchronous morphism *H from* $\mathbf{A}_A$ *to* $\mathbf{A}_B$ *is a function* $h : Act_A \to Act_B$ *such that* $h(\epsilon_A) = \epsilon_B$, *together with a family of functions* $\{h_a | a \in Act_A\}$ *such that* $h_a : \underline{\mathrm{ar}(h(a))} \to \underline{\mathrm{ar}(a)}$. Synchronous morphisms *are asynchronous morphisms such that* $\overline{h(a) = \epsilon_B} \Leftrightarrow \overline{a = \epsilon_A}$.

Identity morphisms have identity functions as their components. Morphism composition ; *is defined by* $(h, \{h_a | a \in Act_A\}); (k, \{k_b | b \in Act_B\}) = (h \circ k, \{k_{h(a)} \circ h_a | a \in Act_A\})$.

Essentially function h specifies the mapping from actions in Act_A to actions in Act_B implementing them. The functions in $\{h_a | a \in Act_A\}$ describe how the parameters of an action $h(a)$ in the target of a morphism are taken from the parameters of the corresponding action a in the source. Note that these functions are contravariant w.r.t. the direction of H.

Morphisms between SAMs are morphisms between their action signatures preserving the additional structure.

Definition 5.14 (Morphisms between SAMs).
Let $S_A = (A, \mathbf{A}_A, Init_A, Fin_A, ActSyn_A)$ and $S_B = (B, \mathbf{A}_B, Init_B, Fin_B, ActSyn_B)$ be two SAMs. A morphism H from the S_A to S_B is a morphism $H : \mathbf{A}_A \to \mathbf{A}_B$ between the corresponding action signatures such that:

1. $a \in Fin_A \Rightarrow h(a) \in Fin_B$;
2. $a \in Init_A \Rightarrow h(a) \in Init_B$;
3. $(a_1, a_2, (c, \mathrm{Mob}_A, \doteq_A)) \in ActSyn_A \Rightarrow (h(a_1), h(a_2), (h(c), \mathrm{Mob}_B, \doteq_B)) \in ActSyn_B$ *and*
   - $\forall n \in \underline{\mathrm{ar}_B(h(c))}. \mathrm{Mob}_A(h_c(n)) = \mathrm{inj}_i(m) \Rightarrow \exists j \in \{1,2\}, m' \in \underline{\mathrm{ar}_A(a_j)}. \mathrm{Mob}_B(n) = \mathrm{inj}_j(m') \wedge \mathrm{inj}_j(h_{a_j}(m')) \doteq_A \mathrm{inj}_i(m)$;
   - $\forall i, j \in \{1,2\}, \forall n \in \underline{\mathrm{ar}_B(h(a_i))}, \forall m \in \underline{\mathrm{ar}_B(h(a_j))}. \mathrm{inj}_i(n) \doteq_B \mathrm{inj}_j(m) \Leftrightarrow \mathrm{inj}_i(h_{a_i}(n)) \doteq_A \mathrm{inj}_j(h_{a_j}(m))$.

A SAM morphism is synchronous iff the underlying action signature morphism is synchronous.

Morphisms can remove parameters or add new action synchronizations, but they must provide corresponding actions for the existing ones. Also initial and final actions are preserved, and the behaviour of existing action synchronizations is preserved too. It is interesting to notice that morphisms can change the representative chosen for a parameter inside the equivalence class defined by $\doteq$.

Proposition 5.1. *SAMs with asynchronous morphisms form the category $\mathcal{ASYNC}$, while SAMs with synchronous morphisms form the subcategory $\mathcal{SYNC}$ of $\mathcal{ASYNC}$.*

Proof. The identity laws and associativity follow trivially from the definition. It is also trivial to prove that identity morphisms are synchronous, and composition of synchronous morphisms produces synchronous morphisms. □

Next lemma characterizes the effect of isomorphisms.

Lemma 5.1. *Let S_1 and S_2 be isomorphic SAMs. Then S_1 and S_2 are equal apart from their identifiers, the names of their actions, the order of parameters of each action, and the choice for each action synchronization of the element chosen by* Mob *inside a $\doteq$-equivalence class.*

Proof. All the functions composing an isomorphism are bijections, thus they must preserve all the cardinalities (number of actions, of action synchronizations, of parameters, ...). Also, all the elements of the structure must be equal up to renaming and up to the choice for each action synchronization of the element chosen by Mob inside a $\doteq$-equivalence class, since they are preserved by the morphism, and they are reflected since they are preserved by the inverse morphism. This proves the thesis. □

We want to analyze now the product and coproduct constructions in the above defined categories. We will show that these constructions correspond to intuitive operations on SAMs, and this justifies the definition above. Also, this provides useful tools for building new SAMs in a simple way.

We start with the product construction in $\mathcal{ASYNC}$.

Lemma 5.2.
Let $(Id_1, \mathbf{A_1}, Init_1, Fin_1, ActSyn_1)$ *and* $(Id_2, \mathbf{A_2}, Init_2, Fin_2, ActSyn_2)$ *with* $\mathbf{A_1} = (Act_1, \mathrm{ar}_1, \epsilon_1)$ *and* $\mathbf{A_2} = (Act_2, \mathrm{ar}_2, \epsilon_2)$ *be two SAMs. The product in* $\mathcal{ASYNC}$*, which we call* asynchronous product*, has the form* $(Id, \mathbf{A}_\otimes, Init_\otimes, Fin_\otimes, ActSyn_\otimes)$ *with* $\mathbf{A}_\otimes = (Act_\otimes, \mathrm{ar}_\otimes, \epsilon_\otimes)$ *where:*

- $Act_\otimes = Act_1 \times Act_2$*;*
- $\mathrm{ar}_\otimes((a_1, a_2)) = \mathrm{ar}_1(a_1) + \mathrm{ar}_2(a_2)$*;*
  - *we can assume that for each action* (a_1, a_2)*, the first* $\mathrm{ar}_1(a_1)$ *parameters correspond to the ones of* a_1*, and the other ones are from* a_2*, since isomorphisms allow to permute parameters;*
- $\epsilon_\otimes = (\epsilon_1, \epsilon_2)$*;*
- $Init_\otimes = Init_1 \times Init_2$*;*
- $Fin_\otimes = Fin_1 \times Fin_2$*;*
- $ActSyn_\otimes = \{((a_1, a_2), (b_1, b_2), ((c_1, c_2), \mathrm{Mob}_\otimes, \doteq_\otimes)) |$
  $(a_1, b_1, (c_1, \mathrm{Mob}_1, \doteq_1)) \in ActSyn_1 \wedge (a_2, b_2, (c_2, \mathrm{Mob}_2, \doteq_2)) \in ActSyn_2\}$*;*
  - *for each* $i \in \{1, 2\}$*,* $\mathrm{Mob}_\otimes$ *and* $\doteq_\otimes$ *are defined on parameters from* a_i *as* Mob_i *and* $\doteq_i$ *respectively.*

The two projection morphisms map each action (a_1, a_2) *to* a_1 *and* a_2 *respectively, and they take the parameters of* a_1 *and* a_2 *from the corresponding parameters of* (a_1, a_2)*.*

Proof. We have to show that $S_\otimes$ is the product of S_1 and S_2 in $\mathcal{ASYNC}$ with the projections $P_i, i \in \{1,2\}$ described above. This amounts to show that, for any SAM S with morphisms $H_i : S \to S_i$ we have a unique morphism $H : S \to S_\otimes$ such that $H_i = H; P_i$.

If we consider just the part of the morphisms that deals with actions we have a product in the category $\mathcal{SET}$ of sets and functions, which is the cartesian product of sets. For each commuting diagram in $\mathcal{ASYNC}$, we thus have an underlying diagram on action sets. However, not all commuting diagrams in $\mathcal{SET}$ commute also in $\mathcal{ASYNC}$. Also, some non isomorphic objects in $\mathcal{ASYNC}$ may correspond to isomorphic objects in $\mathcal{SET}$. If we prove that for each diagram in $\mathcal{ASYNC}$ with the product span defined above as lower edge we can lift the arrow given by the product construction in $\mathcal{SET}$ to an arrow H in $\mathcal{ASYNC}$ then we have the thesis.

It is easy to see that the conditions on $\epsilon_\otimes$, $Init_\otimes$, $Fin_\otimes$ and $ActSyn_\otimes$ (leaving aside the parameters for now) are satisfied, and that the definitions of these four components are forced by the preservation conditions.

We have to consider parameters now. Fixing an action $a_S \in Act_S$ and considering also its images in S_1, S_2 and $S_\otimes$ along morphisms, the mappings among parameters form a coproduct diagram in the category of finite sets and functions. In that category, coproducts are disjoint unions. Each such diagram can be used to build the part dealing with parameters of one action in a SAM morphism. The choice of a particular coproduct inside its isomorphism class determines the components $\mathrm{Mob}_\otimes$ and $\doteq_\otimes$ as described above, up to the choice of the representative done by $\mathrm{Mob}_\otimes$ inside an $\doteq_\otimes$-equivalence class. □

An analogous lemma can be proved for product in $\mathcal{SYNC}$.

Lemma 5.3.
Let $(Id_1, \mathbf{A_1}, Init_1, Fin_1, ActSyn_1)$ *and* $(Id_2, \mathbf{A_2}, Init_2, Fin_2, ActSyn_2)$ *with* $\mathbf{A_1} = (Act_1, \mathrm{ar}_1, \epsilon_1)$ *and* $\mathbf{A_2} = (Act_2, \mathrm{ar}_2, \epsilon_2)$ *be two SAMs. The product in* $\mathcal{SYNC}$*, which we call* synchronous product*, has the form* $(Id, \mathbf{A}_\otimes, Init_\otimes, Fin_\otimes, ActSyn_\otimes)$ *with* $\mathbf{A}_\otimes = (Act_\otimes, \mathrm{ar}_\otimes, \epsilon_\otimes)$ *where:*

- $Act_\otimes = (Act_1 \setminus \{\epsilon_1\}) \times (Act_2 \setminus \{\epsilon_2\}) \cup \{(\epsilon_1, \epsilon_2)\}$;
- $\mathrm{ar}_\otimes((a_1, a_2)) = \mathrm{ar}_1(a_1) + \mathrm{ar}_2(a_2)$;
  - *we can assume that for each action* (a_1, a_2)*, the first* $\mathrm{ar}_1(a_1)$ *parameters correspond to the ones of* a_1*, and the other ones are from* a_2*, since isomorphisms allow to permute parameters;*
- $\epsilon_\otimes = (\epsilon_1, \epsilon_2)$;
- $Init_\otimes = (Init_1 \times Init_2) \cap Act_\otimes$;
- $Fin_\otimes = (Fin_1 \times Fin_2) \cap Act_\otimes$;

- $ActSyn_{\otimes} = \{((a_1,a_2),(b_1,b_2),((c_1,c_2),\mathrm{Mob}_{\otimes},\doteq_{\otimes}))|(a_1,b_1,(c_1,\mathrm{Mob}_1,\doteq_1)) \in ActSyn_1 \wedge (a_2,b_2,(c_2,\mathrm{Mob}_2,\doteq_2)) \in ActSyn_2 \wedge (a_1,a_2),(b_1,b_2) \in Act_{\otimes}\}$;
  - *for each* $i \in \{1,2\}$, $\mathrm{Mob}_{\otimes}$ *and* $\doteq_{\otimes}$ *are defined on parameters from* a_i *as* Mob_i *and* $\doteq_i$ *respectively.*

The two projection morphisms map each action (a_1,a_2) *to* a_1 *and* a_2 *respectively, and they take the parameters of* a_1 *and* a_2 *from the corresponding parameters of* (a_1,a_2).

Proof. Essentially the synchronous product is similar to the asynchronous one, but it has no actions of the form (a_1,ϵ_2) and (ϵ_1,a_2) except (ϵ_1,ϵ_2).

The proof is similar to the one of Lemma 5.3, but now the only action that can be mapped to ϵ_i is $\epsilon_{\otimes}$ because morphisms are synchronous. Thus the underlying diagram on action sets has $(Act_1 \setminus \{\epsilon_1\}) \times (Act_2 \setminus \{\epsilon_2\}) \cup \{(\epsilon_1,\epsilon_2)\}$ as product. In fact, since the only element in Act_S that can be mapped to ϵ_i for $i \in \{1,2\}$ is ϵ_S, and it must also be mapped to ϵ_{3-i}, then these elements are enough to make the diagram to commute (and if other elements are added, uniqueness is lost). □

The coproduct construction gives the same result in both $\mathcal{ASYNC}$ and $\mathcal{SYNC}$.

Lemma 5.4.
Let $(Id_1,\mathbf{A_1},Init_1,Fin_1,ActSyn_1)$ *and* $(Id_2,\mathbf{A_2},Init_2,Fin_2,ActSyn_2)$ *with* $\mathbf{A_1} = (Act_1,\mathrm{ar}_1,\epsilon_1)$ *and* $\mathbf{A_2} = (Act_2,\mathrm{ar}_2,\epsilon_2)$ *be two SAMs. The coproduct in* $\mathcal{ASYNC}$ *coincides with that in* $\mathcal{SYNC}$ *and it has the form* $(Id,\mathbf{A}_+,Init_+,Fin_+,ActSyn_+)$ *with* $\mathbf{A}_+ = (Act_+,\mathrm{ar}_+,\epsilon_+)$ *where:*

- $Act_+ = (Act_1 \setminus \{\epsilon_1\}) \uplus (Act_2 \setminus \{\epsilon_2\}) \uplus \{\epsilon_+\}$;
- $\mathrm{ar}_+(\mathrm{inj}_i(a)) = \mathrm{ar}_i(a)$;
- $\mathrm{inj}_i(a) \in Fin_+ \Leftrightarrow a \in Fin_i$, $\epsilon_+ \in Fin_+$ *iff* $\exists i \in \{1,2\}.\epsilon_i \in Fin_i$;
- $\mathrm{inj}_i(a) \in Init_+ \Leftrightarrow a \in Init_i$, $\epsilon_+ \in Init_+$;
- $(\mathrm{inj}_i(a),\mathrm{inj}_i(b),(\mathrm{inj}_i(c),\mathrm{Mob},\doteq)) \in ActSyn_+ \Leftrightarrow (a,b,(c,\mathrm{Mob},\doteq)) \in ActSyn_i$, $(\mathrm{inj}_i(a),\epsilon_+,(\mathrm{inj}_i(c),\mathrm{Mob},\doteq)) \in ActSyn_+ \Leftrightarrow (a,\epsilon_i,(c,\mathrm{Mob},\doteq)) \in ActSyn_i$, $(\epsilon_+,\epsilon_+,(\epsilon_+,MP_{0,0},EQ_0)) \in ActSyn_+$.

The two injection morphisms map each non ϵ *action to the corresponding action in* Act_+, *and both the actions* ϵ_1 *and* ϵ_2 *to* ϵ_+. *The parameters of an action* $\mathrm{inj}_i(a)$ *are taken from the parameters of* a.

Proof. Let us consider first the underlying diagram on action sets, as done in the proof of Lemma 5.2. This can be seen as a diagram in the category of pointed sets (with ϵ as point) and point-preserving functions. In this category the coproduct is

the disjoint union of the two sets with merged points. Thus, if a coproduct exists in $\mathcal{ASYNC}$ or in $\mathcal{SYNC}$, then it must have this underlying set. Furthermore h must be defined as $[h_1, h_2]$, apart from ϵ_+ that is mapped to ϵ_S. We have that $ActSyn_+$ must contain all the images of the action synchronization tuples from the two components in order to make the diagram commute, and no more, since otherwise we would have no morphism into some SAM S not containing them.

As far as parameters are concerned, fixing one action $\mathrm{inj}_i(a) \in Act_+$, its parameters coincide with the parameters of a. The diagram commutes with $h_{\mathrm{inj}_i(a)} = h_{ia}$, i.e. the parameters of $h(inj_i(a))$ are mapped to the parameters of $inj_i(a)$ in the same way as the parameters of $h_i(a)$ are mapped to the parameters of a by h_{ia}. Note that this definition is the unique (up to isomorphisms) that makes the diagram commute. The conditions on Mob_+ and $\doteq_+$ are satisfied by definition.

Also, actions that are final are injected into final actions. Action ϵ_+ is final iff at least one of the ϵ_i is final (in the synchronous case, both of them are final if one is, otherwise at least one of morphisms h_1 and h_2 does not exist).

Finally, actions that are initial are injected into initial actions, and ϵ_+ is initial by definition. □

We discuss now how these constructions can be applied, providing some simple examples. Coproduct allows to merge two SAMs in a unique one preserving the behaviour of each action, as shown by the example below.

Example 5.2. *Let us consider the coproduct of $Milner_{\{a,b\}}$ and $Hoare_{\{c,d\}}$. It is a SAM where actions a and b, together with their coactions and the τ action, (or, more precisely, their images along the morphisms) interact using Milner synchronization, while actions c and d interact using Hoare synchronization. Notice that there is just one action ϵ. The coproduct construction thus allows to combine different protocols inside the same SAM, and the choice of which protocol to use is done by the components, when they choose which action to perform.*

The coproduct of $Milner_{\{in_i | i \in \underline{254}\}}$ and $Bdc_{\{in_{255}\}}$ can be used to model normal TCP/IP protocol [Ste94], where address 255 is used for broadcast while other addresses are used for point-to-point communication. Naturally this is only an intuition, since far more refined techniques are necessary to deal with all the features of TCP/IP protocol.

Products instead have pairs of actions with one element for each of the component SAMs as actions, with the two elements of the pair acting in parallel.

Example 5.3. *The synchronous product of the two SAMs $Hoare_{inp_1}$ and $Hoare_{inp_2}$ is $Hoare_{inp_1 \times inp_2}$. If we consider the asynchronous product instead, also actions (ϵ_1, a_2) and (a_1, ϵ_2) can be used.*

The case of Milner communication is a bit more complex. The asynchronous product of two Milner SAMs is a message passing communication where at most two communications can happen at each step. For instance, action (a, b) can first

synchronize with $(\overline{a}, \epsilon)$ *producing* (τ, b) *and completing the first communication (i.e., parameters related to the first component are merged), and then synchronize with* $(\epsilon, \overline{b})$ *producing* (τ, τ) *and completing the second one. Notice that this is a final action.*

Instead, the synchronous product of $Milner_{inp_1}$ *and* $Milner_{inp_2}$ *is similar to* $Milner_{(inp_1 \times inp_2) \cup (inp_1 \times \{\overline{a_2} | a_2 \in inp_2\})}$*, but it has some more actions of the form* (λ_1, τ) *or* (τ, λ_2) *where* $\lambda_i \in inp_i \cup \{\overline{a_i} | a_i \in inp_i\}$ *for* $i \in \{1, 2\}$*. These actions can only synchronize with* (ϵ_1, ϵ_2)*. In particular, they have the same behaviour of* (τ, τ)*, and thus they can all be quotiented using a morphism which discards the parameters too (and which is the identity on other actions).*

In next chapter we move to the applications of SAMs in the SHR framework.

Chapter 6

Applications of SAMs to SHR

In this chapter we present two applications of SAMs in the field of SHR. The former is called parametric SHR, and it is an SHR framework that is parametric on a SAM that determines how synchronization among actions executed by different edges is performed. The aim of parametric SHR is to allow to specify systems by choosing each time the most suited synchronization model, whose primitives correspond to the ones used in the modeled system, but without having to develop the whole modeling framework from scratch. The SAM is usually determined by the middleware the system is built on.

Parametric SHR allows to give a uniform presentation to HSHR and MSHR, but also to easily define SHR models based on other interaction mechanisms. Also, properties can be studied and tools can be developed for the parametric setting, thus the results can be applied to any synchronization model without additional effort.

The second framework we present is SHR for heterogeneous systems (SHR-HS), which has been devised to model systems where heterogeneity concerns both the application and the underlying middleware, thus allowing different synchronization policies to be used in different parts of the system. Consider, e.g., a wide area network with both wired connections, which use point-to-point communications, and wireless ones, which use broadcast. Furthermore, the synchronization used is not statically fixed, but it can change over time, thus modeling, e.g., negotiations on the quality of service properties of the communication. While it appears hard to exploit this feature at the network level, where communication mechanisms are fixed by hardware constraints or by the middleware, it is fundamental to model coordination at the application level, where interaction patterns are determined in a much more dynamic way.

6.1 Parametric SHR

In this section we define the semantics of parametric SHR and we show some applications. The main ingredients of this model are a SAM, as defined in Definition 5.7,

which specifies the synchronization model used, and a set of inference rules, parametric on the above SAM, used to derive transitions from productions. Clearly, productions for parametric SHR must use actions in the set of actions Act_S of the SAM S used as parameter.

We present here the inference rules for parametric SHR.

Definition 6.1 (Inference rules for parametric SHR). *The admissible behaviours of parametric SHR are defined by the following inference rules, which are parametric on a SAM* $S = (Id, (Act, \mathrm{ar}, \epsilon), Init, Fin, ActSyn)$.

$$\textit{(par-p)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2'}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G_2'}$$

where $(\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset$.

$$\textit{(merge-p)} \quad \frac{\Gamma, x, y \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

where $\sigma = \{x/y\}$ *and:*

1. $\Lambda(x) = (a_1, \vec{v}_1), \Lambda(y) = (a_2, \vec{v}_2)$
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in ActSyn$
3. $S_1 = \{\vec{v}_{i_1}[j_1] = \vec{v}_{i_2}[j_2] \,|\, \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\}$
4. $S_2 = \{t = u | t, u \in \Gamma, x, y \wedge t\pi = u\pi\}$
5. $\rho = \mathrm{mgu}((S_1 \cup S_2)\sigma)$ *where* ρ *maps names to representatives in* Γ, x *whenever possible*
6. $\vec{w}[i] = (\vec{v}_j[k])\sigma\rho$ *if* $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$
7. $\Lambda'(z) = \begin{cases} (c, \vec{w}) \textit{ if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\sigma\rho) \textit{ for each } z \in \Gamma \end{cases}$
8. $\pi' = \rho|_{\Gamma, x}$
9. $U = \Phi\sigma\rho \setminus \Phi'$

$$\textit{(res-p)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda|_\Gamma, \pi|_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

10. $(\exists y \in \Gamma . x\pi = y\pi) \Rightarrow x\pi \neq x$

11. $\mathrm{act}_\Lambda(x) \in Fin$

12. $Z = \Phi \setminus \Phi'$

$$\textit{(new-p)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,i,\vec{y})\},\pi} \Phi, x, \mathrm{n}(\vec{y}) \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$, $i \in Init$ *and* $\vec{y}$ *is a vector of names such that* $\mathrm{n}(\vec{y}) \cap (\Gamma \cup \Phi \cup \{x\}) = \emptyset$ *and* $\mathrm{ar}(i) = |\vec{y}|$.

The main difference between these inference rules and the ones in Definition 1.23 and in Definition 1.24 is that these ones are parametric on a SAM S. Since this generalization introduces some more technical difficulties we have chosen a more modular presentation.

In particular, rule (merge-p) uses a renaming $\sigma = \{x/y\}$ instead of a generic idempotent renaming. This is just a presentational difference, since the effect of applying a general renaming can be obtained by successive applications of the simpler version of the rule. Synchronization between two actions a_1 and a_2 is allowed iff there is an action synchronization in $ActSyn$ with a_1 and a_2 as first and second field respectively (condition 2). Also, the component Mob is used to compute the parameters of the resulting action (condition 6), while $\doteq$ is used to compute the first set of equalities (condition 3) which contributes to ρ (while the second one depends on π as usual).

Also, the action performed on the bound node in rule (res-p) must belong to Fin (condition 11), while only actions in $Init$ (with a tuple of fresh names as parameters) can be performed on a new node added using rule (new-p).

Notation. *We write* $\mathcal{P} \Vdash_{p(S)} \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *iff* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *can be obtained from the productions in* $\mathcal{P}$ *using parametric inference rules instantiated with SAM* S.

We show now that parametric SHR can be instantiated in order to recover both the synchronization policies for SHR presented in the literature, which produce HSHR and MSHR frameworks.

Theorem 6.1. *Let* $\Gamma \vdash G$ *be a graph, let* $\mathcal{P}$ *be a set of productions and let inp be the set of non* ϵ *actions used in* $\mathcal{P}$. *The two following statements are equivalent:*

- $\mathcal{P} \Vdash_H \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$;
- $\mathcal{P} \Vdash_{p(Hoare_{inp})} \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$.

Proof. We will show that the instantiated parametric inference rules have the same effect of the corresponding Hoare inference rules. Rules (par-H) and (par-p) are equal. Rule (merge-H) allows to use any idempotent renaming, but this is equivalent

to allow just renamings of the form $\{x/y\}$. Since Hoare SAM has all action synchronizations of the form $(\lambda, \lambda, (\lambda, MP_{\mathrm{ar}(\lambda),\mathrm{ar}(\lambda)}, EQ_{\mathrm{ar}(\lambda)}))$ for $\lambda \in Act$ then condition 2 for (merge-p) is instantiated to condition 1 for (merge-H). Also, the definitions of ρ and Λ are instantiated to the ones for Hoare synchronization. Note that, when rule (merge-p) is instantiated with Hoare SAM, U becomes empty since no parameter is discarded.

The only differences between rules (res-p) and (res-H) and between (new-p) and (new-H) arise because in Hoare SAM any action belongs to both Fin and $Init$, and this allows to simplify the rules. □

The theorem above states that HSHR is an instance of parametric SHR with the Hoare synchronization model. An analogous theorem can be proved for MSHR, using Milner synchronization instead of Hoare.

Theorem 6.2. *Let $\Gamma \vdash G$ be a graph, let $\mathcal{P}$ be a set of productions and let inp be the set containing the input actions used in $\mathcal{P}$ and the coactions for output actions used in $\mathcal{P}$. The two following statements are equivalent:*

- $\mathcal{P} \Vdash_M \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$;
- $\mathcal{P} \Vdash_{p(Milner_{inp})} \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$.

Proof. We will show that the instantiated parametric inference rules have the same effect of the corresponding Milner inference rules. Rules (par-p) and (par-M) are equal. Rule (merge-M) allows to use any idempotent renaming, but this is equivalent to allow just renamings of the form $\{x/y\}$. Since Milner SAM allows only synchronization between pairs of complementary actions or synchronization with ϵ actions, then condition 2 for (merge-p) is instantiated to condition 1 for (merge-M). Also, the definitions of ρ and Λ are instantiated to the ones for Milner synchronization.

Furthermore rules (res-M) and (new-M) are instances of the corresponding parametric rules, determined by the fact that $Fin = \{\epsilon, \tau\}$ and $Init = \{\epsilon\}$. Since these actions have both arity 0, then conditions on parameters are simplified. □

The two theorems above show that parametric SHR indeed succeeds in providing a uniform presentation for HSHR and MSHR.

Next examples show that parametric SHR can also be used to go beyond models existing in the literature.

Example 6.1 (AGILE airport case study). *We show here a simple example concerning the airport case study [ABB+02] of the AGILE project [AGI]. Since we are not aiming at tackling the whole case study, but just at showing how parametric SHR can be applied, we will do some simplifications. For a more complete approach to this modeling problem (based on a synchronized version of double pushout) see [BCG04].*

The airport case study concentrates on modeling planes landing and tacking off at airports, with passengers boarding the planes. We model entities (which are classes

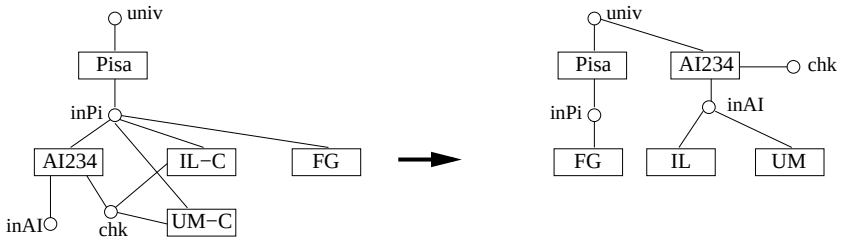

Figure 6.1: Transition of the airport case study.

in UML class diagrams [RJB99]) as edges with attributes modeled as tentacles. In particular there are edges for airports, planes and passengers. The first tentacle of each of these edges represents the attribute AtLoc, proposed in an extension of UML diagrams with mobility [BKKW02] studied inside the AGILE project. Values of attribute AtLoc represent the location containing a mobile object. Also, objects that are locations such as airports and planes have the dual attribute Containing. Furthermore, planes have an attribute CheckedIn, whose value is the set of passengers that have already checked in for next flight. Passengers that have already checked in have the dual attribute. A simple graph modeling a system of this kind is:

$$\emptyset \vdash \nu\, univ, inPi, chk, inAI\ \ Pisa(univ, inPi) | AI234(inPi, inAI, chk) | \\ IL{-}C(inPi, chk) | UM{-}C(inPi, chk) | FG(inPi)$$

featuring one airport (Pisa) located in the universe (univ), a plane (AI234) in the airport and three passengers, two which have already checked in (IL−C, UM−C), and one which has not (FG). See the left part of Figure 6.1 for a graphical representation.

We want to specify a transition that models the boarding of all the passengers who have checked in and the take off of the plane. This transition requires multiple checks and reconfigurations: essentially all the passengers who have checked in must move (changing their location), and the airport must allow the plane to take off. The plane must change its location too.

This is modeled by the following productions (which are written for generic labels AIR, PLA, PAS−C and PAS for airports, planes, checked in passengers and not checked in passengers respectively):

$$at, in \vdash AIR(at, in) \xrightarrow{(in, ack, \langle at \rangle), \mathrm{id}} at, in \vdash AIR(at, in)$$

$$at, in, chk \vdash PLA(at, in, chk) \xrightarrow{(at, req, \langle newat \rangle), (chk, breq, \langle in \rangle), \mathrm{id}} \\ \rightarrow at, in, chk, newat \vdash PLA(newat, in, chk)$$

$$at, chk \vdash PAS{-}C(at, chk) \xrightarrow{(chk, board, \langle newat \rangle), \mathrm{id}} \\ \rightarrow at, chk, newat \vdash PAS(newat)$$

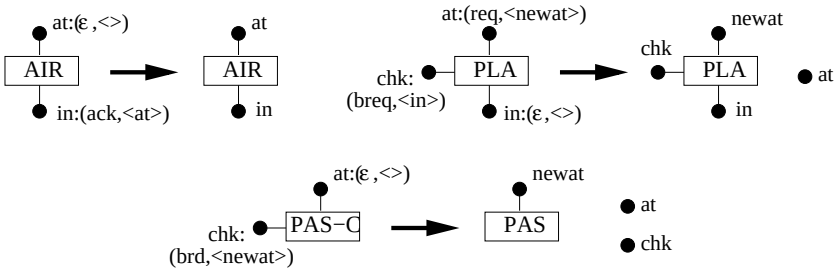

Figure 6.2: Productions for the airport case study.

The productions are graphically represented in Figure 6.2

We have now to specify the SAM S that we want to use. Actions ack and req have to synchronize using Milner synchronization, since they model a message exchange between the airport and the plane allowing the take off, while actions breq and board have to synchronize using broadcast synchronization with breq as output action, since all the checked in passengers have to board. Thus we can build the wanted SAM using the coproduct construction on a Milner SAM and a broadcast SAM.

We can thus derive the following transition (see also Figure 6.1):

$$\emptyset \vdash \nu\, univ, inPi, chk, inAI\ \ Pisa(univ, inPi) | AI234(inPi, inAI, chk) | IL{-}C(inPi, chk) | UM{-}C(inPi, chk) | FG(inPi) \rightarrow$$
$$\emptyset \vdash \nu\, univ, inPi, inAI, chk\ \ Pisa(univ, inPi) | AI234(univ, inAI, chk) | IL(inAI) | UM(inAI) | FG(inPi)$$

It is interesting to notice that, when a suitable SAM is chosen for synchronization, the implementation of the communication protocol becomes trivial. In [BCG04] instead just a simple binary synchronization is used, thus a complex procedure is required to implement broadcast. Our choice allows models at a more abstract level, as suited for modeling complex systems.

We present now a more funny, but however technically interesting, example.

Example 6.2 (The Game of Life).[1] *The Game of Life [BCG82] is a well-known cellular automaton, that is a grid of evolving cells. A cell can be empty or populated (alive). A living cell dies if it has one or no alive neighbours (loneliness) or if it has four or more alive neighbours (overpopulation) and it survives otherwise. An empty cell becomes alive if it has exactly three alive neighbours.*

In order to model the Game of Life we use edges to represent cells, with labels A for alive and E for empty. Edges are connected to their eight neighbours via shared nodes.

[1] We thanks Robin Milner, who suggested the example during a discussion at Dagstuhl seminar 04241.

At each step edges must communicate their state on each connection and, at the same time, receive the state of their neighbours. This can be done using four actions (e,e),(a,a),(e,a) *and* (a,e). *The meaning of, e.g., the last action is "I am alive, you are empty". We also introduce an action ok representing a successful synchronization. Since the structure of the system is static we have no mobility, i.e. all actions have arity* 0. *We use the following SAM:*

- $Act = \{(e,e),(a,a),(e,a),(a,e),ok,\epsilon\}$;
- $\mathrm{ar}(\lambda) = 0$ *for each* $\lambda \in Act$;
- $Init = \{\epsilon\}$;
- $Fin = \{ok\}$;
- $(\lambda, \epsilon, (\lambda, MP_{0,0}, EQ_0)) \in ActSyn$ *for each* $\lambda \in Act$,
  $((e,e),(e,e),(ok, MP_{0,0}, EQ_0)) \in ActSyn$,
  $((e,a),(a,e),(ok, MP_{0,0}, EQ_0)) \in ActSyn$,
  $((a,a),(a,a),(ok, MP_{0,0}, EQ_0)) \in ActSyn$.

In order to force complete synchronizations all nodes must be bound. We use for productions a simplified notation: we just write the labels of edges and the sequence of actions. Some sample productions are:

loneliness: $A \xrightarrow{(a,e),(a,a),(a,e),(a,e),(a,e),(a,e),(a,e),(a,e),(a,e)} E$

survive: $A \xrightarrow{(a,e),(a,a),(a,e),(a,a),(a,e),(a,a),(a,e),(a,e),(a,e)} A$

overpopulation: $A \xrightarrow{(a,a),(a,a),(a,e),(a,a),(a,e),(a,a),(a,a),(a,a),(a,e)} E$

populate: $E \xrightarrow{(e,a),(e,e),(e,e),(e,a),(e,e),(e,e),(e,e),(e,a),(e,e)} A$

On each node the synchronization ensures that the edges correctly exchange information on their previous states. Thus the transition is determined by the actual state of the graph as desired. Clearly, productions are required for each allowed combination of actions. Some more productions are required also for edges on the borders, which have different rank.

6.2 SHR for heterogeneous systems

In this section we introduce a further generalization of SHR aimed at modeling heterogeneous systems (SHR-HS). In particular we consider systems where different subsystems exploit different middlewares, thus their synchronization protocols are built using different primitives. From a technical point of view heterogeneity is introduced into nodes: each node, which models a communication port of the system,

is labeled with a SAM that specifies how synchronization is performed on that node. At a first sight it may seem that the same results can be obtained using parametric SHR instantiated with the coproduct of the SAMs modeling the different used middlewares. However this is not the case, since in this way each primitive is made available on each node, while this does not happen in the modeled system.

Thus, SHR-HS concentrates on the management of the primitives available at each time on each node. In particular, the SAM labeling a node can change dynamically as a result of a synchronization among different parties. Technically, this corresponds to update the labeling when the graph evolves and nodes are merged or created. Some practical modeling experiments have shown that in different circumstances, different strategies have to be followed: in particular, at the network level the labeling is quite static, since it depends on hardware features, while at the application level it can change dynamically as a result of negotiations among different participants to the synchronization. In fact, different required services or services with different quality of service properties can be conveniently described by different SAMs. Technically, a set $\mathcal{A}\mathfrak{lg}$ of SAMs is declared, and it is equipped with a user-defined operator $\diamond$ of SAM composition, as shown by the definition below.

Definition 6.2 (SAM composition). *We assume that all the SAMs used in a SHR-HS model belong to a fixed set $\mathcal{A}\mathfrak{lg}$. Also, we assume a binary operator $\diamond : \mathcal{A}\mathfrak{lg} \times \mathcal{A}\mathfrak{lg} \to \mathcal{A}\mathfrak{lg}$ of SAM composition which is associative, commutative and which has a neutral element S_ϵ. Finally, identifiers of SAMs are required to be unique inside $\mathcal{A}\mathfrak{lg}$.*

The definition above requires $\langle \mathcal{A}\mathfrak{lg}, \diamond, S_\epsilon \rangle$ to be a commutative monoid. Associativity and commutativity are needed so that the result of the composition of SAMs does not depend on the order of composition (which is not determined, as we will see in a while). The requirement of having a neutral element is not restrictive since one can always add an unused element and set it as neutral element of the composition. The existence of a neutral element allows to ensure that the label of a node is preserved during a merge.

Since different SAMs are used at the same time, it is useful to introduce some auxiliary functions to extract a particular field from a SAM.

Notation. *We denote with* fAct$(-)$, fInit$(-)$, fFin$(-)$ *and* fActSyn$(-)$ *the functions that given a SAM $S = (Id, (Act, \mathrm{ar}, \epsilon), Init, Fin, ActSyn)$ compute respectively its set of actions Act, its set of initial actions $Init$, its set of final actions Fin, and its action synchronization relation $ActSyn$.*

We have to extend the main definitions of SHR to keep into account the labeling of nodes. We extend the definition of graphs (Definition 1.19) first.

Definition 6.3 (Labeled graphs). *Let $\mathcal{N}$ be a fixed infinite set of names, LE a ranked alphabet of labels and $\mathcal{A}\mathfrak{lg}$ a set of SAMs. A* labeled graph *is of the form $\Gamma \vdash G$ where:*

1. Γ *is a finite function mapping names to SAMs, written as set of pairs* $n : S$ *where* $n \in \mathcal{N}$ *and* $S \in \mathcal{A}\mathfrak{lg}$*;*

2. G *is a term generated by the grammar*
   $G ::= L(\vec{x}) \mid G|G \mid \nu y : S\ G \mid nil$
   where $\vec{x}$ *is a vector of names,* L *is an edge label with* $\mathrm{rank}(L) = |\vec{x}|$*,* y *is a name and* S *is a SAM.*

We define the restriction operator ν *as a binder for* y*, which also records the label* S*, and we demand* $\mathrm{fn}(G) \subseteq \mathrm{dom}(\Gamma)$.

Notation. *When defining the interfaces, we use the notation* $\Gamma, x : S$ *to denote the set obtained by adding* $x : S$ *to* Γ*, assuming* $x \notin \mathrm{dom}(\Gamma)$ *and* Γ_1, Γ_2 *to denote the union of* Γ_1 *and* Γ_2*, assuming* $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$*. We use* $x_1, \ldots, x_n : S$ *to shorten* $x_1 : S, \ldots, x_n : S$.

The relation of structural congruence on graph terms is the standard one (see Definition 1.20), with axioms preserving the labeling SAMs. The corresponding notion of graph isomorphism also preserves SAMs that label nodes.

We can extend now the definition of transition (Definition 1.21).

Definition 6.4 (SHR-HS transitions).
Let Act be a set of actions and let $\langle \mathcal{A}\mathfrak{lg}, \diamond, S_\epsilon \rangle$ *be a monoid of SAMs. A* SHR-HS transition *is a relation of the form:*

$$\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$$

where $\Gamma \vdash G$ *and* $\Phi \vdash G'$ *are graphs labeled on* $\mathcal{A}\mathfrak{lg}$*,* $\Lambda : \mathrm{dom}(\Gamma) \to (Act \times \mathcal{N}^*)$ *is a total function and* $\pi : \mathrm{dom}(\Gamma) \to \mathrm{dom}(\Gamma)$ *is an idempotent renaming. All the constraints and notations concerning* π*,* Λ *and* Φ *for SHR transitions (Definition 1.21) are assumed for* π*,* Λ *and* $\mathrm{dom}(\Phi)$ *here. We require that for any* $x \in \mathrm{dom}(\Gamma)$*,* $\mathrm{act}_\Lambda(x) \in \mathrm{fAct}(\Gamma(x))$*, that is the action performed on a node is included in the action set of the corresponding SAM.*

The definition above simply extends Definition 1.21 to labeled graphs. Thus, in particular, $\mathrm{dom}(\Gamma)$ and $\mathrm{dom}(\Phi)$ are used instead of Γ and Φ from Definition 1.21 respectively. Also, actions are required to belong to the SAM labeling the node on which they are performed.

We want to be able to specify productions that can be applied to nodes with different labels, while keeping some control on them. Hence, we introduce types.

Definition 6.5 (Types). *A* type t *is a non empty set of SAMs.*

We extend now the definition of productions (Definition 1.22), but since we have to deal also with types we introduce the additional concept of production schema too.

Definition 6.6 (SHR-HS productions). *An* SHR-HS production *is an SHR-HS transition of the form:*

$$x_1 : S_1, \ldots, x_n : S_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G$$

where $x_1, \ldots, x_n$ are all distinct and $S_1, \ldots, S_n \in \mathcal{Alg}$. There are no constraints on the SAMs associated by Φ to nodes in Γ_Λ. Instead, for each $x \in \{x_1, \ldots, x_n\}\pi$, the associated SAM $\Phi(x)$ is $S_1 \diamond \ldots \diamond S_m$ where $\{S_1, \ldots, S_m\}$ are the labels in $x_1 : S_1, \ldots, x_n : S_n$ of the names mapped to x by π.

For each edge label L of rank n and each $S_1, \ldots, S_n \in \mathcal{Alg}$ there is a special idle production $x_1 : S_1, \ldots, x_n : S_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda_\epsilon, id} x_1 : S_1, \ldots, x_n : S_n \vdash L(x_1, \ldots, x_n)$ where $\Lambda_\epsilon(x_i) = (\epsilon, \langle\rangle)$ for each i. Idle productions are included in all sets of productions, which are also closed under α-conversion of names in $\{x_1, \ldots, x_n\} \cup \mathrm{dom}(\Phi)$.

Productions impose some requirements on how the labels in the target graph are chosen. As specified in the definition, two different cases have to be considered. If the node is new (i.e., created by the production), then its label can assume any value. If the node is not new, then it occurs in $\{x_1, \ldots, x_n\}$, and it is the chosen representative of an equivalence class determined by π. In that case, its labeling SAM is obtained by composing using the operator $\diamond$ of SAM composition the labels (taken from $x_1 : S_1, \ldots, x_n : S_n$) of the merged nodes. Notice that the composition is well-defined since SAMs form a commutative monoid.

Production schemas allow to specify in a more concise way sets of productions with the same structure but with different labeling SAMs.

Definition 6.7 (SHR-HS production schemas). *A* production schema *is a generalized production that has types instead of SAMs as labels of nodes.*

Productions are derived from production schemas by specializing each type t_i into a particular SAM $S_i \in t_i$, provided that the result is a correct production.

Last condition requires in particular that the actions in Λ and the SAMs in the resulting graph satisfy the conditions from Definition 6.4 and Definition 6.6 respectively.

Also for SHR-HS we present a set of inference rules to derive transitions from productions.

Definition 6.8 (Inference rules for SHR-HS). *The admissible behaviours of SHR-HS are defined by the following inference rules, which are parametric on a monoid $\langle \mathcal{Alg}, \diamond, S_\epsilon \rangle$ of SAMs.*

$$\textit{(par-HS)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G'_1 \xrightarrow{\Lambda',\pi'} \Phi' \vdash G'_2}{\Gamma, \Gamma' \vdash G_1 | G'_1 \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G'_2}$$

where $(\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Phi)) \cap (\mathrm{dom}(\Gamma') \cup \mathrm{dom}(\Phi')) = \emptyset$.

$$\textit{(merge-HS)} \quad \frac{\Gamma, x:S, y:S \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x:S \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G'\sigma\rho}$$

where $\sigma = \{x/y\}$ *and:*

1. $\Lambda(x) = (a_1, \vec{v}_1), \Lambda(y) = (a_2, \vec{v}_2)$
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in \mathrm{fActSyn}(S)$
3. $S_1 = \{\vec{v}_{i_1}[j_1] = \vec{v}_{i_2}[j_2] \,|\, \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\}$
4. $S_2 = \{t = u | t, u \in \mathrm{dom}(\Gamma) \cup \{x, y\} \wedge t\pi = u\pi\}$
5. $\rho = \mathrm{mgu}((S_1 \cup S_2)\sigma)$ *where* ρ *maps names to representatives in* $\mathrm{dom}(\Gamma) \cup \{x\}$ *whenever possible*
6. $\vec{w}[i] = (\vec{v}_j[k])\sigma\rho$ *if* $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$
7. $\Lambda'(z) = \begin{cases} (c, \vec{w}) \text{ if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\sigma\rho) \text{ for each } z \in \mathrm{dom}(\Gamma) \end{cases}$
8. $\pi' = \rho|_{\mathrm{dom}(\Gamma)\cup\{x\}}$
9. $\mathrm{dom}(U) = (\mathrm{dom}(\Phi))\sigma\rho \setminus \mathrm{dom}(\Phi')$
10. *the label of each node* $x \in \mathrm{dom}(U) \cup \mathrm{dom}(\Phi')$ *is computed as follows:* x *is the representative according to* $\sigma\rho$ *of an equivalence class* $\{x_1, \ldots, x_n\}$ *of nodes which have in* Φ *labels* $S_1, \ldots, S_n$*. Then the label of* x *is* $S_1 \diamond \ldots \diamond S_n$

$$\textit{(res-HS)} \quad \frac{\Gamma, x:S \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x:S\ G_1 \xrightarrow{\Lambda|_{\mathrm{dom}(\Gamma)},\pi|_{\mathrm{dom}(\Gamma)}} \Phi' \vdash \nu Z\ G_2}$$

where:

11. $(\exists y \in \mathrm{dom}(\Gamma).x\pi = y\pi) \Rightarrow x\pi \neq x$
12. $\mathrm{act}_\Lambda(x) \in \mathrm{fFin}(S)$
13. $\Phi'(x) = \Phi(x)$ *for each* $x \in \mathrm{dom}(\Phi')$
14. $Z = \Phi \setminus \Phi'$

$$\textit{(new-HS)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x:S \vdash G_1 \xrightarrow{\Lambda\cup\{(x,i,\vec{y})\},\pi} \Phi, x:S, Y \vdash G_2}$$

where $x \notin \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Phi)$*,* $S \in \mathcal{A}[\mathfrak{g}$*,* $i \in \mathrm{fInit}(S)$*,* $\vec{y}$ *is a vector of nodes such that* $\mathrm{n}(\vec{y}) \cap (\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Phi) \cup \{x\}) = \emptyset$ *and* $\mathrm{ar}(i) = |\vec{y}|$*, and* Y *is a finite function mapping each name* $y \in \vec{y}$ *to* S_ϵ*.*

These rules further extend the ones in Definition 6.1, by allowing many SAMs to be used together. Notice that now contexts are functions thus both their domains (i.e., the sets of nodes) and their values (i.e., the labeling SAMs) must be kept into account.

In rule (merge-HS), the two merged nodes must have the same label S, and the interaction is performed according to one of the action synchronizations in its action synchronization relation. SAM S is also the label of the node resulting from the merge. The SAMs associated to nodes are preserved from the premise for nodes that are not merged, and are computed using the operator of SAM composition otherwise.

In rule (res-HS), node x can be restricted only if the action performed on it belongs to the set of final actions of its labeling SAM. The labels of nodes in Φ are recorded either in Φ' or in Z.

Finally, in rule (new-HS) the created node can be labeled by any SAM $S \in \mathcal{Alg}$, and only actions from $\mathrm{fInit}(S)$ can be performed on it. Parameters are labeled by SAM S_ϵ, thus when they are merged with some node z they do not alter the SAM labeling z.

Notation. *We write* $\mathcal{P} \Vdash_{HS(Mon)} \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *iff* $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ *can be obtained from the productions in* $\mathcal{P}$ *using SHR-HS inference rules instantiated with the monoid of SAMs* Mon.

Parametric SHR can be seen as a special case of SHR-HS where a unique SAM is used, as proved by the following theorem.

Theorem 6.3. *Let* $\Gamma \vdash G$ *and* $\Phi \vdash G'$ *be graphs, let* S *be a SAM and let* $\mathcal{P}$ *be a set of productions. Let* Mon_S *be the monoid with* S *as only element. Let* $\Gamma_S \vdash G_S$ *and* $\Phi_S \vdash G'_S$ *be labeled graphs obtained from* $\Gamma \vdash G$ *and* $\Phi \vdash G'$ *respectively by labeling each node with SAM* S. *Let* $\mathcal{P}_S$ *be a set of labeled productions obtained from productions in* $\mathcal{P}$ *by labeling each node with SAM* S. *The two following statements are equivalent:*

- $\mathcal{P} \Vdash_{p(S)} \Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$;
- $\mathcal{P}_S \Vdash_{HS(Mon_S)} \Gamma_S \vdash G_S \xrightarrow{\Lambda,\pi} \Phi_S \vdash G'_S$.

Proof. The proof is by induction on the derivation, showing that by instantiating all the SHR-HS rules with SAM S for each node we get the corresponding parametric rules, with the added management of labels, which labels each node with SAM S. □

We present here some examples on how to model heterogeneous systems using SHR-HS.

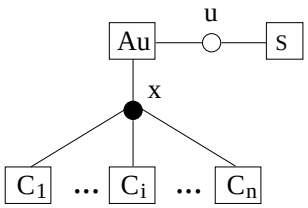

Figure 6.3: Clients waiting for authorization.

Example 6.3. *We describe here a sample scenario from the Internet realm. We start from a simple case that can be modeled also using parametric SHR and then we move to a more complex one that requires SHR-HS. Some clients $C_1, \ldots, C_n$ can invoke a service offered by a remote server S provided that they are authorized. A possible solution might be to interpose an authority Au between S and the clients. Both S and the clients trust Au, and clients get access to S only after they have been authenticated by Au.*

Figure 6.3 and the judgement:

$$x \vdash \nu u\ Au(x,u)|S(u)|C_1(x)|\ldots|C_i(x)|\ldots|C_n(x)$$

represent clients connected to Au on a free node x. The server S is also connected to the authority Au, but this time on a restricted node.

We present now the productions expressing the behaviours of Au, S and a generic client C_i. For the sake of simplicity, we consider the Milner SAM on actions $\{req\} \cup \bigcup_{i\in\{1,\ldots,n\}}\{auth_i\}$ and we call it Mil.

First consider a generic client C_i:

$$x : Mil \vdash C_i(x) \xrightarrow{(x,\overline{auth_i},\langle y\rangle),\mathrm{id}} x, y : Mil \vdash C_i'(y) \tag{6.1}$$

$$y : Mil \vdash C_i'(y) \xrightarrow{(y,\overline{req},\langle\rangle),\mathrm{id}} y : Mil \vdash C_i'(y) \tag{6.2}$$

Production 6.1 models the authentication phase where C_i is attached to a Mil node and it asks Au for the access to the service S. If the authentication takes place, C_i connects to the server through y. Notice that the synchronization between Au and C_i takes place only if C_i is able to provide some information, which is abstracted with $auth_i$, that allows Au to "recognize" C_i as a legal user. Production 6.2 simply states that a request is sent to the server. For simplicity, we assume that the server does not give back any answer.

Productions for Au simply have to accept authorized clients and give them the server address:

$$x, u : Mil \vdash Au(x,u) \xrightarrow{(x,auth_i,\langle u\rangle),\mathrm{id}} x, u : Mil \vdash Au(x,u) \tag{6.3}$$

where i ranges over the indexes of valid clients.

Finally, the server simply accepts requests from clients:

$$u : Mil \vdash S(u) \xrightarrow{(u,req,\langle\rangle),\mathrm{id}} u : Mil \vdash S'(u)$$

where S' is used to model the server when it is busy. It becomes available again via the production:

$$u : Mil \vdash S'(u) \xrightarrow{(u,\epsilon,\langle\rangle),\mathrm{id}} u : Mil \vdash S(u)$$

which does not require any synchronization since it corresponds to an internal transition of the server.

Until now, just Milner synchronization has been used. We want to extend the example by making each request to be served by many servers at once. For instance, assume that the request is a search query on the web that clients want to submit to different search engines. This extension can be easily managed by using multicast synchronization between the clients and the servers. This can be obtained by modifying production (6.1) as follows:

$$x : Mil \vdash C_i(x) \xrightarrow{(x,\overline{auth_i},\langle y\rangle),\mathrm{id}} x : Mil, y : Mul \vdash C'_i(y) \tag{6.4}$$

where Mul is the multicast SAM on the action req, and by defining $Mil \diamond Mul = Mul$. In this way rule (merge-HS) ensures that the synchronization of productions 6.4 and 6.3 yields the expected result, namely, the attachment node of servers uses a multicast SAM, since C_i requests it to do so. Notice also that now only request actions can be executed on node u. To allow components to synchronize on u after the merge, the productions for the servers, the clients and the authority must be extended to allow on u also multicast labels, instead of just Mil. This can be simply specified using a production schema where u is labeled by a type containing all the desired interaction protocols. For instance, the universal type $\mathcal{Alg}$ can be used, but also a more constrained one if some protocols are considered harmful and they must be avoided. For instance, Production 6.2 for service request becomes:

$$y : \mathcal{Alg} \vdash C'_i(y) \xrightarrow{(y,\overline{req},\langle\rangle),\mathrm{id}} y : \mathcal{Alg} \vdash C'_i(y)$$

Notice that in all the instances the two types must be instantiated with the same SAM to preserve the label of node y.

Finally, if some clients want to send the request to just one server, while others want to send it in multicast, they can use different actions for requests (that can be answered however by the same server action, thus this update is transparent to the servers, provided that their production schemas allow the used SAMs), and they can update the SAM on node u in such a way that only the behaviour of their request action is changed. Thus each client can choose the required kind of service at runtime, and the same servers are able to fulfil all the requests.

We present now an example that fully exploits the power of determining at runtime the SAM used on a node as a result of a negotiation among different components.

Example 6.4. *We consider a scenario where multiple processes/hosts are connected through channels with different features. Mainly, we consider two characteristics: the maximum packet size, which can be either* 4kb *or* 16kb*, and the presence/lack of some error-detecting mechanism, e.g., the CRC algorithm. Before starting the real communication, two processes create a logic communication channel between them. This channel supports* 16kb *packets and/or error-detecting capabilities only if all the underlying channels do so.*

We model this scenario using eight SAMs obtained by mixing three distinct features, namely:

- *packet sizes (*4kb/16kb*),*
- *error-detecting mechanism (yes/no),*
- *control or communication channel.*

The first two items correspond to the aforementioned characteristics of channels, while the last one specifies whether the channel is used for exchanging control messages or for data. For simplicity we suppose that control messages are short and thus they can be encoded using some error-correcting code. Hence, control messages are always correctly delivered. Since all the combinations of the three features are possible, we conventionally name the SAMs by means of expressions like CTR_4 *or* $COM^{\surd}_{16}$ *that respectively denote a control channel without error-detecting mechanism and with* 4kb *maximum packet size and a communication channel with error-detecting mechanism and* 16kb *maximum packet size.*

The control SAMs provide essentially Milner synchronization (where we drop the distinction between action and coaction) for control messages. The SAMs for communication without error-detecting mechanism provide faulty Milner synchronization, i.e., the result of a synchronization may be either τ *or* err*. In the former case name fusion is performed while no name fusion is performed in the latter. Note that, with these SAMs, processes can not detect whether the result of a synchronization is* τ *or* err *(unless they perform additional interactions). Communication actions can be of two kinds,* in_4 *for packets of* 4kb *and* in_{16} *for packets of* 16kb*. Corresponding output actions are provided. The same* τ *and* err *actions are used in both the cases. Both the kinds of synchronization are allowed by SAMs* 16kb*, while just the* 4kb*-size packets are allowed by SAMs* 4kb*. Finally, error-detecting SAMs allow two different kinds of input,* $in^{\surd}_s$ *which synchronizes giving* τ *and* in_s *which synchronizes giving a detected error, where* $s \in \{4\text{kb}, 16\text{kb}\}$*. Thus the action performed (and the different production used) allow the process to know whether an error has occurred or not. The nondeterministic behaviour of the channel, which may cause the error to happen or not, is mimicked by the nondeterministic choice of the production.*

To clarify the above description we formally define here the SAM $COM_{16}^{\surd}$*, where for simplicity we suppose that input and output actions carry exactly one parameter:*

- $Id = COM_{16}^{\surd}$*;*
- $Act = \{in_4^{\surd}, in_4, out_4, in_{16}^{\surd}, in_{16}, out_{16}, \tau, err, \epsilon\}$*;*
- $\mathrm{ar}(\lambda) = 0$ *for* $\lambda \in \{\tau, err, \epsilon\}$*,* $\mathrm{ar}(\lambda) = 1$ *otherwise;*
- $Init = \{\epsilon\}$*;*
- $Fin = \{\tau, err, \epsilon\}$*;*
- $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0)) \in ActSyn$ *for each* $\lambda \in Act$*,*
  $(in_4^{\surd}, out_4, (\tau, MP_{0,0}, EQ_1)) \in ActSyn$*,*
  $(in_4, out_4, (err, MP_{0,0}, EQ_0)) \in ActSyn$*,*
  $(in_{16}^{\surd}, out_{16}, (\tau, MP_{0,0}, EQ_1)) \in ActSyn$*,*
  $(in_{16}, out_{16}, (err, MP_{0,0}, EQ_0)) \in ActSyn$*.*

We can now define the operator $\diamond$ *of SAM composition. We define a partial order on the set of SAMs, and* $\diamond$ *will be the greatest lower bound according to that partial order. The order is defined component-wise on the three features of SAMs, over which the following orders are considered:* 16kb $>$ 4kb*, the presence of an error-detecting mechanism is greater than its absence and* $CTR > COM$*. Note that the set of SAMs with the resulting composition operator forms a commutative monoid as required, with* $CTR_{16}^{\surd}$ *as neutral element.*

We consider as types all the singletons, the universal type $\mathcal{Alg}$ *and the type* CT *that contains the four control SAMs.*

For simplicity we consider that the system contains two kinds of subsystems: end systems $x : S \vdash Idle(x)$ *and routers* $x : S_1, y : S_2, z : S_3 \vdash R(x, y, z)$*. End systems can start a communication with the production schema:*

$$x : CT \vdash Idle(x) \xrightarrow{(x, connect, \langle y \rangle), \mathrm{id}} x : CT, y : \mathrm{COM}_{16}^{\surd} \vdash Active(x, y) \tag{6.5}$$

After connecting to another system, an Idle process becomes Active, i.e. able to send/receive data over the channel built toward other systems.

We now show the productions for routers. We write just a meta-rule, with type meta-variables t_1 *and* t_2 *to be instantiated with each singleton containing a control SAM. Furthermore a similar meta-rule is necessary for each permutation of attachment nodes.*

$$\begin{aligned} x : t_1, y : t_2, z : CT \vdash R(x, y, z) &\xrightarrow{(x, connect, \langle w \rangle), (y, connect, \langle w \rangle), \mathrm{id}} \\ &\to x : t_1, y : t_2, z : CT, w : t_1 \diamond t_2 \vdash R(x, y, z) \end{aligned} \tag{6.6}$$

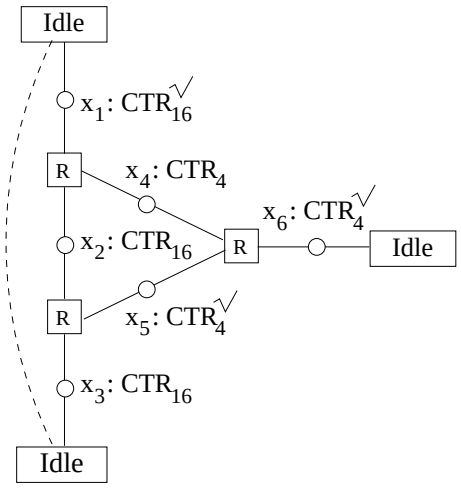

Figure 6.4: Sample communication network.

We can now show some sample transitions. Let us consider as starting graph:

$$\emptyset \vdash \nu x_1 : CTR_{16}^{\surd}, x_2, x_3 : CTR_{16}, x_4 : CTR_4, x_5, x_6 : CTR_4^{\surd}$$
$$Idle(x_1)|R(x_1, x_2, x_4)|R(x_2, x_3, x_5)|Idle(x_3)|R(x_4, x_5, x_6)|Idle(x_6)$$

which is graphically represented in Figure 6.4.

For instance, we can build a communication channel between the two end systems $Idle(x_1)$ *and* $Idle(x_3)$ *via routers* $R(x_1, x_2, x_4)$ *and* $R(x_2, x_3, x_5)$ *(along the dashed line in Figure 6.4) using productions (6.5) and (6.6). The SAM for the resulting connection is* COM_{16}*, obtained as a result of* $COM_{16}^{\surd} \diamond CTR_{16} \diamond CTR_{16} \diamond COM_{16}^{\surd}$*, where the first and the fourth SAMs are from the end systems, and the two in the middle from routers. We obtain the graph:*

$$\emptyset \vdash \nu x_1 : CTR_{16}^{\surd}, x_2, x_3 : CTR_{16}, x_4 : CTR_4, x_5, x_6 : CTR_4^{\surd}, y : COM_{16}$$
$$Active(x_1, y)|R(x_1, x_2, x_4)|R(x_2, x_3, x_5)|Active(x_3, y)|R(x_4, x_5, x_6)|Idle(x_6)$$

which is graphically represented in Figure 6.5. Similarly we can build a connection between $Idle(x_1)$ *and* $Idle(x_6)$ *via* $R(x_1, x_2, x_4)$ *and* $R(x_4, x_5, x_6)$*, but this time the resulting channel uses the SAM* COM_4*. When either of these connections has been established, communication can take place using it. Notice that the label of the channel is computed from the constraints imposed by the interacting components.*

We continue to analyze SAMs in next chapter, but we move to a new application field, namely process calculi.

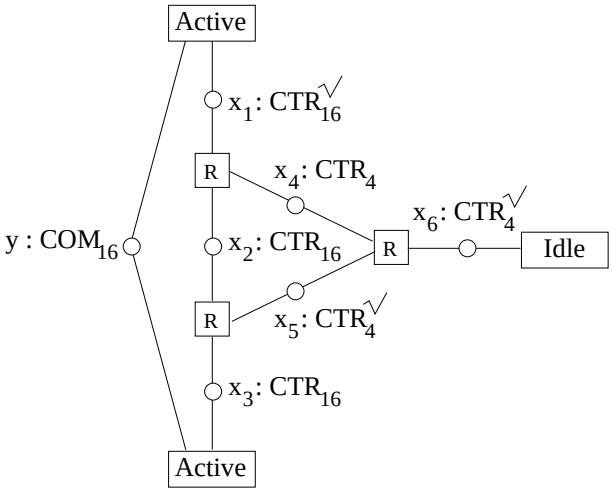

Figure 6.5: Communication network with a new logic channel.

Chapter 7

Applications of SAMs to process calculi

In this chapter we present an application of SAMs in the field of process calculi. The main motivation for this chapter is that SHR provides a framework that is powerful but has strict constraints on the modeling strategies, since different aspects are dealt with using different formal tools (i.e., the graph to describe the topology of the system, productions for its behaviour). Process calculi instead provide some basic operators (parallel composition, restriction, nondeterministic choice) that allow to easily model basic facts about system structure and behaviour, and that can be used to build models more and more complex, keeping into account each time the particular features of interest (e.g., probability, pattern matching, time, ...) with suitable extensions. In fact, the literature on process calculi is rich of such extensions, and many of them can be applied with little effort to different calculi. The same approach is not possible in the SHR framework, also because multiple concurrent synchronizations, as can be found in SHR, make these extensions harder.

In particular, we choose Fusion Calculus as basic process calculus to be extended with SAMs. The first reason is that its simplicity, featuring however the characteristics we are interested in, and in particular mobility and handling of local resources, makes the extension easier than for other more complex calculi. We will however give some suggestions on how to apply the approach to other calculi, and in particular to the π-calculus.

Another important reason for the choice of Fusion Calculus is that the link between Fusion Calculus and MSHR presented in Chapter 2 allows an easier comparison between the results in this chapter and the ones on parametric SHR presented in the previous one. It is true that an extension of Fusion Calculus similar to the one presented here can be obtained by using the mapping in Chapter 2 and animating then the result using parametric SHR instead of MSHR, but, apart from the fact that the syntax of Fusion Calculus must be extended anyway to allow different actions, we think that it is interesting to present a stand-alone semantics in process calculi style in order to allow the application of techniques from process calculi liter-

ature as said above. Also, in this way some issues that would be otherwise hidden in the mapping are more clearly visible, and this allows to get a deeper understanding of aspects such as the scope of a synchronization or the extrusion of a name.

7.1 The PRISMA Calculus

In this section we describe PRISMA Calculus, which is technically an extension of Fusion Calculus with SAMs specifying the used synchronization policy. For simplicity, we consider just one SAM at the time as done in parametric SHR, however the approach can be generalized exploiting the ideas presented for SHR-HS. The name PRISMA, the Italian for "prism", is intended to expose the many different faces it could be looked at.

We present now the syntax of PRISMA Calculus.

Definition 7.1 (Syntax of PRISMA Calculus). *PRISMA processes are parametric on a SAM S. Their syntax is the following one:*

$$\begin{array}{rcll} P ::= & 0 & \textit{(Inaction)} & x a\vec{y}.P \quad \textit{(Prefix)} \\ & P|P & \textit{(Parallel composition)} & P+P \quad \textit{(Nondeterministic sum)} \\ & (x)P & \textit{(Restriction)} & !P \quad \textit{(Replication)} \end{array}$$

where x is a name, $a \in \mathrm{fAct}(S)$ is an action and $\vec{y}$ is a vector of names such that $\mathrm{ar}(a) = |\vec{y}|$.

In PRISMA, (x) is the only binder for x. At the syntax level, the only difference between PRISMA and Fusion is that in PRISMA prefixes are of the form $xa\vec{y}$. Such a prefix, which has the same structure of an SHR synchronization, requires the execution of action a on channel x with $\vec{y}$ as tuple of parameters.

We have chosen to introduce in the calculus only the basic primitives necessary to study interaction, however new operators can be added without particular effort as usual in the process calculi field. Also, the choice of using replication instead of recursion has been done only to simplify the proof of the congruence theorem presented later (Theorem 9.4), but recursion can be included without problems as well.

The intrinsic compositionality of action synchronization in SAMs makes the LTS operational semantics more natural for PRISMA than semantics in the reduction style. In particular, reduction rules specifying the interaction of an unbound number of components are hard to specify, while using the LTS approach just two components at the time can be considered.

The main problem in defining the semantics of PRISMA is how to decide which processes must participate to a given interaction. This problem can be divided into two subproblems:

1. which processes model single components of the system;

2. which components must participate to a given interaction.

Note that in the SHR framework both the problems have a trivial solution, thanks to the explicit description of the topology of the system: a component is an edge of the graph, and it must participate to an interaction iff it is attached to the node on which the interaction is performed.

For the first problem the obvious solution in the process calculi setting is to use as components the sequential processes, that is processes with a prefix or a nondeterministic choice as topmost operator. This choice corresponds to consider parallel processes as sets of interacting components, and it follows the analogy between SHR and Fusion Calculus described in Chapter 2. With this interpretation, SAMs provide first of all a description of the behaviour of the parallel composition operator.

The second problem can be solved following different approaches. As will become clear later, this problem is crucial when the used SAM requires all the processes able to interact to actually provide a non trivial contribution to the synchronization, such as done by Hoare or broadcast SAM. Note that in this case the approach of Fusion Calculus does not give a useful suggestion, since Fusion is based on Milner synchronization, thus the problem is less relevant.

We think that the most interesting alternatives are the following ones:

- all the components;
- all the components that are aware of the existence of the channel x where the synchronization is performed (i.e., the sequential processes whose set of names includes the channel x);
- all the components that are able to communicate on channel x when the synchronization is performed (i.e., the sequential processes that have a prefix with x as subject at the top level).

We think that the first choice, even if technically simpler, is not realistic. In fact, when for instance Hoare synchronization is used, this requires all the sequential processes in the system to interact, but in general many of them are not interested in this synchronization, thus a model based on this choice will have great scalability problems.

The second choice follows the analogy with SHR, since nodes attached to an edge correspond to names of the corresponding sequential process.

We choose the third approach for two reasons:

- we want to explore a different alternative w.r.t. the one obtained via the mapping into SHR;
- it is easier to simulate the second approach inside the third than to do the vice versa.

We define now the operational semantics of PRISMA.

$$\frac{-}{0 \xrightarrow{\neg x} 0} \qquad \frac{x \neq z}{xa\vec{y}.P \xrightarrow{\neg z} xa\vec{y}.P} \qquad \frac{P \xrightarrow{\neg x} P \quad x \neq y}{(y)P \xrightarrow{\neg x} (y)P} \qquad \frac{-}{(x)P \xrightarrow{\neg x} (x)P}$$

$$\frac{P_1 \xrightarrow{\neg x} P_1 \qquad P_2 \xrightarrow{\neg x} P_2}{P_1|P_2 \xrightarrow{\neg x} P_1|P_2} \qquad \frac{P_1 \xrightarrow{\neg x} P_1 \qquad P_2 \xrightarrow{\neg x} P_2}{P_1 + P_2 \xrightarrow{\neg x} P_1 + P_2} \qquad \frac{P \xrightarrow{\neg x} P}{!P \xrightarrow{\neg x} !P}$$

Table 7.1: Operational semantics of PRISMA: transitions with label $\neg x$.

Notation. *We denote with λ a general label.*

Definition 7.2 (Operational semantics for PRISMA Calculus). *The operational semantics for PRISMA Calculus is the least LTS generated from the inference rules in Table 7.1 and Table 7.2. We also need the symmetric versions of rules 7.4, 7.5, 7.10 and 7.11.*

Note that rule 7.3 instead is self-symmetric.

As already said, we have designed PRISMA to force synchronization among processes that are willing to perform an action on the same channel x, i.e., which have an active prefix with x as subject. In order to have a compositional semantics (see Theorem 9.4 in Section 9.2), the set of channels on which a process can not synchronize must be observable. Thus, following the approach of [EM99], we introduce a label $\neg x$ for each $x \in \mathcal{N}$: a transition with label $\neg x$ can be executed by any process P which has no active prefix with subject x (see Table 7.1), and it leads to P itself. To be precise, [EM99] deals only with broadcast synchronization, and there input prefixes and output prefixes are treated differently: only input prefixes are considered active, since in broadcast all the interacting processes but one have to perform an input. We can not follow a similar approach since we have to deal with general synchronization policies, and, in general, actions may be neither inputs nor outputs (consider, e.g., the case of Hoare synchronization).

In addition to labels $\neg x$, two further kinds of labels are used in Table 7.2: the first one has the form $(Y)xa\vec{y}, \pi$ and it models prefix execution, but it adds to prefix $xa\vec{y}$ two additional pieces of information: a set of extruded names Y and a renaming π. First of all, when a bound name y is communicated outside its scope, we want to trace that its scope is growing, and thus we include y in Y, since, when y is removed from the tuple of communicated names (e.g., because of some synchronization), we want to reintroduce the restriction for y. The same technique is used, e.g., in π-calculus, where rule (close) adds the required restriction operator. Note that Y is a subset of $\mathrm{n}(\vec{y})$ and it does not contain x. Furthermore, fusions performed on global names must be traced by π, since they must be applied to parallel processes. This is performed by rule 7.4.

The second kind of label has the form $(\surd, \pi)$ and it is used to further propagate renaming π (which is the same as before) when the channel where the synchronization has been performed is restricted. Renaming π is applied to parallel processes

$$\frac{-}{xa\vec{y}.P \xrightarrow{xa\vec{y},\mathrm{id}} P} \quad (7.1) \qquad \frac{x \in \mathcal{N}}{P \xrightarrow{x\epsilon\langle\rangle,\mathrm{id}} P} \quad (7.2)$$

$$\frac{P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1,\pi_1} P_1' \qquad P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2,\pi_2} P_2' \qquad \Phi}{P_1|P_2 \xrightarrow{(W)xc\vec{w},\pi} (\vec{s})(P_1'|P_2')\rho} \quad (7.3)$$

$$\frac{P_1 \xrightarrow{(Y)xa\vec{y},\pi} P_1' \qquad P_2 \xrightarrow{\neg x} P_2}{P_1|P_2 \xrightarrow{(Y)xa\vec{y},\pi} P_1'|P_2\pi} \quad (7.4) \qquad \frac{P_1 \xrightarrow{\surd,\pi} P_1'}{P_1|P_2 \xrightarrow{\surd,\pi} P_1'|P_2\pi} \quad (7.5)$$

$$\frac{P \xrightarrow{(Z)xa\vec{y},\pi} P' \quad a \in Fin \quad x \notin \mathrm{Im}(\pi) \quad \vec{z} = \mathrm{vect}(Z)}{(x)P \xrightarrow{\surd,\pi|_{\setminus\{x\}}} (x)(\vec{z})P'} \quad (7.6)$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi} P' \quad z \in \mathrm{n}(\vec{y}) \setminus \{x\} \setminus Y \quad z \notin \mathrm{Im}(\pi) \quad z' \notin (\mathrm{fn}(P') \cup \{x\} \cup \mathrm{n}(\vec{y})) \setminus \{z\}}{(z)P \xrightarrow{(\{z'\}\cup Y)xa\vec{y}\{z'/z\},\pi|_{\setminus\{z\}}} P'\{z'/z\}} \quad (7.7)$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi} P' \quad z \notin \mathrm{n}(\vec{y}) \cup \{x\} \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{(Y)xa\vec{y},\pi|_{\setminus\{z\}}} (z)P'} \quad (7.8) \qquad \frac{P \xrightarrow{\surd,\pi} P' \quad x \notin \mathrm{Im}(\pi)}{(x)P \xrightarrow{\surd,\pi|_{\setminus\{x\}}} (x)P'} \quad (7.9)$$

$$\frac{P_1 \xrightarrow{\lambda} P_1' \quad \lambda \neq x\epsilon\langle\rangle, \mathrm{id} \quad \lambda \neq \neg x}{P_1 + P_2 \xrightarrow{\lambda} P_1'} \quad (7.10) \qquad \frac{P|!P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P'} \quad (7.11)$$

Φ in the premise of rule 7.3 is the conjunction of:

1. $Y_1 \cap Y_2 = \emptyset$, $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$;
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in ActSyn$;
3. $\rho = \mathrm{mgu}(\{\vec{y}_{i_1}[j_1] = \vec{y}_{i_2}[j_2] | \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\} \cup \{t = u | t\pi_1 = u\pi_1 \vee t\pi_2 = u\pi_2\})$ where ρ maps names to representatives not in $Y_1 \cup Y_2$ whenever possible;
4. $\pi = \rho|_{\setminus(Y_1\cup Y_2)}$;
5. $\vec{w}[i] = (\vec{y}_j[k])\sigma$ iff $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$;
6. $W = \mathrm{n}(\vec{w}) \cap (Y_1 \cup Y_2)$;
7. $\vec{s} = \mathrm{vect}((Y_1 \cup Y_2) \setminus W)$.

Table 7.2: Operational semantics of PRISMA: transitions with executable actions.

by rule 7.5. Note that no further prefix can interact with $(\surd, \pi)$, and this ensures interleaving among synchronizations: at each step at most one synchronization can be performed. A concurrent semantics in the style of Section 2.3 is also possible, but it is more complex, thus we stick to the interleaving one.

Rules 7.1, 7.5 and 7.11 are straightforward. Rule 7.2 says that each process

can perform action ϵ on any channel. This corresponds to the inclusion of idle productions in all sets of SHR productions. This explains why the exact definition of which processes have to synchronize is not important when the required contribution to the synchronization is just ϵ.

The most complex rule is the rule for synchronization (rule 7.3), which allows two actions a_1 and a_2 performed on the same channel x to synchronize according to some action synchronization $(a_1, a_2, (c, \mathrm{Mob}, \doteq))$ (condition 2). The main effect of the synchronization is to produce a new renaming ρ (condition 3), which combines the previous renamings traced by π_1 and π_2 with the new renaming determined by the equivalence classes defined by $\doteq$. The renaming ρ is applied to the two interacting processes P_1' and P_2' and to the tuple of parameters $\vec{w}$ (computed according to Mob, as specified by condition 5) of the resulting action c, and, as far as free names are concerned, it is traced in the label by new renaming π (condition 4). W is the new set of extruded names (condition 6). Finally, names that were extruded $(Y_1 \cup Y_2)$ and are no longer in the label $((Y_1 \cup Y_2) \setminus W)$ must be bound by inserting them into $\vec{s}$ (in any order), as specified by condition 7. Rule 7.4 allows to add in parallel another process, provided that it can not perform any action on the channel that is the subject of the synchronization in the label.

The rules for restriction deserve some explanations too. Rule 7.6 binds the channel on which the synchronization in the label is performed. This is allowed iff the performed action a belongs to Fin. All the extruded names must be bound in the resulting process. If the equivalence class of the hidden name x according to π is not a singleton, we have to remove x from π, since x is not visible outside its scope. To do that we require that x is not a representative (it is always possible to find a transition where this is the case, since there is a transition for each choice of the representatives among the free names in their equivalence class), and we delete its binding. Rule 7.7 corresponds to the rule (open) of π-calculus: if the restricted name appears in the tuple of parameters of the action, then it is marked as extruded. Note that the extruded name z can be α-converted to a new name z', but z itself can be used too, since we may have $z' = z$. The choices that allow to build a full derivation are the ones where the chosen name z' does not occur elsewhere in the process. Rule 7.8 says that restricting channel z does not influence actions performed on a different channel x. Similarly, rule 7.9 guarantees that restriction has no influence on labels of the form $(\surd, \pi)$, apart from removing the bound name from π.

It is interesting to note in rule 7.10 that actions of the form $x\epsilon\langle\rangle$, id or $\neg x$ do not force any of the two alternatives of a nondeterministic choice operator to be chosen. This has been decided since these actions do not represent actual activities from processes.

Note that the above semantics does not require any axiom of structural congruence, and this simplifies the proofs of the properties of the calculus.

In the above semantics, action ϵ can be freely performed on any node. This is necessary since components may not participate, e.g., to a Milner synchronization. However, a restriction operator can transform this action into $(\surd, \mathrm{id})$ if $\epsilon \in Fin$.

This may be confused with a true internal synchronization, thus this is not a desired behaviour. However, while in a concurrent setting ϵ can be the final action on many channels, in an interleaving setting this is possible only if the whole process stays idle, and this is not an interesting case. Thus we remove ϵ from Fin when using SAMs in the PRISMA setting.

We present here some examples to clarify the above semantics and to show some applications of PRISMA Calculus. We start with a simple one.

Example 7.1. *The allowed transitions for the PRISMA process:*

$$P = (x)(x\ in\langle y\rangle.P_1 \mid x(out,3)\langle z\rangle.P_2 \mid x(out,2)\langle w\rangle.P_3)$$

which exploits the priority SAM Pri of Definition 5.11 are:

$$\begin{aligned} P &\xrightarrow{\checkmark,\{z/y\}} (x)(P_1 \mid P_2 \mid x(out,2)\langle w\rangle.P_3)\{z/y\} \\ P &\xrightarrow{\checkmark,\{w/y\}} (x)(P_1 \mid x(out,3)\langle z\rangle.P_2 \mid P_3)\{w/y\} \\ P &\xrightarrow{\checkmark,\{z/y\}} (x)(P_1 \mid P_2 \mid P_3)\{z/y\} \end{aligned}$$

In the first two transitions there is simply an interaction between an input and an output, in Fusion style. The most interesting transition is the third one, where three different sequential processes interact. Only the output $(out,3)\langle z\rangle$, which has the highest priority, is received, thus z and y are merged. The other output is discarded.

Similar transitions where y is chosen as representative instead of z or w, and transitions to P itself with labels of the form $u\epsilon\langle\rangle, \mathrm{id}$ or $\neg u$ for any u are allowed too.

Note that a full message exchange has to be performed since x is bound, thus no other transitions are allowed.

We present now a simple application to publish-subscribe synchronization.

Example 7.2 (Publish-subscribe synchronization). *Let us consider a server:*

$$S = !(y)(!x\ publish\ \langle y\rangle.0 \mid !y\ news_{|\vec{z}|}\ \vec{z}.0)$$

which can:

1. *publish the name of a channel for news using an action x publish $\langle y\rangle$ or,*
2. *send news on a channel using an action y $news_{|\vec{z}|}$ $\vec{z}$ (the number of parameters may depend on the news).*

A client C has the form:

$$C = x\ subscribe\ \langle w\rangle.!w\ get_{|\vec{t}|}\ \vec{t}.0$$

If publish is an output of a multicast SAM, with subscribe as corresponding input, then any number of processes can subscribe at the same time. Also, more complex actions can be used to embed in the synchronization some choices on whether to subscribe or not. For instance, we can have actions publish − CS for publishing computer science news and publish − math for mathematic news. Suitable actions can be defined to subscribe to both the kinds of news or to choose the wanted one. The operation of sending a news can be defined as a broadcast in order to ensure that any process ready to receive the news will actually receive it. Notice however that busy processes (which have no active input on the channel where the news is broadcasted) do not spoil the delivery mechanism.

We exploit here the categorical structure of SAMs to define a translation between PRISMA processes defined on different SAMs.

Definition 7.3. *Given a SAM morphism* $H : S_1 \to S_2 = (h, \{h_a | a \in \mathrm{fAct}(S_1))$*, the corresponding translation from PRISMA processes on SAM* S_1 *into PRISMA processes on SAM* S_2 *is defined as the homomorphic extension of the translation of prefixes mapping* $xa\vec{y}$ *to* $x\, h(a)\, \vec{w}$ *where* $\forall i \in \underline{\mathrm{ar}(h(a))}.\vec{w}[i] = \vec{y}[h_a(i)]$.

The above translation intuitively preserves part of the behaviour of processes. However, transitions are not preserved, since, e.g., fewer merges can be performed in the translated process. As an example, consider simply the morphism that deletes all the parameters of actions. This kind of morphism is necessary to compose SAMs: projection morphisms in the product construction may be of this kind. Preservation properties can be stated for smaller classes of morphisms. We consider here the case of isomorphisms.

We need to extend the translation also to labels.

Definition 7.4 (Translation of labels along a morphism). *Given a SAM morphism* $H : S_1 \to S_2 = (h, \{h_a | a \in \mathrm{fAct}(S_1))$*, the corresponding translation from PRISMA labels on SAM* S_1 *into PRISMA labels on SAM* S_2 *maps* $(Y)xa\vec{y}, \pi$ *to* $(W)xh(a)\vec{w}, \pi$ *where* $\forall i \in \underline{\mathrm{ar}(h(a))}.\vec{w}[i] = \vec{y}[h_a(i)]$ *and* $\vec{w}[i] \in W$ *iff* $\vec{y}[h_a(i)] \in Y$*. The translation is the identity on labels of the form* $\neg x$ *and* $(\surd, \pi)$.

We can now state the proposition.

Proposition 7.1. *Given a SAM isomorphism* $H : S_1 \to S_2$*,* $P \xrightarrow{\alpha} P'$ *can be derived using SAM* S_1 *iff* $H(P) \xrightarrow{H(\alpha)} H(P')$ *can be derived using SAM* S_2.

Proof. The proof is by induction on the derivation of the transition. □

The effects of the translation on the observable behaviour of processes are described in Section 9.2.

We present now a further example on how to introduce accounting on a channel.

Example 7.3 (Introducing accounting on a channel). *Let us consider the SAM account defined as follows:*

- $Act = \{c, \epsilon\}$;
- $\mathrm{ar}(c) = 0$;
- $Init = \{\epsilon\}$;
- $Fin = \{c\}$;
- $(\lambda, \epsilon, (\lambda, MP_{0,0}, EQ_0)) \in ActSyn$ *for any* $\lambda \in Act$.

The asynchronous product of account with any SAM S allows a controller process P_c to count the number of synchronizations performed on a channel. Let P be any process and x be the channel under accounting. Let H be the inclusion morphism from S to the product synchronization that maps a to (a, ϵ). Suppose that S contains an action $\$$ of arity 0.

Then $(x)H(P)|!x(\epsilon, c)\langle\rangle.y(\$, \epsilon)\langle\rangle.0$ with the product synchronization behaves as $(x)P$ (up to translation of actions) with SAM S, but it sends a message $(\$, \epsilon)$ on channel y for each synchronization performed on x. In fact, to get a final action, any action on x must synchronize with $x(\epsilon, c)\langle\rangle$, thus releasing a message $y(\$, \epsilon)\langle\rangle$.

Not accounted actions can be added by using a coproduct construction.

7.2 Relations between PRISMA and other process calculi

In this section we analyze the relationships between PRISMA Calculus and other process calculi from two different points of view. First of all we consider the possibility of mapping other calculi into PRISMA. In particular, we analyze the relation between Fusion Calculus and PRISMA, which is, as expected, very strict. Then we consider the possibility of applying the PRISMA approach based on SAMs using as starting point another calculus instead of Fusion Calculus. We analyze in particular the case of π-calculus.

As first result we show that PRISMA over $Milner_{IN}$, where $IN = \{in_n | n \in \mathbb{N}\}$ with $\mathrm{ar}(in_n) = n$, is essentially Fusion Calculus. The operational semantics that we get for Fusion processes via the mapping is half-way between the standard one (see Definition 1.13) and the concurrent semantics described in Section 2.3 (it is however an interleaving semantics).

We consider the subset of Fusion Calculus whose processes are defined by:

$$P ::= 0 \mid u\vec{x}.P \mid \overline{u}\vec{x}.P \mid P|P \mid P + P \mid (x)P \mid !P$$

i.e. containing the inactive process, communication prefixes, parallel composition, summation, hiding and replication (instead of recursion). We do not allow fusion prefixes, but $\{\vec{x} = \vec{y}\}.P$ can be encoded as $(z)(z\vec{x}.P|\overline{z}\vec{y}.0)$ for $z \notin \mathrm{fn}(P)$. The

mapping can be extended to include other operators by adding the corresponding ones to PRISMA Calculus.

We present now the definition of the translation.

Definition 7.5 (Translation from Fusion into PRISMA).
The uniform encoding $[\![-]\!]^F_{PRI}$ from Fusion processes into PRISMA processes is the homomorphic extension to the whole calculus of:

- $[\![u\vec{x}.P]\!]^F_{PRI} = u\ in_{|\vec{x}|}\vec{x}.[\![P]\!]^F_{PRI}$,
- $[\![\overline{u}\vec{x}.P]\!]^F_{PRI} = u\ out_{|\vec{x}|}\vec{x}.[\![P]\!]^F_{PRI}$,

where in_n and $out_n = \overline{in_n}$ are complementary actions in the SAM $Milner_{IN}$.

The mapping can be extended to communication labels as follows.

Definition 7.6 (Translation of communication labels).
The translation $[\![-]\!]^F_{PRI}$ is defined on Fusion communication labels by:

- $[\![(\vec{y})u\vec{x}]\!]^F_{PRI} = (\mathrm{n}(\vec{y}))u\ in_{|\vec{x}|}\vec{x}, \mathrm{id}$
- $[\![(\vec{y})\overline{u}\vec{x}]\!]^F_{PRI} = (\mathrm{n}(\vec{y}))u\ out_{|\vec{x}|}\vec{x}, \mathrm{id}$

Note that the above translation loses the order of extruded names, but this is unimportant, since in Fusion all different orderings can be obtained thanks to structural congruence.

The following theorem formalizes the correctness and the completeness of the translation.

Theorem 7.1 (Correctness and completeness).
Let P be a Fusion process. $P \xrightarrow{\alpha} P'$ iff:

1. *α is a communication action, $[\![P]\!]^F_{PRI} \xrightarrow{[\![\alpha]\!]^F_{PRI}} [\![P_1]\!]^F_{PRI}$ and $P_1 \equiv P'$ or;*
2. *α is a fusion action, $[\![P]\!]^F_{PRI} \xrightarrow{\lambda} [\![P_1\pi]\!]^F_{PRI}$ and $P_1\pi \equiv P'\pi$ where λ is either $(\surd, \pi)$ or $(x\tau\langle\rangle, \pi)$ for some $x \in \mathrm{fn}([\![P]\!]^F_{PRI})$ and where π is a mgu of α.*

Proof. The proof is by structural induction on Fusion processes. We have a case for each operator. We have to prove that the Fusion process P has a transition iff its translation $[\![P]\!]^F_{PRI}$ has a transition of the wanted form. As far as the only if part is concerned, note that translations of Fusion processes have just four kinds of transitions: the two described by the theorem, and $x\epsilon\langle\rangle, \mathrm{id}$ or $\neg x$ transitions to themselves. Thus completeness is proved by checking for each case that transitions of the first two kinds are in correspondence with Fusion ones. Note that, when referring to rule n, we mean rule n in Table 7.2.

Case 0) We have no rules for 0 in Fusion Calculus, and we can have only either $x\epsilon\langle\rangle, \mathrm{id}$ or $\neg x$ transitions in PRISMA. This proves the thesis.

Case prefix) We have to show that if the thesis holds for P, then it also holds for $xa\vec{y}.P$. The two rules for prefix are equal up to the translation. The thesis follows trivially.

Case $|$) We have to show that if the thesis holds for P and for Q, then it also holds for $P|Q$. We have two applicable rules here. Let us consider rule (PAR) first. If α is a communication label, by hypothesis $[\![P]\!]^F_{PRI} \xrightarrow{[\![\alpha]\!]^F_{PRI}} [\![P_1]\!]^F_{PRI}$ and $P_1 \equiv P'$. Then we can use rule 7.2 to derive $[\![Q]\!]^F_{PRI} \xrightarrow{x\epsilon\langle\rangle,\mathrm{id}} [\![Q]\!]^F_{PRI}$ and then rule 7.3 using action synchronization $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0))$ to derive $[\![P|Q]\!]^F_{PRI} \xrightarrow{[\![\alpha]\!]^F_{PRI}} [\![P_1|Q]\!]^F_{PRI}$ since we have $\rho = \mathrm{id}$. The thesis follows because $P_1|Q \equiv P'|Q$. If α is a fusion action, its translation can be either of the form $x\tau\langle\rangle, \pi$ or of the form $(\surd, \pi)$. In the first case we have by hypothesis that $[\![P]\!]^F_{PRI} \xrightarrow{\lambda} [\![P_1\pi]\!]^F_{PRI}$, and we can use the same rules as before to deduce $[\![P|Q]\!]^F_{PRI} \xrightarrow{\lambda} [\![(P_1\pi|Q)\pi]\!]^F_{PRI} = [\![(P_1|Q)\pi]\!]^F_{PRI}$, since $\vec{s}$ is empty, and $\rho = \pi$ (up to a change of representatives). Last step is justified by idempotence of π. The thesis follows again. If the label has the form $(\surd, \pi)$, then the desired transition can be derived using rule 7.5.

Let us now consider rule (COM). By hypothesis we have two transitions $[\![P]\!]^F_{PRI} \xrightarrow{[\![u\vec{x}]\!]^F_{PRI}} [\![P_1]\!]^F_{PRI}$ and $[\![Q]\!]^F_{PRI} \xrightarrow{[\![\overline{u}\vec{y}]\!]^F_{PRI}} [\![Q_1]\!]^F_{PRI}$ with $P_1 \equiv P'$ and $Q_1 \equiv Q'$. Since the lengths of the two vectors $\vec{x}$ and $\vec{y}$ coincide, the two translations use complementary actions and thus they can interact using the action synchronization $(in_n, out_n, (\tau, MP_{0,0}, EQ_n))$ where n is the length of $\vec{x}$. Thus we can derive a transition of the form $[\![P|Q]\!]^F_{PRI} \xrightarrow{u\tau\langle\rangle,\pi} (P_1|Q_1)\pi$ as required. Notice in fact that we have no extruded names, thus $\vec{s}$ is empty and that $\pi_1 = \pi_2 = \mathrm{id}$, thus the new ρ is just given by the merge on parameters and it is a mgu of $\{\vec{x} = \vec{y}\}$. Furthermore $\pi = \rho$ since we have no bound names again. Notice that the Fusion rule (COM) deals only with actions with no extruded names, while rule 7.3 in the semantics of PRISMA is more general. However, the more general case can be simulated by Fusion, in fact extruded names can be avoided by moving restrictions to the outside using structural congruence. The allowed transitions are the same in both the cases. This will be proved formally later, when observational semantics is considered, and it is proved that the translations of the axioms of structural congruence bisimulate (see Lemma 8.8 and Lemma 8.9).

Case (x)) Let us consider restriction now. Here we have three rules. Rule (SCOPE) is simulated by rule 7.9 if the PRISMA action is of the form $(\surd, \pi)$, by rule 7.8 if it is of the form $x\tau\langle\rangle, \pi$ where x is not the bound name and by rule 7.6 if it is of the form $x\tau\langle\rangle, \pi$ and x is the bound name. In that case, the rule can be applied because $\tau \in Fin$, and we have $Z = \emptyset$ because $\mathrm{ar}(\tau) = 0$. Notice also that in all the cases the binding for x in π is deleted, and there is no

need to apply it to the process as done in Fusion, since it has already been applied. The condition that $x \notin \mathrm{Im}(\pi)$ is not restrictive since we have similar transitions for each choice of the representatives.

Rule (PASS) is an instance of rule 7.8 where $\pi = \mathrm{id}$ for communication actions and where the restricted name does not belong to $\mathrm{n}(\pi)$ for fusion actions.

Rule (OPEN) is an instance of rule 7.7 where $\pi = \mathrm{id}$. Note that in Fusion just x is used as extruded name since it can be changed using α-conversion and rule (STRUCT), while in PRISMA any allowed name is generated.

In order to have the other implication we have to check that PRISMA rules do not allow further transitions. PRISMA has four rules for restriction. Rule 7.8 is simulated by rule (PASS) when $z \notin \mathrm{n}(\pi)$ and by (SCOPE) otherwise (the restriction can be removed since z is no more used inside the process, furthermore the renaming is not necessary in PRISMA since a transition with a different name can be used as a premise). Rule 7.6 is simulated by rule (PASS) in Fusion Calculus, and it switches between the two different representations of fusion actions in PRISMA when a name where a τ action is performed is restricted. Rule 7.7 is simulated by rule (OPEN) (notice that here π is always the identity if the transition has been generated by the translation of a Fusion process). Finally, the same observations highlighted for rule 7.8 hold also for rule 7.9, when the PRISMA label has the form $(\surd, \pi)$.

Case ! and +) For each of the two operators, the rules in the two frameworks are equal up to the translation, thus the proof is trivial. Note just that in PRISMA we require that $x\epsilon\langle\rangle$, id transitions and $\neg x$ transitions can not force any nondeterministic choice to be taken.

Structural congruence) The first part of rule (STRUCT) is modeled since translations of equivalent Fusion agents have the same allowed transitions. This will be proved formally later, when observational semantics is considered, and it is proved that the translations of the axioms of structural congruence bisimulate (see Lemma 8.8 and Lemma 8.9). The second part is simulated since the theorem just requires to get as a result the translation of an equivalent process.

This concludes the proof. □

Note that in PRISMA each process can always perform idle steps to itself, with labels of the form $x\epsilon\langle\rangle$, id or $\neg x$, which have no Fusion correspondence.

We can now compare the semantics of Fusion obtained via the mapping with both the standard one from Section 1.4 and the concurrent one from Section 2.3. We consider the different issues analyzed in Section 2.3:

concurrent vs interleaving: the semantics presented here is interleaving, as the standard one.

located τ: τ here is located on a channel, as in the concurrent semantics. This is necessary since in the general framework of PRISMA τ is a normal action, and it must be located as other actions are. A semantics of the same kind can be obtained by modifying the standard Fusion Calculus semantics, adding in particular located fusions $x\phi$ to the set of labels. Furthermore, we have to change rule COM into the one below, and to add two new rules SCOPE' and SCOPE" to deal with the new labels:

$$\text{COM}\quad \frac{P \xrightarrow{u\vec{x}} P' \quad Q \xrightarrow{\overline{u}\vec{y}} Q' \quad |\vec{x}|=|\vec{y}|}{P|Q \xrightarrow{u\{\vec{x}=\vec{y}\}} P'|Q'}$$

$$\text{SCOPE'}\quad \frac{P \xrightarrow{u\phi} P' \quad z\phi x \quad z\neq x \quad z\neq u}{(z)P \xrightarrow{u\phi\setminus z} P'\{x/z\}}$$

$$\text{SCOPE"}\quad \frac{P \xrightarrow{z\phi} P' \quad z\phi x \quad z\neq x}{(z)P \xrightarrow{\phi\setminus z} P'\{x/z\}}$$

We think that, in general, the located semantics can be useful, since, for accounting or performance reasons, synchronizations performed on different channels may not be equivalent. Suppose for instance that channel x is provided by some company while channel y is local and owned by the user: a process performing a synchronization on y is cheaper than a process performing the same synchronization using channel x. Local channels are instead all equivalent, since any interaction, included traffic check (see Example 7.3), is required to happen inside the scope of the channel.

renamings vs fusions: here we follow the approach of the concurrent semantics of using renamings in the labels, but the two approaches are equivalent.

application of fusion renamings: here again we follow the approach of the concurrent semantics of directly applying the fusion renaming when it is computed, but however both the approaches can be followed.

free names: here, according to the decision taken at the beginning of chapter, we do not observe in the label the set of free names as in the concurrent semantics, but we observe the set of active names instead.

idle transitions: here we have two kinds of idle transitions, the ones with label $x\epsilon\langle\rangle, \mathrm{id}$ which correspond to the idle transitions in the concurrent semantics (here however a name is chosen as subject in the label, and it may not be a free name of the process), and the ones with label $\neg x$.

extrusions: here extrusions are visible in the label, as typical of process calculi semantics and as done in the standard semantics of Fusion Calculus.

other differences: there are other differences, for instance in the syntax of actions, but these differences have no real impact on the LTS since they are just due

to the different syntaxes used to carry the same information (the syntax of PRISMA must be more general to allow also other SAMs to be used).

We present now an example where different parts of the theory developed so far are combined to address interoperability issues, and in particular to execute a Fusion process in a priority scenario.

Example 7.4 (Executing Fusion processes in a priority scenario). *Let us consider an infrastructure built for priority communication as specified in Definition 5.11. We want to run a Fusion process P_F in that framework. Suppose, for the moment, that the Fusion process P_F has just prefixes with one parameter. We will show how P_F can be made to interact with the other processes (although, clearly, it will not be able to fully exploit the priority mechanism). First of all the translation from Fusion to PRISMA (see Definition 7.5) can be used to have a corresponding PRISMA process P_M on the SAM $Milner_{\{in_1\}}$. In the category $\mathcal{ASYNC}$ there is a morphism H_n from the SAM $Milner_{\{in_1\}}$ to the SAM for priority communication Pri for each priority n. H_n maps in_1 to in, out_1 to (out, n) and τ to $(out+, n)$. The corresponding translation of PRISMA processes (see Definition 7.3) allows to produce in an automated way a priority process $P_P = H_n(P_M)$. The process essentially has all the outputs at the fixed priority n, and it inputs the message with the highest priority as specified by priority synchronization. Notice that, if we want to have Fusion processes based on actions with different arities, we can just extend the priority SAM accordingly and then apply the same procedure.*

Until now we have analyzed the relationships between Fusion Calculus and PRISMA Calculus, and we have shown that they are very strict, as expected.

We give now some insights on the relationships with two other calculi: π-calculus and bπ-calculus [EM99]. These two calculi are quite similar, the main difference being that π-calculus is based on Milner synchronization while bπ-calculus is based on broadcast. Both the calculi use a slightly different approach to mobility w.r.t. Fusion Calculus, namely their input operator is a binder, thus communication is unidirectional (information flows from output to input). From a technical point of view, an important consequence of that fact is that two names restricted using the ν operator are guaranteed to remain distinct during the whole computation. This property is quite useful since it allows, e.g., to model nonces in security protocols, guaranteeing that independently created nonces are different.

The mapping of π-calculus into Fusion Calculus presented in [Vic98] maps the input operator of π-calculus into the input of Fusion Calculus, together with an hiding on each of the parameters in the input (which must also be all different). This approach ensures that the transitions of the translations of π-calculus processes correspond to the ones of the original π-calculus processes. However, as pointed out in [BBM04], the mapping is not fully abstract since Fusion Calculus allows more powerful contexts than π-calculus, such as $uzz|\bullet$ which allows to merge two arbitrary names, and which spoils in particular the above said property. The problem can

be solved using either two different binders for input and restriction, as proposed in [BBM04], or a unique binder which takes a set of names as additional parameter, as proposed in [BBM05]. In both the cases distinctions are used to guarantee that some names are kept distinct.

Those techniques, together with the above proposed mapping of Fusion into PRISMA, can be used to bridge the gap between π-calculus and PRISMA Calculus instantiated with Milner synchronization, and a similar approach can be used also to map bπ-calculus into PRISMA instantiated with broadcast synchronization.

A dual approach to the whole problem consists in introducing mobility in the π-calculus style into PRISMA Calculus. In order to do that we must however partition the set of actions into a subset of input actions, a subset of output actions and a subset of control actions. We must then define input prefixes as binders (and force their parameters to be all different). Then, the component $\doteq$ of each action synchronization may just match parameters of input actions and at most one parameter from an output action. If the output parameter exists, then its value is assigned to the input parameters, otherwise the input parameters are simply unified. In both the cases the renaming is applied only inside the scope of the input binders, thus it has no influence on parallel processes. Notice that in this way two distinct names not binded by inputs are never unified as required.

It is easy to see that this approach can be applied to many of the SAMs presented in previous chapters, in particular to Milner, broadcast and priority SAMs, and to many of their variants, but not to symmetric SAMs such as Hoare or threshold SAMs. Thus we think that our approach is more general, since it allows to deal with a larger family of synchronization models.

Part III

Observational semantics and compositionality

Roadmap to part III

Until now we have concentrated our attention on the operational behaviour of systems, that is on the transitions that they are able to perform. In this last part we analyze instead the observational semantics, that is we decide which features of the behaviour of systems are interesting for us, and we define equivalences among systems which quotient systems which are indistinguishable w.r.t. the above said features.

In general, given a model of a full system or of a component, it is important to well understand its interactions with its environment for various reasons: this allows to analyze the behaviour of a system when it is plugged in its execution environment, but this also allows to compute the behaviour of a large and complex system starting from the behaviours of its components.

Thus many formalisms for GC applications are equipped with a well-defined concept of interface that allows to compose different systems, and where interactions take place. In the frameworks that we consider, the interface is modeled by the set of free names. Thus observational semantics analyzes the actions that are performed on the free names.

Different observational semantics exist in the literature, such as bisimilarity, testing semantics, and others, which differ on which features are chosen as interesting and on which mechanisms are used to observe them. We concentrate on bisimilarity, whose basic definition has been presented in Definition 1.2, and on its variants.

From our point of view, a main problem is whether bisimilarity is a congruence w.r.t. the operators of system composition or not, i.e. whether the compositions of bisimilar systems are bisimilar or not. If they are, then the observational properties of a complex system can be derived by composing the results obtained on their components.

In Chapter 8 we concentrate on SHR, and in particular on parametric SHR. We prove that bisimilarity is a congruence w.r.t. a set of operators on graphs for a large class of SAMs. This result is obtained by applying a generalization of the standard bialgebraic techniques outlined in Section 1.2. The application is non trivial, since it requires to define a suitable algebra of graphs and to define SHR transitions by structural induction on the term representing the source of the transition.

The only other approach to observational congruences for graph transformation that we are aware of is [KM01], but the result therein refers to a SHR framework with only Hoare synchronization (instead of the parametric synchronization mechanism of parametric SHR) and where only new nodes can be communicated. This form of mobility is less powerful than ours, and it is not powerful enough to allow a link with Fusion Calculus. Finally, the result in [KM01] allows only restriction and parallel composition as operators for building systems, while we consider also renamings.

In Chapter 9 we apply a similar approach to process calculi instead. In particular, we prove that the concurrent semantics of Fusion Calculus presented in Section 2.3 produces a bisimilarity relation which is a congruence w.r.t. all the operators in the

calculus. The result is interesting since it does not hold for standard interleaving semantics. In that case, in fact, hyperbisimilarity is usually used. A comparison of the two semantics is presented.

Also, we prove that hyperbisimilarity is a congruence for PRISMA Calculus over any SAM S, thus extending a known analogous result for Fusion Calculus. Hyperbisimilarity is also used to analyze some properties of PRISMA Calculus and to relate instances of PRISMA Calculus based on different SAMs.

Chapter 8

Observational properties of parametric SHR

In this chapter we analyze the observational properties of parametric SHR, and in particular we prove that bisimilarity is a congruence w.r.t. the operators of a suitable algebra of graphs for a large class of SAMs. We consider a multi-sorted algebra first, and a single-sorted one later. Note that the presented results are automatically valid also for MSHR and HSHR.

We prove those results following the approach developed in [PT97] and in [BM02], and outlined in Section 1.2, which ensures that the LTS under analysis has a bialgebraic structure. Actually, we develop and exploit a generalization of the theory in [BM02].

Thus we have to provide a semantics for parametric SHR that satisfies the conditions of Theorem 1.1. The condition of Theorem 1.2 requiring that all the axioms bisimulate is not satisfied, but we show that we can add the axioms in two steps, and in this way they all bisimulate.

Actually the LTS of parametric SHR presented in Section 6.1 has a bialgebraic structure, but in order to make this explicit we have to define an algebraic structure on judgements, and rewrite the rules to define the semantics by structural induction on this algebra.

8.1 A bialgebraic framework for SHR

In this section we define an algebra for judgements, and we provide a new version of the inference rules in Definition 6.1 based on that algebra. In particular, we define an algebraic signature Σ_G and a set of axioms E such that the defined algebra of judgements is isomorphic to the term algebra $T_{(\Sigma_G,E)}$.

This algebra is similar to the one presented in [BC87]. The main difference is that we use directly the set of free nodes of each graph as its interface, thus names of free nodes are important and they appear in the sort of the operators. This choice

has been made since, in our interpretation, free nodes are ports of the system which are accessible from the outside, thus they naturally are the interface of the system. Technically, our algebra is multi-sorted, and the sort of each term is a set of names, which is the set of free names of the corresponding graph.

The notation that we use for operators in Σ_G recalls monoidal categories [Mac71], since both the algebra and the congruence property have been discovered while working on a mapping of SHR into the Tile Model [GM00], a categorical framework for studying behavioural properties of systems. The result is however presented here without referring to the Tile Model to make it easily understandable for a larger audience.

Before defining the algebra we introduce some notations.

Notation. *Given a term t we denote with* $\mathrm{sort}(t)$ *its sort and we write* $t : \mathrm{sort}(t)$.

Note that the sort of an operator is a function from tuples of term sorts to term sorts. We can now define the signature Σ_G.

Definition 8.1 (Signature Σ_G for graph terms).
The signature Σ_G is composed by several families of operators, with each family including operators for each of the allowed sorts. We list here the operators, using sort variables to give a compact description of all the operators in the same family:

- $nil : \emptyset$, *a constant representing the empty graph;*
- $L_{x_1,\ldots,x_n} : \{x_1, \ldots, x_n\}$ *where L is an edge label of rank n and $\{x_1, \ldots, x_n\}$ is a set of distinct names, a constant representing the edge $L(x_1, \ldots, x_n)$;*
- $|| : \Gamma_1 \times \Gamma_2 \to \Gamma_1 \cup \Gamma_2$ *where $\Gamma_1 \cap \Gamma_2 = \emptyset$, an operator composing graphs with disjoint sets of free names;*
- $\nabla_{x,y} : \Gamma \to \Gamma \setminus \{y\}$ *where $x, y \in \Gamma$ and $x \neq y$, an operator corresponding to the renaming $\{x/y\}$;*
- $?_x : \Gamma \to \Gamma \setminus \{x\}$ *where $x \in \Gamma$, an operator restricting name x;*
- $!_x : \Gamma \to \Gamma \cup \{x\}$ *where $x \notin \Gamma$, an operator adding a free isolated node x to the graph.*

Note that $\nabla_{x,y}$ is a binder for y and $?_x$ is a binder for x.

We present now an interpretation of the operators in Σ_G on judgements. This allows to give an algebraic structure to the set of judgements (with sort determined by the interface Γ in $\Gamma \vdash G$).

Definition 8.2 (Interpretation function $[\![-]\!]_G$).
The interpretation function $[\![-]\!]_G$ of the operators in Σ_G into judgements is defined

by:

$$[\![nil]\!]_G = \emptyset \vdash nil$$
$$[\![L_{x_1,\ldots,x_n}]\!]_G = x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$$
$$\Gamma_1 \vdash G_1 [\![||]\!]_G \Gamma_2 \vdash G_2 = \Gamma_1, \Gamma_2 \vdash G_1 | G_2$$
$$[\![\nabla_{x,y}]\!]_G \Gamma, x, y \vdash G = \Gamma, x \vdash G\{x/y\}$$
$$[\![?_x]\!]_G \Gamma, x \vdash G = \Gamma \vdash \nu x\ G$$
$$[\![!_x]\!]_G \Gamma \vdash G = \Gamma, x \vdash G$$

We assume the trivial extension of the interpretation function $[\![-]\!]_G$ to terms.

Note that the extension of $[\![-]\!]_G$ to terms provides an evaluation of terms with judgements as values.

We show here a simple example of interpretation of a term.

Example 8.1.
The interpretation function $[\![-]\!]_G$ maps the graph term $?_y \nabla_{y,y_1} \nabla_{x,x_1}(R_{x,y}||R_{y_1,x_1})$ to the judgement $x \vdash \nu y\ R(x,y)|R(y,x)$, which is the two elements ring from Example 1.1.

The following lemma shows that the sort of a term is the interface of the represented graph.

Lemma 8.1. *Let t be a term in T_{Σ_G}: $[\![t]\!]_G = \Gamma \vdash G$ iff $\mathrm{sort}(t) = \Gamma$.*

Proof. By structural induction on t. □

Next lemma proves that the above definition indeed defines an algebraic structure on judgements.

Lemma 8.2. *Judgements together with the interpretation $[\![-]\!]_G$ of the operators in the signature Σ_G form a Σ_G-algebra.*

Proof. One just needs to check that the defined functions have the correct domain and codomain, where domains and codomains are determined by the sort of the corresponding operators. □

The above definition provides an algebraic structure on judgements, but it does not consider structural axioms. We want to refine it to define a similar structure on judgements up to axioms. We introduce a special notation to clarify when we consider judgements up to axioms.

Notation. *We denote with $[\Gamma \vdash G]_\equiv$ the equivalence class of $\Gamma \vdash G$ modulo the axioms of structural congruence.*

We can now define the interpretation function $[\![-]\!]^{\equiv}_{G}$ mapping operators in Σ_G into functions on judgements up to structural congruence. It simply considers a representative of the class of judgements, it applies the interpretation function $[\![-]\!]_G$ to it and it takes the equivalence class of the result.

Definition 8.3 (Interpretation function $[\![-]\!]^{\equiv}_{G}$).
The interpretation function $[\![-]\!]^{\equiv}_{G}$ is defined on a unary operator op by $[\![op]\!]^{\equiv}_{G}[\Gamma \vdash G]_{\equiv} = [[\![op]\!]_G \Gamma \vdash G]_{\equiv}$. The definition is analogous for constants and binary operators.

We have to check that this definition is well-posed.

Lemma 8.3. *Definition 8.3 is well-posed, thus judgements modulo axioms of structural congruence together with the interpretation $[\![-]\!]^{\equiv}_{G}$ of the operators in Σ_G form a Σ_G-algebra.*

Proof. One just needs to check that the result of each function does not depend on the choice of the representative in the equivalence class in the left part. □

We want to define a set of axioms E on Σ_G-terms that captures the equivalences that hold among judgements up to structural axioms.

Definition 8.4 (Axiomatization of judgements). *The set of axioms E for the algebra of graph terms contains the families of axioms defined by schemas in Table 8.1 (axioms are required for each allowed sort).*

Even if the axiomatization requires many axioms, they are all pretty intuitive. Axioms 1 and 2 allow α-conversion of names bound by either the operator ? or the operator ∇. Note that $\nabla_{x,y}$ binds only the parameter y. Axioms 3, 4 and 5 allow to commute operator $||$ with other operators. Similarly axioms 6 and 7 allow to commute ∇ with the remaining operators and axiom 8 allows to commute ? and !. All these axioms can be applied only if the commuted operators are labeled by different names. Axioms 9 and 10 concern the application of a ∇ operator to a name x generated by a $!_x$ operator. If x is chosen as representative by ∇, then a renaming has to be applied to the term (axiom 10), otherwise the two operators can simply be deleted (axiom 9). Similarly, if a name x just created by $!_x$ is bound using $?_x$, then the two operators can be deleted (axiom 11). This provides garbage-collection of bound names. Axioms 12 and 13 formalize commutativity and associativity of operator $||$, while axiom 14 states that *nil* is its neutral element. Axioms 15, 16 and 17 ensure that the properties of the renamings represented by ∇ operators are verified. Finally axioms 18 and 19 allow to commute two occurrences of operators ? and ! respectively, provided that they have different names as parameters.

We define here a normal form for terms in T_{Σ_G}.

Definition 8.5 (Normal form for terms in T_{Σ_G}). *A term $t \in T_{\Sigma_G}$ is in normal form iff it has the structure $c_! c_? c_\nabla t_i$ where:*

1. $?_x\, t =_{\mathrm{E}} ?_y(t\{y/x\})$ if $y \notin \mathrm{sort}(t)$
2. $\nabla_{x,y} t =_{\mathrm{E}} \nabla_{x,z}(t\{z/y\})$ if $z \notin \mathrm{sort}(t)$
3. $(\nabla_{x,y} t_1)||t_2 =_{\mathrm{E}} \nabla_{x,y}(t_1||t_2)$ if $y \notin \mathrm{sort}(t_2)$
4. $(?_x\, t_1)||t_2 =_{\mathrm{E}} ?_x(t_1||t_2)$ if $x \notin \mathrm{sort}(t_2)$
5. $(!_x\, t_1)||t_2 =_{\mathrm{E}} !_x(t_1||t_2)$
6. $\nabla_{x,y}\, ?_z\, t =_{\mathrm{E}} ?_z\, \nabla_{x,y} t$ if $z \neq x, y$
7. $\nabla_{x,y}\, !_z\, t =_{\mathrm{E}} !_z\, \nabla_{x,y} t$ if $z \neq x, y$
8. $?_x\, !_y\, t =_{\mathrm{E}} !_y\, ?_x\, t$ if $x \neq y$
9. $\nabla_{x,y}\, !_y\, t =_{\mathrm{E}} t$
10. $\nabla_{x,y}\, !_x\, t =_{\mathrm{E}} t\{x/y\}$
11. $?_x\, !_x\, t =_{\mathrm{E}} t$
12. $t_1||t_2 =_{\mathrm{E}} t_2||t_1$
13. $(t_1||t_2)||t_3 =_{\mathrm{E}} t_1||(t_2||t_3)$
14. $t||nil =_{\mathrm{E}} t$
15. $\nabla_{x,y}\nabla_{z,w} t =_{\mathrm{E}} \nabla_{z,w}\nabla_{x,y} t$ if $\{x, y\} \cap \{z, w\} = \emptyset$
16. $\nabla_{x,y}\nabla_{y,z} t =_{\mathrm{E}} \nabla_{x,y}\nabla_{x,z} t$ if $x \neq y \neq z$
17. $\nabla_{x,y}\nabla_{x,z} t =_{\mathrm{E}} \nabla_{x,z}\nabla_{x,y} t$ if $x \neq y \neq z$
18. $?_x\, ?_y\, t =_{\mathrm{E}} ?_y\, ?_x\, t$
19. $!_x\, !_y\, t =_{\mathrm{E}} !_y\, !_x\, t$

Table 8.1: Axiomatization of graph terms.

1. *t_i is the constant nil if term t contains no edge constants and a $||$ composition of edge constants left associated, in a canonical order, and with distinct names for nodes otherwise;*

2. *for each $\sigma \in \{!, ?, \nabla\}$ c_σ is a context composed only by operators σ;*

3. *in each operator $\nabla_{x,y}$ the second argument is always distinct from any other argument of a ∇ operator, and the order of operators is determined by the order of the occurrences of x and y in the inner part;*

4. *the operators $?_x$ and $!_x$ appearing in the respective contexts are in a canonical order;*

5. *names y in $\nabla_{x,y}$ and names x in $?_x$ are canonical names.*

Note that, as in Definition 3.12, we suppose to have a canonical order on names and edge constants.

Notice also that the graph term in Example 8.1 is in normal form, for a suitable standard ordering on names.

Next lemma proves that the above defined normal form finds a standard representative of equivalence classes of terms modulo E.

Lemma 8.4. *For each term $t \in T_{\Sigma_G}$ there exists a unique term $t' \in T_{\Sigma_G}$ in normal form such that $t =_{\mathrm{E}} t'$.*

Proof. We prove the existence first and the uniqueness later. Note that we can always use the axioms to sort the operators in the order (from inside to outside) constants, $||$, ∇, ?, ! to match the term structure imposed by conditions 1 and 2. Suppose for instance that there is a $||$ operator outside some other operator: we can make it to commute with ∇ using axiom 3 (if the side condition is not satisfied then we can use axiom 2 to α-convert y and thus satisfy it). Similarly we can use axiom 4 to commute $||$ and ? (if the side condition is not satisfied then we can use axiom 1 to solve the problem) and axiom 5 to commute $||$ and !. Similarly we can commute the other operators. Thus we can suppose that the term has the desired structure.

Condition 1 can be satisfied using axiom 12 to sort the edge constants in the desired order, axiom 13 to make them left associated and axiom 14 to delete exceeding *nil* constants. Note that names are necessarily distinct otherwise the $||$ operator is not applicable. Condition 3 can be satisfied using axioms 15, 16 and 17. Condition 4 can be satisfied using axioms 18 and 19. The last condition can be satisfied using axioms 1 and 2. We can always find a sequence of renamings that makes the side conditions satisfied, since we want to end up with distinct names.

Suppose now to have two terms t_1 and t_2 in normal form such that $t_1 =_{\mathrm{E}} t_2$. Note that the only operators that can be deleted using the axioms are ! operators (in this case also a ∇ operator or a ? operator is deleted) and *nil*. However the normal form forces both the numbers of ! operators and *nil* to be minimal. In fact, a ! operator

can be deleted only if it is inside a ∇ operator or a ? operator, and this can not happen in a normal form. Also, there is at most one *nil* constant, and none of them if there are other constants. Thus the two terms can differ just for the ordering among the operators of the same kind or for their arguments. However the order among the operators is fixed, thus no choice is possible. Furthermore, names that do not appear in the sort are canonical and thus fixed (the other names can not be changed since subject reduction holds). Finally, the order is forced also among the arguments of different ∇ operators, thus no choice is possible again. Thus the two terms must be equal. □

We show now that the axiomatization captures the properties of judgements up to structural axioms.

Lemma 8.5. *For each pair of terms* $t, t' \in T_{\Sigma_G}$, $t =_{\mathrm{E}} t'$ *iff* $[\![t]\!]_G \equiv [\![t']\!]_G$.

Proof. We prove that there is a bijection between normal forms for graph terms and standard decompositions of judgements, as defined in Definition 3.12. Since each normal form corresponds to an equivalence class thanks to Lemma 8.4, and the same happens for standard decompositions of judgements thanks to Lemma 3.2, then the thesis will follow. Let us consider a standard decomposition of a judgement $\Gamma \vdash G$, which has the form $\Gamma \vdash \nu X\ \hat{G}\sigma_G$. We want to build in a unique way a term t in normal form such that $[\![t]\!]_G \equiv \Gamma \vdash G$.

We follow the steps below:

1. the inner part of t has the same structure of $\hat{G}$, with constants for *nil* and for the edges, composed using the $||$ operator;
2. to the inner part, an operator $\nabla_{x,y}$ is applied for each binding $\{x/y\}$ in σ_G, in the canonical order;
3. then an operator $?_x$ is applied for each name in X, in the same order;
4. finally an operator $!_x$ is applied for each name in $\Gamma \backslash \mathrm{fn}(\nu X\ \hat{G}\sigma_G)$, in a canonical order.

Note that the steps above can always be applied (in particular, when an operator has to be applied, the term has a compatible sort), that the built term t is in normal form by construction (supposing to use corresponding canonical orders for names in judgements and in terms) and that the construction is deterministic. The correctness of the construction can be proved step by step, since the interpretations of the terms obtained after each step are respectively $\mathrm{fn}(\hat{G}) \vdash \hat{G}$, $\mathrm{fn}(\hat{G}\sigma_G) \vdash \hat{G}\sigma_G$, $\mathrm{fn}(\nu X\ \hat{G}\sigma_G) \vdash \nu X\ \hat{G}\sigma_G$ and finally the wanted judgement.

Note that the defined correspondence is indeed a bijection, as required. □

An easy corollary of the result above is that the algebra of judgements modulo structural congruence is isomorphic to the term algebra $T_{(\Sigma_G,E)}$.

Corollary 8.1. *The algebra of judgements modulo structural congruence of Lemma 8.3 is isomorphic to the algebra* $T_{(\Sigma_G,E)}$.

Proof. It is easy to check that the bijective correspondence defined by Lemma 8.5 is an isomorphism. □

Thanks to the above result we can use the two algebras interchangeably.

Now that we have defined the needed algebraic structure we give a presentation of the rules for parametric SHR based on it. In particular, we use an inference rule for each operator (we write schemas of rules summarizing the rules for similar operators with different sorts).

Since the rules will be used to derive transitions for terms up to axioms, we can use some notational extensions justified by the axioms.

Notation. *Given a set* X *we denote with* $?_X$ *(resp.* $!_X$*) any composition of* $?_{x_i}$ *(resp.* $!_{x_i}$*) operators such that* $X = \bigcup_i \{x_i\}$. *Similarly, given an idempotent renaming* ρ *we denote with* ∇_ρ *any composition of* $\nabla_{x,y}$ *operators whose renamings* $\{x/y\}$*, composed in the order of application of the operators, equal* ρ.

The rules are defined below.

Definition 8.6 (Inference rules for parametric SHR). *The admissible behaviours of parametric SHR are defined by the following inference rules, which are parametric on a SAM* $S = (Id, (Act, \mathrm{ar}, \epsilon), Init, Fin, ActSyn)$.

$$\textit{(nil-pa)} \quad \frac{-}{nil \xrightarrow{\emptyset,id} nil}$$

$$\textit{(edge-pa)} \quad \frac{x_1,\ldots,x_n \vdash L(x_1,\ldots,x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G}{L_{x_1,\ldots,x_n} \xrightarrow{\Lambda,\pi} t}$$

where $[\![t]\!]_G = \Phi \vdash G$.

$$\textit{(par-pa)} \quad \frac{t_1 \xrightarrow{\Lambda,\pi} t_2 \qquad t_1' \xrightarrow{\Lambda',\pi'} t_2'}{t_1||t_1' \xrightarrow{\Lambda\cup\Lambda',\pi\cup\pi'} t_2||t_2'}$$

where $(\mathrm{sort}(t_1) \cup \mathrm{sort}(t_2)) \cap (\mathrm{sort}(t_1') \cup \mathrm{sort}(t_2')) = \emptyset$.

$$\textit{(merge-pa)} \quad \frac{t_1 \xrightarrow{\Lambda,\pi} t_2}{\nabla_{x,y} t_1 \xrightarrow{\Lambda',\pi'} ?_U \nabla_{\rho'} \nabla_{x,y} t_2}$$

where:

1. $\Lambda(x) = (a_1, \vec{v}_1), \Lambda(y) = (a_2, \vec{v}_2)$
2. $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in ActSyn$

3. $S_1 = \{\vec{v}_{i_1}[j_1] = \vec{v}_{i_2}[j_2] \mid \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\}$
4. $S_2 = \{t = u \mid t, u \in \mathrm{sort}(t_1) \wedge t\pi = u\pi\}$
5. $\rho = \mathrm{mgu}((S_1 \cup S_2)\{x/y\})$ *where* ρ *maps names to representatives in* $\mathrm{sort}(t_1)$ *whenever possible*
6. $\rho' = \rho\lfloor_{\mathrm{sort}(\nabla_{x,y}t_2)}$
7. $\vec{w}[i] = (\vec{v}_j[k])\{x/y\}\rho$ *if* $\mathrm{Mob}(i) = \mathrm{inj}_j(k)$
8. $\Lambda'(z) = \begin{cases} (c, \vec{w}) \text{ if } z = x \\ (\mathrm{act}_\Lambda(z), (\mathrm{n}_\Lambda(z))\{x/y\}\rho) \text{ for each } z \in \mathrm{sort}(t_1) \setminus \{x, y\} \end{cases}$
9. $\pi' = \rho\lfloor_{\mathrm{sort}(\nabla_{x,y}t_1)}$
10. $U = ((\mathrm{n}(\Lambda))\{x/y\}\rho \setminus \mathrm{n}(\Lambda')) \cap \mathrm{sort}(\nabla_{\rho'}\nabla_{x,y}t_2)$

$$\textit{(res-pa)} \quad \frac{t_1 \xrightarrow{\Lambda,\pi} t_2}{?_x\, t_1 \xrightarrow{(\Lambda\lfloor_{\setminus\{x\}})\{y/x\},(\pi\lfloor_{\setminus\{x\}})\{y/x\}} (?_Z\, t_2)\{y/x\}}$$

where:

11. $(\exists z \in \mathrm{sort}(t_1).x\pi = z\pi \wedge x \neq z) \Rightarrow x\pi \neq x$
12. $\mathrm{act}_\Lambda(x) \in Fin$
13. $Z = ((\mathrm{n}(\Lambda) \cup \{x\}) \setminus \mathrm{n}(\Lambda\lfloor_{\setminus\{x\}})) \cap \mathrm{sort}(t_2)$
14. $y = x$ *or* $y \notin \mathrm{sort}(t_1) \cup \mathrm{n}(\Lambda\lfloor_{\setminus\{x\}})$

$$\textit{(new-pa)} \quad \frac{t_1 \xrightarrow{\Lambda,\pi} t_2}{!_x\, t_1 \xrightarrow{\Lambda\cup(x,i,\vec{y}),\pi} !_{\mathrm{n}(\vec{y})}\, !_x\, t_2}$$

where $x \notin \mathrm{sort}(t_1) \cup \mathrm{sort}(t_2)$, $i \in Init$ *and* $\vec{y}$ *is a vector of names such that* $\mathrm{n}(\vec{y}) \cap (\mathrm{sort}(t_1) \cup \mathrm{sort}(t_2) \cup \{x\}) = \emptyset$ *and* $\mathrm{ar}(i) = |\vec{y}|$.

The intuition behind these rules is exactly the one behind rules in Definition 6.1, but there are however some important technical differences.

First of all, the rules (nil-pa) and (edge-pa) have been added: the first rule adds a trivial transition for term *nil* and no other transitions, thus it is not important from an operational point of view but it is necessary to make axiom 14 in E to bisimulate. Rule (edge-pa) just explicitly introduces in the LTS the transitions that correspond to productions.

In the rules it is no more possible to refer to contexts Γ or Φ, since they are no more visible, thus they are replaced by the sort of the corresponding graph terms. Also, restrictions on the final graph must be computed explicitly, since they can no more be computed using the information on contexts (see condition 10 of rule (merge-pa) and condition 13 of rule (res-pa)).

In order to apply renamings to terms sequences of ∇ operators are used, but, to make the sort requirements satisfied, renamings are restricted to the names in the sort (see, e.g., condition 6 of rule (merge-pa)). The same trick is required to bind names via the ? operator.

Finally, in rule (res-pa), if the bound name x is extruded then we allow to choose a different name y for it. If judgements are considered up to structural axioms, this can be obtained by α-converting x into y, but this possibility must be included in the rule to make axiom 1 in E to bisimulate.

Notation. *We write* $\mathcal{P} \Vdash_{pa(S)} t_1 \xrightarrow{\Lambda,\pi} t_2$ *iff* $t_1 \xrightarrow{\Lambda,\pi} t_2$ *can be obtained from the productions in* $\mathcal{P}$ *using parametric inference rules from Definition 8.6 instantiated with SAM* S.

We show here an example on how to derive a transition using the rules above.

Example 8.2. *This example is based on Milner amoeboids, as defined in Definition 3.19. A suitable Milner SAM is used for synchronization. We will further extend the example in the following.*

Let us consider the term $?_{n_1} \nabla_{n_1,n_2}(R_{x,n_1,y}||R_{z,n_2,w})$, *which represents an amoeboid with interface* $\{x,y,z,w\}$. *Let us consider an action* a *such that* $\mathrm{ar}(a) = 1$.

From rule (edge-pa) we can derive:

$$R_{x,n_1,y} \xrightarrow{(x,a,\langle y_1\rangle),(n_1,\overline{a},\langle y_2\rangle),id} R_{x,n_1,y}||L_{y_1,y_2}$$
$$R_{z,n_2,w} \xrightarrow{(n_2,a,\langle z_1\rangle),(w,\overline{a},\langle z_2\rangle),id} R_{z,n_2,w}||L_{z_1,z_2}$$

We can then apply rule (par-pa) to have:

$$R_{x,n_1,y}||R_{z,n_2,w} \xrightarrow{(x,a,\langle y_1\rangle),(n_1,\overline{a},\langle y_2\rangle),(n_2,a,\langle z_1\rangle),(w,\overline{a},\langle z_2\rangle),id} R_{x,n_1,y}||L_{y_1,y_2}||R_{z,n_2,w}||L_{z_1,z_2}$$

Now we can use rule (merge-pa) to synchronize actions at n_1 *and* n_2*:*

$$\nabla_{n_1,n_2}(R_{x,n_1,y}||R_{z,n_2,w}) \xrightarrow{(x,a,\langle y_1\rangle),(n_1,\tau,\langle\rangle),(w,\overline{a},\langle z_2\rangle),id} ?_{y_2} \nabla_{y_2,z_1} \nabla_{n_1,n_2}(R_{x,n_1,y}||L_{y_1,y_2}||R_{z,n_2,w}||L_{z_1,z_2})$$

Finally we can apply rule (res-pa) to get:

$$?_{n_1} \nabla_{n_1,n_2}(R_{x,n_1,y}||R_{z,n_2,w}) \xrightarrow{(x,a,\langle y_1\rangle),(w,\overline{a},\langle z_2\rangle),id} ?_{n_1} ?_{y_2} \nabla_{y_2,z_1} \nabla_{n_1,n_2}(R_{x,n_1,y}||L_{y_1,y_2}||R_{z,n_2,w}||L_{z_1,z_2})$$

which is an allowed transition.

The theorem below ensures that the above defined semantics is the usual semantics of parametric SHR from Definition 6.1. However it holds only if the used SAM satisfies the following conditions, which are called conditions for SAM compositionality since they are needed to prove the congruence theorem (Theorem 8.3).

Definition 8.7 (Conditions for SAM compositionality). *A SAM S satisfies the* conditions for SAM compositionality *iff:*

1. $\mathrm{fInit}(S) \cap \mathrm{fFin}(S) \neq \emptyset$;
2. *if there is an action synchronization* $(\epsilon, i, (i, \mathrm{Mob}, \doteq)) \in \mathrm{fActSyn}(S)$ *with* $i \in \mathrm{fInit}(S)$ *then* $i = \epsilon$.

Note that these conditions are satisfied by most SAMs (Milner, Hoare, broadcast, priority, threshold, ...), thus the theorem is general anyway. Also, if the properties are not satisfied, then the congruence theorem does not hold, and the correspondence is not needed.

We have not required these properties to hold for any SAM since a few useful SAMs, and in particular the account SAM of Example 7.3 and the multicast SAM, do not satisfy them.

We can now prove the correspondence theorem.

Theorem 8.1. *Let $\mathcal{P}$ be a set of productions, S a SAM satisfying the conditions for SAM compositionality and t a graph term (different from nil). The two following statements are equivalent:*

- $\mathcal{P} \Vdash_{pa(S)} t_1 \xrightarrow{\Lambda,\pi} t_2$;
- $\mathcal{P} \Vdash_{p(S)} [\![t_1]\!]_G^{\equiv} \xrightarrow{\Lambda,\pi} [\![t_2]\!]_G^{\equiv}$.

Proof. The proof is by rule induction, and it is almost trivial since the two sets of inference rules are very similar. The only important difference is that the first derivation does not use structural congruence, but all the axioms of structural congruence bisimulate, as will be proved in Lemma 8.8 and Lemma 8.9 (this proof requires the conditions for SAM compositionality), thus any representative of the equivalence class allows the same transitions. □

8.2 Observational semantics for SHR

In this section we exploit the formal framework developed in the previous section to prove that bisimilarity is a congruence for parametric SHR w.r.t. all the operators in the algebra $T_{(\Sigma_G, E)}$ of graph terms for a large class of SAMs, namely for SAMs satisfying the conditions for SAM compositionality. Since the algebra is multi-sorted, the most straightforward congruence result concerns only terms of the same sort.

We prove this result using a generalization of the theory developed in Section 1.2, that is proving that the inference rules are in De Simone format and that all the axioms in E bisimulate.

Lemma 8.6. *All the rules defined by the schemas in Definition 8.6 are in De Simone format.*

Proof. The proof is by inspection. Notice that for each schema we must consider all the instances which are obtained by choosing a particular sort for the starting graph and a particular label among the ones satisfying the side conditions. Notice also that this fixes the sort of the result. The obtained instances have no side conditions.

The only rule that requires an additional trick is rule (res-pa), since its target term contains a renaming $\{y/x\}$. If $y = x$ or x is not free in the term then the renaming can simply be deleted. Otherwise it can be obtained by applying the context $\nabla_{y,x}\,!_y$. □

We have to prove that all the axioms bisimulate. Actually, this is not the case, as shown by the following examples.

Example 8.3. *Axiom* $(?_x\,t_1)||t_2 =_{\mathrm{E}} ?_x(t_1||t_2)$ *if* $x \notin \mathrm{sort}(t_2)$ *(Axiom 4 in Table 8.1) does not bisimulate. In fact, if the transition chosen for* t_2 *in the derivation introduces* x *as freshly generated name, then in the RHS the side condition of rule (par-pa) is not satisfied (since* x *occurs in the both the transitions). Vice versa, in the LHS rule (res-pa) is applied first, thus* x *disappears from the first premise (unless it is extruded) and it can be used in the second one. Thus we can derive for the LHS a transition that has no correspondent for the RHS.*

Example 8.4. *Axiom* $?_x\,!_x\,t =_{\mathrm{E}} t$ *(Axiom 11 in Table 8.1) does not bisimulate if the first condition for SAM compositionality does not hold. Consider, e.g., the instance* $?_x\,!_x\,nil =_{\mathrm{E}} nil$. *The RHS can perform the transition* $nil \xrightarrow{\emptyset,id} nil$. *Trying to derive the same transition for the LHS one can just derive* $!_x\,nil \xrightarrow{(x,i,\vec{y}),id} !_{\mathrm{n}(\vec{y})}\,!_x\,nil$ *for some* $i \in \mathrm{fInit}(S)$, *but then rule (res-pa) is not applicable, thus no transition is derivable for the LHS.*

Other axioms show similar difficulties. Thus to prove the desired theorem we have to refine the approach. First, we restrict our attention to SAMs satisfying the conditions for SAM compositionality. Luckily, most of the used SAMs are included in this class, thus the approach is still general. Second, we add the axioms to the algebra in two steps, so that we can use the axioms added in the first step in the proof of the second one. We present here a generalization of Theorem 1.2 that enables us to do that.

Theorem 8.2. *Let* A *be a* Σ*-algebra for some signature* Σ, *lts be a LTS with states in* A *and* $=_{\mathrm{E}}$ *be a congruence relation among elements in* A *generated by a set* E *of axioms. If lts is a bialgebra with underlying algebra* A, *i.e. it corresponds to a*

coalgebra $h : A \to F(A)$ *with* $F : \mathcal{A}lg(\Sigma) \to \mathcal{A}lg(\Sigma)$ *a functor, and for all equations* $t_1 =_{\mathrm{E}} t_1'$ *in* E*, with free variables* $\{x_i | i \in I\}$*, we have proofs of:*

$$\frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1 \xrightarrow{l} t_2} \text{ implies } \frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1' \xrightarrow{l} t_2'} \text{ and } t_1' =_{\mathrm{E}} t_2' \tag{8.1}$$

and vice versa, then the LTS with equivalence classes of elements in A *modulo* E *as states and transitions defined by:*

$$[t_1]_{=_{\mathrm{E}}} \xrightarrow{l} [t_2]_{=_{\mathrm{E}}} \text{ if } t_1 \xrightarrow{l} t_2$$

has a bialgebraic structure.

Proof. We have to prove that the bialgebraic structure of A induces a corresponding bialgebraic structure on A modulo E.

Let $h : A \to F(A)$ be the coalgebra in $\mathcal{A}lg(\Sigma)$ corresponding to lts. We have to define a new coalgebra $h_E : A_E \to F(A_E)$ where A_E is the quotient of A modulo E corresponding to the quotiented LTS and prove that it is well-defined.

We define h_E as follows:

$$\begin{aligned} h_E(\{[t_i]_{=_{\mathrm{E}}} | i \in I, t_i \in |A|\}) = \{[h(t_i)]_{=_{\mathrm{E}}} | i \in I, t_i \in |A|, h(t_i) \in |A|\} \cup \\ \cup \{\langle l, [t]_{=_{\mathrm{E}}} \rangle | i \in I, t_i \in |A|, h(t_i) = \langle l, t \rangle, l \in L, t \in |A|\} \end{aligned}$$

The defined coalgebra corresponds to the quotiented LTS as required. It is well-defined iff each choice of a representative inside the equivalence class produces the same result. Let t_1 and t_2 be two such representatives. We have that $t_1 =_{\mathrm{E}} t_2$ iff there exists a chain of equalities $t_1 =_{\mathrm{E}} t_1'$, $t_1' =_{\mathrm{E}} t_2'$, $\ldots$, $t_n' =_{\mathrm{E}} t_2$ where each equality is derived from an instance of an axiom in E using congruence or symmetry. It is enough to prove the thesis for a single equality, since then it follows by transitivity. Symmetry is trivial too. Let us consider first an instance of an axiom. Condition 8.1 ensures exactly the result. The congruence property follows since h is a bialgebra. □

We have now to prove that the axioms in E satisfy the desired conditions. As anticipated, we start with a subset of them.

Before proving the lemma, we present an auxiliary result useful to deal with injective renamings.

Lemma 8.7. *Let* ψ *be an injective renaming,* $\mathcal{P}$ *a set of productions and* S *a SAM. If* $\mathcal{P} \Vdash_{pa(S)} t_1 \xrightarrow{\Lambda,\pi} t_2$ *then* $\mathcal{P} \Vdash_{pa(S)} t_1\psi \xrightarrow{\Lambda\psi,\pi\psi} t_2\psi$.

Proof. By rule induction. □

We can now prove the main lemma below.

Lemma 8.8. *For axioms 1, 2, 5, 12, 13, 14, 15, 16, 17, 19 in Table 8.1, with free variables $\{x_i\}_{i\in I}$, we have proofs of:*

$$\frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1 \xrightarrow{l} t_2} \text{ implies } \frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1' \xrightarrow{l} t_2'} \text{ and } t_2 =_{\mathrm{E}} t_2'$$

and vice versa, using the rules in Definition 8.6 instantiated with a SAM S.

Proof. The proof has a case for each axiom. Since each case is quite technical, we show in detail just a pair of them as examples, while adding some short suggestions on the proofs of the other ones.

Axiom 1) $?_x\, t =_{\mathrm{E}} ?_y(t\{y/x\})$ if $y \notin \mathrm{sort}(t)$.

Suppose that we have a premise of the form $t \xrightarrow{\Lambda,\pi} t'$. Then using rule (res-pa) we can derive:

$$?_x\, t \xrightarrow{(\Lambda|_{\setminus\{x\}})\{z/x\},(\pi|_{\setminus\{x\}})\{z/x\}} (?_Z\, t')\{z/x\}$$

where we have to satisfy the side conditions of rule (res-pa).

Thanks to Lemma 8.7, choosing as renaming $\{y/x\}$ which is injective since $y \notin \mathrm{sort}(t)$ we have $t\{y/x\} \xrightarrow{\Lambda\{y/x\},\pi\{y/x\}} t'\{y/x\}$.

We can apply rule (res-pa) to bind y iff we can apply rule (res-pa) to bind x in the transition above since the side conditions 11 and 12 are satisfied in the same cases.

Thus we have:

$$?_y(t\{y/x\}) \xrightarrow{(\Lambda\{y/x\}|_{\setminus\{y\}})\{z/y\},(\pi\{y/x\}|_{\setminus\{y\}})\{z/y\}} (?_{Z'}\, t'\{y/x\})\{z/y\}$$

The allowed choices for z are the same in both the cases, since either $z \neq x, y$, thus it is a new name in both the transitions, or $z = y$, which can be chosen as new name in the first transition, and as previously used name in the second one, or $z = x$, swapping the two motivations.

Also, $(\Lambda\{y/x\}|_{\setminus\{y\}})\{z/y\} = (\Lambda|_{\setminus\{x\}})\{y/x\}\{z/y\} = (\Lambda|_{\setminus\{x\}})\{z/x\}$, and the same happens for π.

Finally, the names in Z and in Z' are the same ones, apart from x in Z (if it occurs there) which is replaced by y in Z'. If it is bound, then the renaming can be discarded and the two terms are equal up to axiom 1, otherwise the two terms are equal.

Axiom 2) $\nabla_{x,y} t =_{\mathrm{E}} \nabla_{x,z}(t\{z/y\})$ if $z \notin \mathrm{sort}(t)$.

Suppose that we have a premise of the form $t \xrightarrow{\Lambda,\pi} t'$. Then using rule (merge-pa) we can derive a transition $\nabla_{x,y} t \xrightarrow{\Lambda',\pi'} ?_U\, \nabla_{\rho'} \nabla_{x,y} t'$ provided that the side conditions are satisfied. Using Lemma 8.7 with renaming $\{z/y\}$ we can derive $t\{z/y\} \xrightarrow{\Lambda\{z/y\},\pi\{z/y\}} t'\{z/y\}$. Then we can apply rule (merge-pa) to this transition obtaining as a result $\nabla_{x,z}(t\{z/y\}) \xrightarrow{\Lambda',\pi'} ?_U\, \nabla_{\rho'} \nabla_{x,z}(t'\{z/y\})$. Notice that Λ' and π'

are the same as before, since the renaming $\{x/z\}$ is applied before computing them. Thus also ρ' and U are as before, and the two resulting terms are equivalent.

Axiom 5) The proofs for the two transitions simply swap the order of application of the operators. Since the side conditions require just the names of the two transitions and the new names to be disjoint, they are satisfied independently of the order of application. Also, the condition that the performed action is in $\mathrm{fInit}(S)$ is satisfied in a derivation iff it is satisfied in the other.

Axioms 12 and 13) The proof is based on commutativity and associativity of disjoint union.

Axiom 14) The proof is based on the existence of the idle transition for *nil*.

Axioms 15, 16 and 17) The proof is based on the independence between different synchronizations and on associativity and commutativity of SAMs.

Axiom 19) The proof is based on the independence between different newly created names. □

In order to make the remaining axioms to bisimulate we require the used SAM to satisfy the conditions for SAM compositionality, and we exploit the previous axioms.

Lemma 8.9. *For axioms 3, 4, 6, 7, 8, 9, 10, 11, 18 with free variables $\{x_i\}_{i\in I}$, we have proofs of:*

$$\frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1 \xrightarrow{l} t_2} \text{ implies } \frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1' \xrightarrow{l} t_2'} \text{ and } t_2 =_{\mathrm{E}} t_2'$$

and vice versa, using the rules in Definition 8.6, instantiated with a SAM satisfying the conditions for SAM compositionality, and structural congruence with the axioms listed in Lemma 8.8.

Proof. The proof has a case for each axiom. Since each case is quite technical, we show in detail just a pair of them as examples, while adding some short suggestions on the proofs of the other ones. Also, proofs are similar to the ones in Lemma 8.8.

Axiom 3) The proof for this axiom is based on the fact that names in t_2 are not involved in the synchronization caused by $\nabla_{x,y}$, and is similar to the following one, which is simpler. Just note that if the transition for t_2 creates a new name, then y is a correct choice for this name in the LHS, but not in the RHS. However, using axiom 2, we can change the second parameter of ∇ to a different name. After that, y can be used also in the RHS. A similar trick is required for all the axioms featuring operators ∇ and $?$, which delete names, together with other operators requiring that two names are different.

Axiom 4) $(?_x\, t_1)||t_2 =_{\mathrm{E}} ?_x(t_1||t_2)$ if $x \notin \mathrm{sort}(t_2)$.

Suppose to have two premises of the form $t_1 \xrightarrow{\Lambda_1,\pi_1} t_1'$ and $t_2 \xrightarrow{\Lambda_2,\pi_2} t_2'$.

Using rule (res-pa) we can derive:

$$?_x\, t_1 \xrightarrow{(\Lambda_1 |_{\setminus\{x\}})\{y/x\},(\pi_1 |_{\setminus\{x\}})\{y/x\}} (?_Z\, t_1')\{y/x\}$$

provided that the side conditions are satisfied.

Then we can add the second premise in parallel using rule (par-pa), provided that it uses a disjoint set of free names. For the RHS of the axiom, the two steps must be applied in the opposite order. In both the cases y, if it is extruded, must not be used in the second premise (otherwise, in the first case, the side condition of (par-pa) is not satisfied, while in the second one the side condition 14 of rule (res-pa) is not satisfied). The other side conditions of (res) are equally satisfied in both the derivations, since they do not involve free names in the second premise. The second premise can not contain x too, because of the side condition of the axiom, but the transition can use x as new name in the LHS but not in RHS. The same trick of previous axiom is required, using axiom 1 instead of axiom 2.

Axioms 6 and 7) The proof is based on the independence of the names on which the synchronization is performed w.r.t. the other name.

Axiom 8) The proof is based on the independence between the hidden name and the new name.

Axioms 9 and 10) The proof is based on the condition 3 of the definition of SAM to guarantee that the LHS has at least the same transitions as the RHS, and on the condition 2 to guarantee that it has no further transitions. The second condition for SAM compositionality guarantees that when action ϵ is done on the old merged node, synchronizing it with an action in $\mathrm{fInit}(S)$ can not add further possibilities.

Axiom 11) The proof is based on the fact that $\mathrm{fInit}(S) \cap \mathrm{fFin}(S) \neq \emptyset$ (first condition for SAM compositionality), thus the new node created by rule (new-pa) can be bound by rule (res-pa).

Axiom 18) The proof is based on the independence between different restricted names. □

We can now prove that bisimilarity $\approx$ is a congruence for the multi-sorted LTS with states in $T_{(\Sigma_G,E)}$ specified using rules in Definition 8.6 instantiated with any SAM S satisfying the conditions for SAM compositionality.

Theorem 8.3. *Let us consider the LTS defined by rules in Definition 8.6 instantiated with a SAM S satisfying the conditions for SAM compositionality. Let t_1, t_2 be terms of the same sort in $T_{(\Sigma_G,E)}$ such that $t_1 \approx t_2$ and let $c[-]$ be a context such that $c[t_1]$ is defined. Then $c[t_1] \approx c[t_2]$.*

Proof. Thanks to Proposition 1.1 the LTS obtained without structural axioms can be represented as a coalgebra h for a functor $P_L : \mathcal{SET} \to \mathcal{SET}$. Using Proposition 1.2 P_L can be lifted to a functor $P_\Delta : \mathcal{Alg}(\Sigma_G) \to \mathcal{Alg}(\Sigma_G)$. Thanks to Proposition 1.3 the coalgebra h can be lifted too, when structural axioms are not present, because the rules are in De Simone format thanks to Lemma 8.6. Now we can apply two times Theorem 8.2, using Lemma 8.8 and Lemma 8.9 to add the axioms. This proves that the LTS has a bialgebraic structure, thus bisimilarity is a congruence thanks to Theorem 1.1. □

One may want to derive congruence results for a single-sorted algebra, where graphs with different sets of free names can be compared. In particular, we want not to distinguish graphs that differ only on the set of their free names, but not on their behaviour. The algebra used so far can also be seen as a partial single-sorted algebra on the unique carrier $T_{(\Sigma_G,E)}$. For instance, $||$ can be seen as a partial operator (and no more as a family of operators) $|| : T_{(\Sigma_G,E)} \times T_{(\Sigma_G,E)} \to T_{(\Sigma_G,E)}$ which is defined only if the two arguments have disjoint sets of names.

Even if we do not consider the sort of a term explicitly, its set of free names can be easily derived from the observation, since it is the domain of function Λ. To avoid this we allow corresponding labels to have different domain, provided that on the nodes appearing in just one of them only initial actions with fresh names (and all the existing ones) can be performed. Notice that this is the behaviour of isolated nodes.

Notice that the set of free names of a term can be deduced also in another way: when a new name is generated, the allowed choices for it are all the names which are not free. Thus the set of free names is the set of names that can not be used as extruded names. To avoid to observe the set of free names, a new definition of bisimilarity can be used.

Definition 8.8 (Bisimilarity for single-sorted SHR).
The bisimilarity for single-sorted SHR *based on SAM S is the largest relation $\approx_{S(S)}$ on $T_{(\Sigma_G,E)}$ such that $t_1 \approx_{S(S)} t_2$ implies:*

- *for each $t_1 \xrightarrow{\Lambda_1,\pi} t_1'$ and each Λ_1' such that $\mathrm{dom}(\Lambda_1') \supseteq \mathrm{sort}(t_1) \cup \mathrm{sort}(t_2)$, $\Lambda_1'(x) = \Lambda_1(x)$ if $x \in \mathrm{dom}(\Lambda_1) \cap \mathrm{dom}(\Lambda_1')$ and $\mathrm{act}_{\Lambda_1'}(x) \in \mathrm{fInit}(S)$ and $\mathrm{n}(\Lambda_1'(x))$ is a vector of new names of length $\mathrm{ar}(\mathrm{act}_{\Lambda_1'}(x))$ if $x \in \mathrm{dom}(\Lambda_1') \setminus \mathrm{dom}(\Lambda_1)$ satisfying also $(\mathrm{n}(\Lambda_1') \setminus \mathrm{fn}(t_1)) \cap \mathrm{fn}(t_2) = \emptyset$ there is some $t_2 \xrightarrow{\Lambda_2,\pi} t_2'$ such that $t_1' \approx_{S(S)} t_2'$ and $\Lambda_2(x) = \Lambda_1'(x)$ if $x \in \mathrm{dom}(\Lambda_2) \cap \mathrm{dom}(\Lambda_1')$, $\mathrm{act}_{\Lambda_2}(x) \in \mathrm{fInit}(S)$ and $\mathrm{n}(\Lambda_2(x))$ is a vector of new names if $x \in \mathrm{dom}(\Lambda_2) \setminus \mathrm{dom}(\Lambda_1')$;*
- *vice versa.*

Note that the additional conditions are trivially true for terms of the same sort that are bisimilar according to standard bisimilarity. Thus for terms of the same sort we can reuse the theory developed so far, and in particular Theorem 8.3.

The condition $(\mathrm{n}(\Lambda_1') \setminus \mathrm{fn}(t_1)) \cap \mathrm{fn}(t_2) = \emptyset$ extends the usual condition for name passing calculi with restriction (see, e.g., Definition 1.16), which says that $\mathrm{fn}(t_2)$ must not contain any extruded name, to a case where new names are not created only by extrusions. We have no need to apply renaming π, which corresponds to σ in Definition 1.16, since it is directly applied in the LTS.

This definition allows to quotient graphs distinguished by standard bisimilarity. For instance, if ϵ is the only initial action then subgraphs allowing only idle transitions are not observed. In particular, isolated nodes are not observed (even if there are non ϵ initial actions).

We can now prove a new congruence result exploiting the new definition.

Theorem 8.4. *Let S be a SAM satisfying the conditions for SAM compositionality, let t_1, t_2 be terms in $T_{(\Sigma_G,E)}$ such that $t_1 \approx_{S(S)} t_2$ and let $c[-]$ be a context such that $c[t_1]$ and $c[t_2]$ are both defined. Then $c[t_1] \approx_{S(S)} c[t_2]$.*

Proof. For each context c there exist two sets of names N_1 and N_2 such that $c[t]$ is defined iff $\mathrm{sort}(t) \cap N_1 = \emptyset$ and $N_2 \subseteq \mathrm{sort}(t)$. Thus if $c[t_1]$ and $c[t_2]$ are both defined, then $c[t]$ is also defined for each t such that $\mathrm{sort}(t) = \mathrm{sort}(t_1) \cup \mathrm{sort}(t_2)$. In particular, it is defined for $t_1' = !_{\mathrm{sort}(t_2)\setminus\mathrm{sort}(t_1)}\, t_1$ and $t_2' = !_{\mathrm{sort}(t_1)\setminus\mathrm{sort}(t_2)}\, t_2$.

We easily have that $c[t_1'] \approx_{S(S)} c[t_1]$ and $c[t_2'] \approx_{S(S)} c[t_2]$. Also, t_1' and t_2' have the same sort, thus thanks to Theorem 8.3 $c[t_1'] \approx c[t_2']$. This implies $c[t_1'] \approx_{S(S)} c[t_2']$, thus the thesis follows by transitivity. □

We claim that these congruence results are due to the concurrent flavour of SHR semantics, that allows many actions to be performed at the same time on different nodes, and this allows to distinguish between parallelism and nondeterminism. In fact, if we force interleaving in the semantics, as done for Milner synchronization in Section 2.2, then the property is lost. We will discuss this issue more in detail in Section 9.1, for Fusion Calculus, where examples can be stated in a more concise way. The translation of the examples therein into SHR produces examples also for this framework.

We present here an example where single-sorted bisimilarity is used to study the concurrency properties of Milner amoeboids.

Example 8.5. *We consider M amoeboids as defined in Definition 3.19.*

As said in Section 3.4, different M amoeboids with the same interface have different concurrency properties, since they can allow different sets of concurrent synchronizations. As a comparison, all Hoare amoeboids with the same interface instead allow the same behaviours. This explains why Theorem 3.8 can not be extended to prove a result similar to the one of Theorem 3.7.

We give here a full characterization of the concurrency properties of M amoeboids. A M amoeboid is fully characterized by its connection relation C, which is a set of sets of unordered pairs of names in its interface, containing the set c iff there exist disjoint paths inside the amoeboid connecting the two nodes in each pair in c.

For instance, the connection relation of the amoeboid in Example 8.2 (which is also the left amoeboid in Figure 8.1) contains all the singletons, plus the set of pairs $\{(x,y),(z,w)\}$.

Before proving this characterization, we present a result that describes the set of allowed transitions of a M amoeboid, refining Lemma 3.6. We need also some notation.

Notation. *We denote with $||_i t_i$ the composition using operator $||$ of terms t_i for each i.*

We can now prove the lemma.

Lemma 8.10. *Let M_1 be a M amoeboid with connection relation C. All the transitions for M_1 are of the form:*

$$M_1 \xrightarrow{\Lambda,\mathrm{id}} M_1||(||_i L_i)$$

where each L_i is a chain of L edges where only the two nodes at the extremities are free, i.e.:

$$[\![L_i]\!]^{\equiv}_G = x, x' \vdash \nu y_1, \ldots, y_n \; L(x, y_1)|L(y_1, y_2)|\ldots|L(y_n, x')$$

for some nodes x and x'.

In particular, there is one such transition for each choice of:

- *an element $c \in C$, determining which elements of the interface are synchronized;*
- *an action $\mathrm{act}_\Lambda(x)$ for each node x appearing in c with the compatibility condition that $(x, x') \in c \Rightarrow \mathrm{act}_\Lambda(x) = \overline{\mathrm{act}_\Lambda(x')}$, determining the action used on each connection;*
- *a vector $\mathrm{n}_\Lambda(x)$ of length $\mathrm{ar}(\mathrm{act}_\Lambda(x))$ of fresh names for each node x appearing in c, determining the choice of the new names.*

In this case Λ is defined on the nodes appearing in c as specified by the above conditions, while $\Lambda(x) = (\epsilon, \langle\rangle)$ otherwise. Furthermore we have a chain L_i connecting nodes y and z iff y and z are in the same position in $\mathrm{n}_\Lambda(x)$ and $\mathrm{n}_\Lambda(x')$ for $(x, x') \in c$.

Proof. The choice of c determines which elements of the interface are used for synchronization, and the connections between them. For each pair, a path inside the amoeboid is used by the transition. By hypothesis disjoint paths can be chosen. Also, no transition using non disjoint paths can be derived. Notice that the choice of the actual path influences just the length of the chains L_i, since internal actions are hidden. Each path acts independently. Thus the allowed transitions are determined by the choices for the actions performed on the interface of each path. The names transmitted on the interface matter too, while the ones used on internal nodes are not important since they will be bound. □

For instance, the transition derived in Example 8.2 corresponds to choose $c = \{(x, w)\}$ with action a on x with parameter y_1 and action $\overline{a}$ on w with parameter z_2. Chain $?_{y_2} \nabla_{y_2,z_1}(L_{y_1,y_2}||L_{z_1,z_2})$ is created to connect y_1 and z_2.

We can now prove the above claimed abstract characterization.

Theorem 8.5. *Let M_1 and M_2 be two M amoeboids, let Mil be a Milner SAM (with at least an input action), then $M_1 \approx_{S(Mil)} M_2$ iff M_1 and M_2 have the same connection relation.*

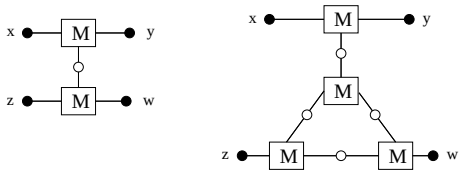

Figure 8.1: Two bisimilar M amoeboids.

Proof. Let us consider two M amoeboids M_1 and M_2 with the same connection relation C. We will prove that $M_1 \approx_{S(Mil)} M_2$ by coinduction. Suppose that $M_1 \xrightarrow{\Lambda,\pi} M_1'$. Thanks to Lemma 8.10 we have $\pi = \mathrm{id}$ and Λ and M_1' are of the form determined by the choices described in Lemma 8.10. In particular, $M_1' = M_1||(||_i L_i)$. Since M_1 and M_2 have the same connection relation then we also have:

$$M_2 \xrightarrow{\Lambda,\pi} M_2||(||_i L_i')$$

thanks to Lemma 8.10 because we can make the same choices. Since, for each i, $L_i \approx_{S(Mil)} L_i'$ and since bisimilarity is a congruence, then $M_1 \approx_{S(Mil)} M_2$.

The reverse implication can be proved by contradiction. Suppose to have two amoeboids M_1 and M_2 such that $M_1 \approx_{S(Mil)} M_2$. If M_1 and M_2 have different connection relations C_1 and C_2, then we choose a c that belongs to exactly one of the connection relations C_1 and C_2, and each transition using c can not be mimicked by the other amoeboid. □

Notice that the congruence property has been useful to simplify the above proof.

Two bisimilar M amoeboids are shown in Figure 8.1. By using multi-sorted bisimilarity instead of single-sorted one, also the set of free nodes becomes important: adding an isolated node produces a non bisimilar graph.

Chapter 9

Observational properties of process calculi

In this chapter we continue to analyze the observational properties of the models we have presented, but we move to the process calculi side. In particular, we concentrate on Fusion Calculus and on PRISMA Calculus.

For Fusion Calculus, we consider the concurrent semantics proposed in Section 2.3, and we prove that it is a congruence w.r.t. all the operators in the calculus. This is proved by exploiting the relation with SHR, and the congruence results for SHR presented in previous chapter. The result for Fusion Calculus is interesting since the bisimilarity relation that emerges from the standard semantics of Fusion Calculus is not a congruence, and hyperbisimilarity (see Definition 1.17) has to be used to get a compositional equivalence (see [Vic98]).

For PRISMA Calculus, first of all we analyze some semantic issues, giving an observational description of some of its properties. After that, we extend the congruence result for Fusion Calculus in Theorem 1.4 (see also [Vic98]), stating that hyperbisimilarity is a congruence, to deal with PRISMA Calculus. The generalization is non trivial, since the result is shown to hold for PRISMA Calculus on any SAM.

9.1 Concurrent semantics for Fusion Calculus is compositional

In this section we prove that the concurrent semantics for Fusion Calculus is a congruence w.r.t. all the operators in the calculus and w.r.t. renamings, we analyze some variants of the semantics, and we analyze the relationships with the standard semantics from an observational point of view.

Since the concurrent semantics for Fusion Calculus has the same labels as the SHR semantics, we can use single-sorted bisimilarity from Definition 8.8 also on Fusion processes. This semantics inherits the properties of the corresponding SHR

semantics, and thus it is a congruence w.r.t. the operators of term composition (recursion is not considered as an operator of term composition).

Theorem 9.1. *The single-sorted bisimilarity $\approx_{S(Milner_{IN})}$ is a congruence for the concurrent LTS of Fusion Calculus defined in Definition 2.6 w.r.t. all the operators of process composition and w.r.t. renamings.*

Proof. For prefix and summation the proof is trivial. As far as parallel composition, hiding and renamings are concerned, this is a consequence of the corresponding property of SHR (since SAM $Milner_{IN}$ satisfies the conditions for SAM compositionality).

Note first of all that the choice of Γ in the mapping is not important, since all the possible terms obtained as result of the translation are bisimilar. Also, all the names required to make the application of the context well-typed can be added. Let us consider the required contexts.

The context for hiding of x is $?_x \bullet$, and it is always well-typed thanks to the observation above. Let us consider renamings. Each renaming can be decomposed as the composition of an idempotent renaming and a bijective renaming. Idempotent renamings can be realized by composing contexts $\nabla_{x,y}\bullet$, which are always well-typed thanks to the observation above. Bijective renamings are generated by swappings such as $\{x/y, y/x\}$, which can be realized by context $\nabla_{y,z}\nabla_{x,y}\,!_x\,\nabla_{z,x}\,!_z \bullet$ where z is a new name. This context is always well-typed too. Parallel composition is realized using a bijective renaming σ to map the names of a process to new names, applying then the $||$ operator and mapping back names using the inverse renaming σ^{-1}. This is done by the context $(\bullet\sigma||\bullet)\sigma^{-1}$, where renamings are realized as described above. Again, the context is well-typed. □

In particular, when concurrent semantics is used, bisimilarity and hyperbisimilarity coincide. Hyperbisimilarity is the compositional equivalence usually used for Fusion Calculus, but it requires to explicitly consider all the possible renamings, thus it is more difficult to decide whether two processes are equivalent than with bisimilarity.

We can now analyze the bisimilarity relation that emerges using the concurrent semantics, and compare it with the notions of bisimilarity and hyperbisimilarity derived from the standard interleaving semantics.

One may guess that concurrent bisimilarity is a refinement of hyperbisimilarity, but this is not the case since in concurrent bisimilarity extrusions are not observed. For instance, the two Fusion processes $(x)ux.0$ and $(x)ux.0 + uy.0$ are not bisimilar with the standard bisimilarity since the transition with label uy of the second process can not be matched by the first one. Instead, they are bisimilar with the concurrent semantics, since the corresponding transition with label $(u\ in_1\langle y\rangle)$, id can be matched by the one of the first process obtained by extruding name x and choosing y as new name through α-conversion.

However, in the multi-sorted setting the set E of names extruded by the transition $t_1 \xrightarrow{\Lambda,\pi} t_2$ can be easily deduced, since $E = \mathrm{sort}(t_2) \setminus \mathrm{sort}(t_1)$. Thus we can choose as new observation the label $(E)\Lambda, \pi$ without changing the theory, and in particular preserving the congruence result.

Definition 9.1 (Extrusion-aware bisimilarity).
The extrusion-aware bisimilarity $\approx_E$ *is the bisimilarity relation obtained from single-sorted bisimilarity* $\approx_{S(Milner_{IN})}$ *by adding the set* E *of extruded names to the labels, and requiring bisimilarity to preserve them.*

Theorem 9.2. *The extrusion-aware bisimilarity* $\approx_E$ *is a congruence for the concurrent LTS of Fusion Calculus w.r.t. all the operators of process composition and w.r.t. renamings.*

Proof. The proof is analogous to the proof of Theorem 9.1, since the added observation does not change the multi-sorted semantics, since here the information can be deduced from sorts. The following parts of the proof are analogous to the ones for single-sorted bisimilarity. □

Extrusion-aware bisimilarity is a congruence w.r.t. the operators of process composition and w.r.t. renamings and it is a refinement of standard hyperbisimilarity.

This is proved by the following theorem.

Theorem 9.3.
Let P_1 *and* P_2 *be Fusion processes. Then* $P_1 \approx_E P_2 \Rightarrow P_1 \sim_f P_2$.

Proof. We have that $P_1 \approx_E P_2 \Rightarrow P_1 \approx_f P_2$ from the property of preservation of interleaving transitions (Corollary 2.1) and from the observation that all the concurrent transitions have a different form. In addition to that, the bisimilarity relation $\approx_E$ is closed under renamings, thus the thesis follows. □

We can now compare this bisimilarity with the hyperbisimilarity obtained using the standard interleaving semantics. We consider the same aspects considered in Section 2.3:

concurrent vs interleaving: the concurrent nature of concurrent semantics indeed guarantees a more detailed observation. For instance,

$$uy.0|\overline{v}x.0 \approx_f uy.\overline{v}x.0 + \overline{v}x.uy.0$$

while the two processes are not bisimilar according to $\approx_E$. Indeed, hyperbisimilarity distinguishes the two processes too, since after applying renaming $\{u/v\}$ the first process can perform a fusion transition (with fusion $\{x = y\}$) to 0, while the second one can not. Notice however that concurrent bisimilarity is even more distinguishing than hyperbisimilarity, since:

$$uy.0|vx.0 \sim_f uy.vx.0 + vx.uy.0$$

The two processes instead are not equivalent from a concurrency point of view, since the former can perform a concurrent transition while the latter can not. Notice however that $uy.0|ux.0 \approx_E uy.ux.0 + ux.uy.0$ since the two actions are on the same channel and thus they must be performed in interleaving also according to the concurrent semantics.

located τ: this is another difference that makes concurrent bisimilarity more distinguishing than standard one, since, e.g., $!x.0|!\overline{x}.0$ and $(y)(y.0|\overline{y}.0)|!x.0|!\overline{x}.0$ are equivalent with the standard semantics since they can both perform only inputs and outputs on x, and fusion transitions with $\emptyset$ as fusion. However any bisimilarity relation based on a located τ allows to distinguish the first process, which can make only fusion transitions synchronizing on node x, from the second one, which is also able to perform a fusion transition on an hidden node. An infinite amount of possible synchronizations on x has to be allowed, to avoid that the two processes can be distinguished just by counting the number of synchronizations. Notice that located τ is necessary for the congruence result, e.g., when inserting a process in the context $x.0|\bullet$. In fact, x can be performed together with a τ from the inserted process iff the τ is not located at x.

renamings vs fusions: the choices of using either a renaming or a fusion to represent the equivalence relation in the label produce the same result from an observational point of view, since each representative can be chosen inside any equivalence class. Thus, using fusions, there is just one label, while, using renamings, there are many different labels, but they can all be deduced from the fusion one.

application of fusion renamings: the idea of applying the fusion renaming directly in the LTS allows to simplify the definition of bisimulation, however the two approaches are equivalent, thus this has no influence on the deduced equivalence relation.

free names: if free names are observed (using a different definition of bisimilarity, e.g., requiring the two functions Λ to be equal in Definition 9.2) we get a more detailed observation, since for instance it is possible to distinguish $(x)xy.0$ from 0.

idle transitions: idle transitions are not important from an abstract point of view (at least, if free names are not observed using them), since they are available for any process.

extrusions: as already said, extrusions must be observed to make concurrent bisimilarity at least as detailed as standard one.

other differences: differences that are only syntactic are clearly not important from an observational point of view.

The discussion above shows that the concurrent observational semantics of Fusion Calculus is strongly related to the standard one, the main differences being the ability to observe concurrent transitions and the node where a synchronization is performed. Both these differences are fundamental for the congruence result, and they correspond to the observation of interesting features of the system under analysis, namely its degree of parallelism and which ports of the system are used.

9.2 Observational properties of PRISMA Calculus

In this section we concentrate on PRISMA Calculus. We start by proving some properties of the observational semantics of PRISMA Calculus. In particular, we extend to PRISMA Calculus a known result for Fusion Calculus, namely that hyperbisimilarity is a congruence w.r.t. all the operators in the calculus. The stronger result of bisimilarity to be a congruence can not hold, since it is not valid in the case of Fusion Calculus, which corresponds to PRISMA Calculus on Milner SAM. After that we exploit observational semantics to relate instances of PRISMA Calculus based on different SAMs.

We need a particular definition of bisimilarity for PRISMA, to deal with the different kinds of allowed labels. The intuition behind it is however the same behind Definition 1.16 for Fusion Calculus. Just note that also labels of the form $\neg x$ must be simulated, otherwise the resulting semantics is not compositional.

Definition 9.2 (Bisimilarity for PRISMA Calculus).
A bisimulation for PRISMA Calculus *on SAM S is a relation $\mathcal{R}_S$ on PRISMA processes such that $P\ \mathcal{R}_S\ Q$ implies:*

- $P \xrightarrow{\neg x} P \Rightarrow Q \xrightarrow{\neg x} Q$,
- $P \xrightarrow{\surd,\pi} P' \Rightarrow Q \xrightarrow{\surd,\pi} Q' \wedge P'\ \mathcal{R}_S\ Q'$,
- $P \xrightarrow{(Y)xa\vec{y},\pi} P' \wedge Y \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{(Y)xa\vec{y},\pi} Q' \wedge P'\ \mathcal{R}_S\ Q'$

and vice versa, where all the transitions are derived using SAM S. Bisimilarity $\approx_S$ is the maximal bisimulation.

We can use for hyperbisimilarity the standard definition (see Definition 1.17).

Notation. *We denote hyperbisimilarity for PRISMA Calculus on SAM S as $\sim_S$. If $P \sim_S Q$, we say that P and Q are hyperbisimilar.*

We present now some properties of hyperbisimilarity for PRISMA processes. Note that these can be stated once and they apply to any SAM S, or to any SAM S that satisfies some requirements (see, e.g., Lemma 9.2). Next lemma, in particular, shows that hyperbisimilarity abstracts from certain syntactic features of processes which are intuitively not important from an observational point of view.

Lemma 9.1. *For any SAM S and any PRISMA processes P, Q and R the following hyperbisimilarity relations hold.*

$$P|Q \sim_S Q|P \quad (P|Q)|R \sim_S P|(Q|R) \quad P|0 \sim_S P$$
$$P + Q \sim_S Q + P \quad (P + Q) + R \sim_S P + (Q + R) \quad P + 0 \sim_S P$$
$$P + P \sim_S P \quad (x)(y)P \sim_S (y)(x)P$$
$$(x)P|Q \sim_S P|(x)Q \text{ if } x \notin \mathrm{fn}(P) \quad (x)P \sim_S (y)P\{y/x\} \text{ if } y \notin \mathrm{fn}(P)$$

Proof. The proofs are similar to some of the proofs of Lemma 8.8 and Lemma 8.9, and some of them are quite technical. Thus we give only some intuitions on them. Note that, when referring to rule n, we mean rule n in Table 7.2.

$P|Q \sim_S Q|P$) The proof is based on the fact that rule 7.3 is self-symmetric because of commutativity of action synchronization composition, and that for rules 7.4 and 7.5 symmetric versions exist.

$(P|Q)|R \sim_S P|(Q|R)$) As far as rule 7.3 is concerned, the thesis follows from associativity of SAMs, while for rules 7.4 and 7.5 the proof is trivial. Notice that also the case of application of rule 7.3 for one parallel composition operator and rule 7.4 for the other one must be considered.

$P|0 \sim_S P$) The thesis follows from the fact that the only transitions with source 0 have labels either of the form $\neg x$ (and in this case rule 7.4 can be used) or of the form $x\epsilon\langle\rangle, \mathrm{id}$ (and in this case rule 7.3 can be used, thanks to the conditions in SAM definition concerning action ϵ).

$P + Q \sim_S Q + P$) The thesis follows since a symmetric version of rule 7.10 exists.

$(P + Q) + R \sim_S P + (Q + R)$) The thesis follows since projection is associative.

$P + 0 \sim_S P$) The thesis follows since the only actions that process 0 can perform are not able to force a choice of a nondeterministic sum to be taken.

$P + P \sim_S P$) The thesis follows because a symmetric version of rule 7.10 exists.

$(x)(y)P \sim_S (y)(x)P$) The thesis follows from the independence of hidings of different names.

$(x)P|Q \sim_S P|(x)Q$) The thesis follows since, either the hidden name is independent w.r.t. the one on which synchronization is performed, or, because of the side condition of the relation, P performs on x just an action of the form $\neg x$ or $x\epsilon\langle\rangle, \mathrm{id}$.

$(x)P \sim_S (y)P\{y/x\}$) The thesis follows either for coinductive hypothesis since the name remains hidden, or since, if it is extruded by rule 7.7, it can be α-converted. □

The relation $(x)0 \sim_S 0$ holds instead only for PRISMA on some of the SAMs.

Lemma 9.2. *The hyperbisimilarity relation $(x)0 \sim_S 0$ holds iff $\epsilon \notin \mathrm{fFin}(S)$.*

Proof. The first implication holds since if $\epsilon \in \mathrm{fFin}(S)$ then rule 7.6 can be applied to the transition with label $x\epsilon\langle\rangle, \mathrm{id}$ to get a transition with label $(\surd, \mathrm{id})$ for the left process which can not be derived for the right one.

The other implication is trivial. □

Including ϵ in $\mathrm{fFin}(S)$ corresponds to observe internal idle steps as $(\surd, \mathrm{id})$ labels. As observed when describing PRISMA (see Section 7.1), if this feature is not desired, the precondition can be easily enforced without changing any other aspect of the semantics.

Next theorem proves that the observational semantics is compositional. This result is fundamental to compute the observational semantics of large complex systems from the observational semantics of their components. It extends in a non trivial way an analogous result for Fusion Calculus [PV98] (see also Theorem 1.4): the interesting point is that it holds for PRISMA on any SAM.

Before proving it we present a lemma that is used in the proof of the case dealing with parallel composition.

Lemma 9.3. *If $P \xrightarrow{(Y)xa\vec{y},\pi} P'$ is a PRISMA transition with $y \in Y$, then there is also a PRISMA transition $P \xrightarrow{(Y\{y'/y\})xa\vec{y}\{y'/y\},\pi} P'\{y'/y\}$ for each $y' \notin \mathrm{fn}(P') \cup \{x\} \cup \mathrm{n}(\vec{y})$.*

Proof. The proof is by rule induction on the derivation of the transition. □

We can now prove the claim above.

Theorem 9.4. *For any SAM S hyperbisimilarity $\sim_S$ is a congruence w.r.t. all the operators in PRISMA.*

Proof. For each unary (resp. binary) operator op, we have to prove that for each SAM S and for each P_1, P_2, Q_1, Q_2 processes such that $P_1 \sim_S Q_1$ and $P_2 \sim_S Q_2$ we have $op(P_1) \sim_S op(Q_1)$ (resp. $op(P_1, P_2) \sim_S op(Q_1, Q_2)$). The proof is by rule induction on the derivation of the transition of $op(P_1)$ (resp. $op(P_1, P_2)$), and each step requires a coinductive proof. However, rule induction is needed just for replication, while in the other cases it is enough to consider each operator in isolation (in a suitable order). Thus we will use rule induction just for replication.

Notice that the labels of the form $\neg x$ allow exactly to identify the set of active names of a process, but by applying any operator to processes with the same sets of active names, we get again processes with the same sets of active names. Thus in the following discussion we will not consider explicitly the labels of the form $\neg x$.

The first cases of the proof are fully discussed to show the general idea, the last ones are less detailed. Note that, when referring to rule n, we mean rule n in Table 7.2.

Case prefix) We have to prove that for each SAM S, each prefix $xa\vec{y}$ and each pair of processes P and Q such that $P \sim_S Q$ we also have $xa\vec{y}.P \sim_S xa\vec{y}.Q$. Thus we have to prove that, for each renaming σ, $(xa\vec{y}.P)\sigma$ and $(xa\vec{y}.Q)\sigma$ can perform the same transitions, going into hyperbisimilar processes. The only allowed transitions are the ones from rule 7.2, which are to the process itself and which thus verify the thesis (by coinductive hypothesis), and the ones from rule 7.1, which have the same label as required and which lead to processes $P\sigma$ and $Q\sigma$ which are hyperbisimilar by hypothesis. In general, we have not to consider explicitly the cases of all the

renamings σ, since we can always choose $P' = P\sigma$ and $Q' = Q\sigma$ and reason without an explicit σ. Also, we have not to reason explicitly about transitions labeled by $x\epsilon\langle\rangle, \mathrm{id}$, since they can always be trivially simulated.

Case $+$) Suppose that $P_1 \sim_S Q_1$ and $P_2 \sim_S Q_2$. We have to prove that $P_1 + P_2 \sim_S Q_1 + Q_2$. Each (non ϵ) transition $P_1 + P_2 \xrightarrow{\lambda} M$ is derived from rule 7.10 (or its symmetric, and the two cases are analogous), thus we have that $P_1 \xrightarrow{\lambda} M$. By hypothesis also $Q_1 \xrightarrow{\lambda} M'$ with $M \sim_S M'$. Since we can derive $Q_1 + Q_2 \xrightarrow{\lambda} M'$ the thesis follows.

Case (x)) Suppose that $P \sim_S Q$. We have to prove that $(x)P \sim_S (x)Q$. We have different cases according to which rule is used for deriving the transition. Notice that, in all the cases, the majority of the side conditions deal with the label and the bound name, thus they hold for P iff they hold for Q. Thus we can use the transition corresponding to the premise of the rule used to derive the transition for P to derive the transition for Q using the same rule. We have to check that: (1) the derived labels are the same and (2) the resulting processes are hyperbisimilar. For rule 7.8, the first part is straightforward, while the second follows by coinduction. The same happens for rule 7.6. As far as rule 7.7 is concerned, we have the condition that $z' \notin (\mathrm{fn}(P') \cup \{x\} \cup \mathrm{n}(\vec{y})) \setminus \{z\}$ which involves P', but notice that z' is an extruded name, thus for definition of bisimulation we have to consider just the cases where $z' \notin \mathrm{fn}(Q)$. Since z' is not an extruded name in the premise (because $z' \notin \mathrm{n}(\vec{y})$), then it is not a free name of Q' and the condition is satisfied also for the derivation for Q. Hyperbisimilarity of the resulting processes follows from the closure under renamings. The proof for rule 7.9 is straightforward.

Case $|$) Suppose that $P_1 \sim_S Q_1$ and $P_2 \sim_S Q_2$. We have to prove that this implies $P_1|P_2 \sim_S Q_1|Q_2$. We have three rules to check here. Let us consider rule 7.3 first. As before, most of the conditions deal only with the labels, thus they are verified for P_1 and P_2 iff they are verified for Q_1 and Q_2. The only condition to check is $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$. By definition of bisimulation we need to simulate only transitions such that $W \cap \mathrm{fn}(Q_1|Q_2) = \emptyset$. Thus the only names that can cause problems are the ones in $(Y_1 \cup Y_2) \setminus W$. These are extruded names that will be restricted in the resulting process. Thanks to Lemma 9.3, we have infinite choices for which names to use in the premise, thus we can find a choice that will not use names in $\mathrm{fn}(Q_1) \cup \mathrm{fn}(Q_2)$. Any choice is good, since these names are restricted afterward, and the α-conversion axiom bisimulates thanks to Lemma 9.1. We still have to prove that the two resulting processes, namely $(\vec{s})(P_1'|P_2')\sigma$ and $(\vec{s})(Q_1'|Q_2')\sigma$ are hyperbisimilar. For what said before, we can suppose that $\vec{s}$ is the same in both the cases. By hypothesis $P_1' \sim_S Q_1'$ and $P_2' \sim_S Q_2'$. By coinductive hypothesis, $P_1'|P_2' \sim_S Q_1'|Q_2'$. Thanks to the closure under renamings of hyperbisimilarity, $(P_1'|P_2')\sigma \sim_S (Q_1'|Q_2')\sigma$. Finally, for what already proved for restriction, $(\vec{s})(P_1'|P_2')\sigma \sim_S (\vec{s})(Q_1'|Q_2')\sigma$. The cases for rules 7.4 and 7.5 are simpler than the one just shown.

Case $!$) We have to prove that if $P \sim_S Q$, then $!P \sim_S !Q$. We have to use rule induction for that case. If $!P \xrightarrow{\lambda} P'$, then we also have $P|!P \xrightarrow{\lambda} P'$, which is a

premise. By inductive hypothesis on the context $\bullet|!\bullet$, $Q|!Q \xrightarrow{\lambda} Q'$ with $Q' \sim_S P'$. Since also $!Q$ has the same transition the thesis follows. □

We can prove that bisimilarity is not a congruence by considering the same counterexample used for Fusion Calculus, which, in PRISMA syntax, is:

$$x_1 a_1 \vec{y}_1.0 | x_2 a_2 \vec{y}_2.0 \approx_S x_1 a_1 \vec{y}_1.x_2 a_2 \vec{y}_2.0 + x_2 a_2 \vec{y}_2.x_1 a_1 \vec{y}_1.0$$

The two processes are bisimilar for each SAM S, since they can just execute the actions a_1 and a_2 in one of the two possible orders, and no synchronization is possible (there are however the usual transitions to the process itself). If we apply renaming $\{x_1/x_2\}$ to the processes, and a_1 and a_2 have been chosen in such a way that an action synchronization of the form $(a_1, a_2, (c, \mathrm{Mob}, \doteq))$ belongs to $\mathrm{fActSyn}(S)$ then the first member can synchronize, while the second one can not. Notice that this counterexample can be used for any (non trivial) SAM S. To solve this problem a concurrent semantics in the style of Section 2.3 can be used, but this makes the LTS more complex, thus we leave this development for future work.

We can exploit hyperbisimilarity to justify some of the categorical constructions in Section 5.3, together with the translation of PRISMA processes along morphisms in Definition 7.3. The first proposition shows that translations along isomorphisms preserve hyperbisimilarity.

Proposition 9.1.
Let P and Q be two PRISMA processes and $H(P)$ and $H(Q)$ be their translations according to SAM isomorphism $H : S_1 \to S_2$. Then $P \sim_{S_1} Q$ iff $H(P) \sim_{S_2} H(Q)$.

Proof. The thesis follows from Proposition 7.1. □

Note that if H is just a morphism hyperbisimilarity is neither preserved nor reflected. In fact, a morphism H can collapse two actions that have the same allowed synchronizations but which produce different observations, thus making two non bisimilar processes equivalent (or even equal). Vice versa let us consider a SAM S where actions a_1 and a_2 can not synchronize. We have:

$$x a_1 \vec{y}_1.0 | x a_2 \vec{y}_2.0 \approx_S x a_1 \vec{y}_1.x a_2 \vec{y}_2.0 + x a_2 \vec{y}_2.x a_1 \vec{y}_1.0$$

However a morphism can add an action synchronization for this pair of actions, making the two processes no more equivalent.

However, in some cases bisimilarity is preserved. This allows for instance to justify the coproduct construction, showing essentially that adding other actions, which are not used by the translated process, does not change its behaviour.

Proposition 9.2. *Let P and Q be two PRISMA processes and $H(P)$ and $H(Q)$ be their translations according to SAM injection $H : S_1 \to S_1 + S_2$. If $\epsilon \in \mathrm{fFin}(S_1) \Rightarrow \epsilon \in \mathrm{fFin}(S_2)$ then $P \sim_{S_1} Q$ iff $H(P) \sim_{S_1+S_2} H(Q)$.*

Proof. The proof is by induction on the derivation of the transition. □

A similar result can be shown for the product construction in $\mathcal{ASYNC}$, when ϵ is chosen as other action of the pair.

Proposition 9.3. *Let P and Q be two PRISMA processes and $H(P)$ and $H(Q)$ be their translations according to SAM morphism $H : S_1 \to S_1 \otimes S_2$ which maps a to (a, ϵ_2). If $\epsilon_2 \in \mathrm{fFin}(S_2)$ and $(\epsilon_2, \epsilon_2, (\epsilon_2, MP_{0,0}, EQ_0)) \in \mathrm{fActSyn}(S_2)$ then $P \sim_{S_1} Q$ iff $H(P) \sim_{S_1+S_2} H(Q)$.*

Proof. The proof is by induction on the derivation of the transition. □

Intuitively the above proposition says that if only one element of the pair is actually used, then the semantics is unchanged.

We have thus concluded our analysis of the observational behaviour of PRISMA Calculus, showing that the translation of processes using suitable morphisms preserves the observational properties, but morphisms that affect the semantics in more complex ways can be defined too.

Chapter 10

Conclusions and future work

10.1 Conclusions

In our understanding, the work presented so far is built on two main facts.

The first one is the strong relationship between process calculi, and in particular Fusion Calculus, on one side and graph transformation, and in particular SHR, on the other side. The results presented in Chapter 2 highlight this relationship. Our results are not the only ones of this kind. This thread of research started in fact with [Mil79], with the aim of providing a visual representation for interacting systems. Further contributions have been given, e.g., by [Mil94, MP95, Gad03, GM05, HM01, FMT01, JM03, LPV01]. However our approach has some peculiarities, due mainly to the careful choice of two strongly related models, whose comparison allows a fruitful exchange of results, techniques and ideas between the two worlds. In particular, while the mapping is from process calculi to graph transformation, the main flow of contributions is in the opposite direction. Many results of this kind, such as the concurrent semantics of Fusion Calculus (Section 2.3), the PRISMA Calculus (Section 7.1) and the congruence result for Fusion Calculus (Section 9.1) have been presented here, and others are left for future work. Many approaches such as [Mil79, Mil94, LPV01] can not obtain this kind of results, since they rely on graphical formalisms developed ad-hoc. Other approaches are limited in this direction by the large distance between the two compared frameworks. Our case is different, since SHR is quite similar to process calculi, featuring in particular a SOS semantics which includes synchronization and mobility aspects. The same is true for [HM01], that uses a different version of SHR, but that work has not been exploited so far to obtain additional results in the process calculi field, but just to show an evidence of the expressive power of SHR.

The relation between SHR and Logic Programming has been explored with a similar aim, and Logic Programming with restriction has emerged as a first result. Further results are left for future work, since the distance between Logic Programming and SHR is larger than the one between Fusion Calculus and SHR, and this

makes exchanges of results more difficult to work out. Some ideas are discussed in the next section.

The second main point in our work is the introduction of SAMs. We think that this is an important point, since they allow to extend the theory of SAs to calculi with name mobility. The derived frameworks (parametric SHR, SHR-HS, PRISMA Calculus) allow to simply model many different systems, and the key aspect is that SAMs can be used to code inside the model the needed interaction patterns. This part of the model is kept distinct w.r.t. the modeling of system structure and behaviour using processes or graphs (together with productions), and this separation of concerns allows to simplify the modeling step. This idea is partly shared with process frameworks [Gar00], but we use a LTS approach instead of a reduction approach. It is important to notice that these parametric frameworks allow to develop theory and tools in a general way, so that they can be applied in different contexts. Also, SAMs highlight the role of synchronization, thus allowing to reason on it as a separate dimension in the "space of models". When synchronization patterns have been recognized as first-class entities, they can be compared (e.g., using morphisms), combined, and used together (e.g., as in SHR-HS).

In addition to this main points, we recall the congruence results in chapters 8 and 9. The one on SHR is important since it deals with a large class of graph transformations, and since this kind of result is not common in that field (here again, the fact that SHR has some process calculi features has been exploited). The one for Fusion Calculus instead is interesting since it does not hold for the standard Fusion semantics, and it suggests that a concurrent semantics is more desirable than an interleaving one.

10.2 Plans for future work

Many of the issues discussed so far require further investigations, and some related aspects are interesting too.

Starting from the part dealing with mappings, the relation between SHR and Logic Programming should allow new developments inside Logic Programming itself. We have shown, in fact, that Logic Programming is related to Hoare SHR, and the Hoare synchronization mechanism is related to unification. We think that variations of Logic Programming that correspond to other synchronization models may be developed. These variations should exploit a different unification algorithm, and they should allow to write programs using a different paradigm. We think that the main feature of Hoare synchronization that makes its unification algorithm simple is that Hoare synchronization is idempotent, thus neither the number of occurrences of a variable in the goal nor the order in which unification steps are performed are important. Further work is required to devise how to use other synchronization models. Dually, HSHR can be extended to match some more features of Logic Programming. See the discussion at the end of Section 4.3 for some insights on that

topic.

As far as the comparison between Hoare and Milner synchronization is concerned, two extensions are possible. On one side, the introduction of SAMs should allow some general results relating the properties of the SAM with the possibility of implementing it using other SAMs. It should be interesting to find whether a SAM able to implement all the others exists or not. Another line of research concerns different forms of expressiveness, which are related to observational semantics instead of reachability in the reduction system. In this direction we should look in particular for translations from models on one SAM to models on another that preserve the bisimilarity relation. The difficult points here are that infinite behaviours should be considered (it is always possible to simulate finite LTSs using graphs of just one edge with suitable productions), and that dealing with free names in a distributed setting is difficult since many different parties should agree on their meaning.

On the side of SAMs, recent works [HT05] have shown a relation between SAMs and c-semirings [BMR97], allowing to combine synchronization issues with quality of service issues. It would be interesting to formally analyze the relation, in order to see how a SAM can be extended (hopefully in a standard way) to a c-semiring. Also, an extension of SHR where different transitions are chosen according to their quality of service properties should be developed (in [HT05], quality of service measures can only enable or disable transitions).

Moving towards the process calculi side, an extension of PRISMA Calculus in the direction traced by SHR-HS should be interesting too, however we think that it is quite straightforward.

Going to the observational semantics aspects, an axiomatization of PRISMA processes that captures the notion of hyperbisimilarity should be investigated. Here the main point is to analyze how the properties of the SAM lift to properties of processes on that SAM. A similar work, but simpler since the scenario is not parametric, is required also for the concurrent semantics of Fusion Calculus.

As far as congruence results are concerned, we remember that the results on parametric SHR and on Fusion Calculus rely on the theory of bialgebras and, in particular, on the condition that all the structural axioms bisimulate. We think that this condition is too restrictive. In fact, its intuitive meaning is that structural axioms simply capture properties of the LTS, that is equivalent terms should have the same transitions. This means that all the derivations can be done without using structural congruence. This eliminates the main advantage of structural congruence, that is the possibility to simplify the rules by avoiding to explicitly deal with some cases which can be obtained for free using structural congruence. Consider, e.g., α-conversion of extruded names as in rule 7.7, or closure of a name during synchronization as in rule 7.3 in the operational semantics of PRISMA (Figure 7.2). The same issues are hidden in the semantics of Fusion Calculus, since it exploits structural congruence instead. It would be interesting to find general techniques with less strict application conditions to allow to simplify the LTSs. It is interesting to note that, in the proof of the congruence properties, applications of axioms inside

processes do not cause problems, while applications of axioms transforming pieces of the process together with pieces of the context break compositionality.

Another interesting point is to find a concurrent LTS like the one presented for Fusion Calculus also for π-calculus and for PRISMA Calculus, and to see whether the corresponding bisimilarity is a congruence or not. We guess that the answer is positive. To this aim, inspiration should be taken from SHR presented in [HM01] and from parametric SHR respectively.

Finally, we think that our work shows that SHR has many interesting features as a model for GC systems, since it combines in a coherent framework a graphical representation, interaction and mobility. Thus it is important to analyze its relationships with other approaches to graph transformation which are used in the same field. We refer in particular to double pushout [EPS73] and bigraphs [Mil01].

In the first case we think that SHR should be used to break the application of a double pushout rule into smaller parts, enabling a distributed application. However, many different such breakings can be found in general, and finding a suitable canonical one looks hard. We think however that techniques similar to the ones used to find a LTS corresponding to a given rewriting system [LM00] should be used. However, the problem of distributing the label among the different connections remains.

In the case of bigraphs, the same approach should be analyzed, but a dual approach is also interesting. In fact, the link structure of bigraphs is essentially analogous to the structure of SHR hypergraphs. However bigraphs also use a place structure. Thus an extension of SHR to bigraphs should consider SHR style synchronization on the links, together with some other kind of interaction on the place structure. A first try can use simple unlabeled rewrite rules, while a more refined approach may use Ambient [CG98] style capabilities. The second approach looks more promising, since it may allow interactions between the two different dimensions. Another approach could be to code bigraphs into hypergraphs and consider the derived behaviour, but this may be not natural when observed directly in the bigraphical framework.

Apart from these possible next steps, we remember that the general aim of finding a categorization of models for GC systems, if not a unique model, is still far away, even if we think that we have contributed with some small steps in this direction.

Bibliography

[AB05] L. Acciai and M. Boreale. XPi: A typed process calculus for XML messaging. In M. Steffen and G. Zavattaro, editors, *Proc. of FMOODS'05*, volume 3535 of *Lect. Notes in Comput. Sci.*, pages 47–66. Springer, 2005.

[ABB+02] L. F. Andrade, P. Baldan, H. Baumeister, R. Bruni, A. Corradini, R. De Nicola, J. L. Fiadeiro, F. Gadducci, S. Gnesi, P. Hoffman, N. Koch, P. Kosiuczenko, A. Lapadula, D. Latella, A. Lopes, M. Loreti, M. Massink, F. Mazzanti, U. Montanari, C. Oliveira, R. Pugliese, A. Tarlecki, M. Wermelinger, M. Wirsing, and A. Zawlocki. AGILE: Software architecture for mobility. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Proc. of WADT'02*, volume 2755 of *Lect. Notes in Comput. Sci.*, pages 1–33. Springer, 2002.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL'01*, pages 104–115. ACM Press, 2001.

[AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. of ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AGI] AGILE: Architectures for mobility. `http://www.pst.informatik.uni-muenchen.de/projekte/agile/`.

[AHS90] J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories*. Wiley, 1990.

[AM89] P. Aczel and N. P. Mendler. A final coalgebra theorem. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Proc. of CTCS'89*, volume 389 of *Lect. Notes in Comput. Sci.*, pages 357–365. Springer, 1989.

[BBM04] M. Boreale, M. G. Buscemi, and U. Montanari. D-fusion: A distinctive fusion calculus. In W.-N. Chin, editor, *Proc. of APLAS'04*, volume 3302 of *Lect. Notes in Comput. Sci.*, pages 296–310. Springer, 2004.

[BBM05] M. Boreale, M. G. Buscemi, and U. Montanari. A general name binding mechanism. In D. Sangiorgi and R. De Nicola, editors, *Proc. of TGC '05*, volume 3705 of *Lect. Notes in Comput. Sci.* Springer, 2005.

[BC87] M. Bauderon and B. Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(2–3):83–127, 1987.

[BC94] G. Boudol and I. Castellani. Flow models of distributed computations: Three equivalent semantics for ccs. *Inform. and Comput.*, 114(2):247–314, 1994.

[BCG82] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, 1982.

[BCG04] P. Baldan, A. Corradini, and F. Gadducci. Specifying and verifying UML activity diagrams via graph transformation. In C. Priami and P. Quaglia, editors, *Proc. of Global Computing 2004*, volume 3267 of *Lect. Notes in Comput. Sci.*, pages 18–33. Springer, 2004.

[BDPF98] L. Bettini, R. De Nicola, R. Pugliese, and G. L. Ferrari. Interactive mobile agents in X-Klaim. In *Proc. of WETICE'98*, pages 110–117. IEEE Computer Society Press, 1998.

[BFL+04] R. Bruni, J. L. Fiadeiro, I. Lanese, A. Lopes, and U. Montanari. New insights on architectural connectors. In J.-J. Levy, E. W. Mayr, and J. C. Mitchell, editors, *Proc. of IFIP TCS'04, 3rd IFIP International Conference on Theoretical Computer Science*, pages 367–379. Kluwer Academics, 2004.

[BG95] N. Busi and R. Gorrieri. A petri net semantics for pi-calculus. In I. Lee and S. A. Smolka, editors, *Proc. of CONCUR'95*, volume 962 of *Lect. Notes in Comput. Sci.*, pages 145–159. Springer, 1995.

[BGG+05] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: A synergic approach for system design. In B. Benatallah, F. Casati, and P. Traverso, editors, *Proc. of ICSOC'05*, volume 3826 of *Lect. Notes in Comput. Sci.*, pages 228–240. Springer, 2005.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[BKKW02] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing. Extending activity diagrams to model mobile systems. In M. Aksit, M. Mezini, and R. Unland, editors, *Proc. of NetObjectDays'02*, volume 2591 of *Lect. Notes in Comput. Sci.*, pages 278–293. Springer, 2002.

[BL04] R. Bruni and I. Lanese. On graph(ic) encodings. In B. Koenig, U. Montanari, and P. Gardner, editors, *Proceedings of Dagstuhl Seminar n.04241, Graph Transformations and Process Algebras for Modeling Distributed and Mobile Systems*, Electronic proceedings, pages 23–39, 2004.

[BLM05] R. Bruni, I. Lanese, and U. Montanari. Complete axioms for stateless connectors. In J. L. Fiadeiro, M. Harman, M. Roggenbach, and J. J. Rutten, editors, *Proc. of CALCO'05, First Conference on Algebra and Coalgebra in Computer Science*, volume 3629 of *Lect. Notes in Comput. Sci.*, pages 98–113. Springer, 2005.

[BM00] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inform. and Comput.*, 156(1-2):46–89, 2000.

[BM02] M. G. Buscemi and U. Montanari. A first order coalgebraic model of pi-calculus early observational equivalence. In L. Brim, P. Jancar, M. Kretínský, and A. Kucera, editors, *Proc. of CONCUR'02*, volume 2421 of *Lect. Notes in Comput. Sci.*, pages 449–465. Springer, 2002. Full version in Technical Report TR-02-14, Dipartimento di Informatica, Universitá di Pisa, 2002.

[BMR97] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[BMR01] R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming*, 1(6):647–690, 2001.

[CDM85] A. Corradini, P. Degano, and U. Montanari. Specifying highly concurrent data structure manipulation. In G. Bucci and G. Valle, editors, *Proc. of Computing 85: A Broad Perspective of Current Developments.* Elsevier Science, 1985.

[CEHW01] A. Corradini, H. Ehrig, R. Heckel, and U. Wolter. Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. *Applied Categorical Structures*, 9(1):83–110, 2001.

[CFM90] A. Corradini, G. L. Ferrari, and U. Montanari. Transition systems with algebraic structure as models of computations. In I. Guessarian, editor, *Proc. of Semantics of Systems of Concurrent Processes*, volume 469 of *Lect. Notes in Comput. Sci.*, pages 185–222. Springer, 1990.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proc. of FoSSaCS'98*, volume 1378 of *Lect. Notes in Comput. Sci.*, pages 140–155. Springer, 1998.

[CH04] A. Corradini and D. Hirsch. An operational semantics of CommUnity based on graph transformation systems. In *Proc. of GT-VMT 2004*, volume 109 of *Elect. Notes in Th. Comput. Sci.*, pages 111–124. Elsevier Science, 2004.

[CHM94] S. Christensen, Y. Hirshfeld, and F. Moller. Decidable subsets of ccs. *The Computer Journal*, 37(4):233–242, 1994.

[CL99] S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective-caml. In *Proc. of ASA/MA'99*, pages 22–29. IEEE Computer Society Press, 1999.

[CM83] I. Castellani and U. Montanari. Graph grammars for distributed systems. In H. Ehrig, M. Nagl, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science*, volume 153 of *Lect. Notes in Comput. Sci.*, pages 20–38. Springer, 1983.

[Dal00] S. Dal-Zilio. Mobile processes: A commented bibliography. In F. Cassez, C. Jard, B. Rozoy, and M. D. Ryan, editors, *Proc. of MOVEP'00*, volume 2067 of *Lect. Notes in Comput. Sci.*, pages 206–222. Springer, 2000.

[de 85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoret. Comput. Sci.*, 37(3):245–267, 1985.

[DFP98] R. De Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *Software Engineering*, 24(5):315–330, 1998.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoret. Comput. Sci.*, 34:83–133, 1984.

[DM87] P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM*, 34(2):411–449, 1987.

[EHKP91] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comput. Sci.*, 1(3):361–404, 1991.

[EKMR99] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.3: Concurrency, Parallellism, and Distribution.* World Scientific, 1999.

[EM99] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In G. Ciobanu and G. Paun, editors, *Proc. of FCT'99*, volume 1684 of *Lect. Notes in Comput. Sci.*, pages 258–268. Springer, 1999.

[Eng93] J. Engelfriet. A multiset semantics for the pi-calculus with replication. In E. Best, editor, *Proc. of CONCUR'93*, volume 715 of *Lect. Notes in Comput. Sci.*, pages 7–21. Springer, 1993.

[EPS73] H. Ehrig, M. Pfender, and H. J. Schneider. Graph grammars: an algebraic approach. In *Proc. of SWAT'73*, pages 167–180. IEEE Computer Society Press, 1973.

[FG96] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. of POPL'96*, pages 372–385. ACM Press, 1996.

[FM97] J. L. Fiadeiro and T. S. E. Maibaum. Categorical semantics of parallel program design. *Science of Computer Programming*, 28(2–3):111–138, 1997.

[FM00] G. L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Inform. and Comput.*, 156(1–2):173–235, 2000.

[FMT01] G. L. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In S. Ronchi Della Rocca A. Restivo and L. Roversi, editors, *Proc. of ICTCS'01*, volume 2202 of *Lect. Notes in Comput. Sci.*, pages 1–16. Springer, 2001.

[Gad03] F. Gadducci. Term graph rewriting for the π-calculus. In A. Ohori, editor, *Proc. of APLAS'03*, volume 2895 of *Lect. Notes in Comput. Sci.*, pages 37–54. Springer, 2003.

[Gar00] P. Gardner. From process calculi to process frameworks. In C. Palamidessi, editor, *Proc. of CONCUR'00*, volume 1877 of *Lect. Notes in Comput. Sci.*, pages 69–88. Springer, 2000.

[Gel85] D. Gelernter. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[Gloa] Global computing II initiative homepage. `http://www.cordis.lu/ist/fet/gc.htm`.

[Glob] Global computing initiative homepage. `http://www.cordis.lu/ist/fet/gc-5fp.htm`.

[GM00] F. Gadducci and U. Montanari. The tile model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[GM02] F. Gadducci and U. Montanari. Comparing logics for rewriting: rewriting logic, action calculi and tile logic. *Theoret. Comput. Sci.*, 285(2):319–358, 2002.

[GM05] F. Gadducci and U. Montanari. Graph processes with fusions: Concurrency by colimits, again. In H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lect. Notes in Comput. Sci.*, pages 84–100. Springer, 2005.

[GS89] H. Gaifman and E. Y. Shapiro. Fully abstract compositional semantics for logic programs. In *Proc. of POPL'89*, pages 134–142. ACM Press, 1989.

[GW00] P. Gardner and L. Wischik. Explicit fusions. In M. Nielsen and B. Rovan, editors, *Proc. of MFCS'00*, volume 1893 of *Lect. Notes in Comput. Sci.*, pages 373–382. Springer, 2000.

[GW04] P. Gardner and L. Wischik. Strong bisimulation for the explicit fusion calculus. In I. Walukiewicz, editor, *Proc. of FoSSaCS'04*, volume 2987 of *Lect. Notes in Comput. Sci.*, pages 484–498. Springer, 2004.

[HH05] J. He and C. A. R. Hoare. Linking theories of concurrency. In D. Van Hung and M. Wirsing, editors, *Proc. of ICTAC'05*, volume 3722 of *Lect. Notes in Comput. Sci.*, pages 303–317. Springer, 2005.

[HIM00] D. Hirsch, P. Inverardi, and U. Montanari. Reconfiguration of software architecture styles with name mobility. In A. Porto and G.-C. Roman, editors, *Proc. of Coordination '00*, volume 1906 of *Lect. Notes in Comput. Sci.*, 2000.

[Hir03] D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, U.B.A., 2003.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[HM01] D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In K. G. Larsen and M. Nielsen, editors, *Proc. of CONCUR'01*, volume 2154 of *Lect. Notes in Comput. Sci.* Springer, 2001.

[Hoa80] C. A. R. Hoare. A model for communicating sequential processes. In R. M. McKeag and A. M. Macnaghten, editors, *On the Construction of Programs*. Cambridge University Press, 1980.

[Hoa05] C. A. R. Hoare. Why ever CSP? In L. Aceto and A. D. Gordon, editors, *Algebraic Process Calculi: The First Twenty Five Years and Beyond*, number NS-05-3 in Notes Series, pages 141–146. BRICS, 2005.

[HP00] O. M. Herescu and C. Palamidessi. Probabilistic asynchronous pi-calculus. In J. Tiuryn, editor, *Proc. of FoSSaCS'00*, volume 1784 of *Lect. Notes in Comput. Sci.*, pages 146–160. Springer, 2000.

[HR98] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. of POPL'98*, pages 378–390. ACM Press, 1998.

[HT91] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proc. of ECOOP'91*, volume 512 of *Lect. Notes in Comput. Sci.*, pages 1–20. Springer, 1991.

[HT05] D. Hirsch and E. Tuosto. SHReQ: A framework for coordinating application level QoS. In B. Aichernig and B. Beckert, editors, *Proc. of SEFM'05*, pages 425–434. IEEE Computer Society Press, 2005.

[JM03] O. H. Jensen and R. Milner. Bigraphs and transitions. In *Proc. of POPL'03*, pages 38–49. ACM Press, 2003.

[KM01] B. König and U. Montanari. Observational equivalence for synchronized graph rewriting. In N. Kobayashi and B. C. Pierce, editors, *Proc. of TACS'01*, volume 2215 of *Lect. Notes in Comput. Sci.*, pages 145–164. Springer, 2001.

[Lab95] Intelligent Systems Laboratory. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, 1995.

[Lan02] I. Lanese. Process synchronization in distributed systems via Horn clauses. Master's thesis, University of Pisa, Computer Science Department, 2002. Downloadable from `http://www.di.unipi.it/~lanese/work/tesi.ps`.

[Le 98] D. Le Métayer. Describing software architecture styles using graph grammars. *IEEE Trans. Software Eng.*, 24(7):521–533, 1998.

[Llo93] J. W. Lloyd. *Foundations of Logic Programming, Second Extended Edition*. Springer, 1993.

[LM00] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Proc. of CONCUR'00*, volume 1877 of *Lect. Notes in Comput. Sci.*, pages 243–258. Springer, 2000.

[LM02] I. Lanese and U. Montanari. Software architectures, global computing and graph transformation via logic programming. In L. Ribeiro, editor, *Proc SBES'2002 - 16th Brazilian Symposium on Software Engineering*, pages 11–35. Anais, 2002.

[LM04a] I. Lanese and U. Montanari. A graphical fusion calculus. In F. Honsell, M. Lenisa, and M. Miculan, editors, *Proceedings of the Workshop of the COMETA Project on Computational Metamodels*, volume 104 of *Elect. Notes in Th. Comput. Sci.*, pages 199–215. Elsevier Science, 2004.

[LM04b] I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Theory and Practice of Logic Programming, Special Issue on Multiparadigm Languages and Constraint Programming*, 2004. To appear.

[LM04c] I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In J. Rathke, editor, *Proc. of FGUC'04 – Foundations of Global Ubiquitous Computing*, volume 138 of *Elect. Notes in Th. Comput. Sci.*, pages 43–60. Elsevier Science, 2004.

[LM05a] I. Lanese and U. Montanari. Hoare vs Milner: Comparing synchronizations in a graphical framework with mobility. In *Proc. of GT-VC'05, Graph Transformation for Verification and Concurrency*, Elect. Notes in Th. Comput. Sci. Elsevier Science, 2005. To appear.

[LM05b] I. Lanese and U. Montanari. Insights emerged while comparing three models for global computing. In J. L. Fiadeiro, U. Montanari, and M. Wirsing, editors, *Proceedings of Dagstuhl Seminar n. 05081, Foundations of Global Computing*, Electronic proceedings, 2005. To appear.

[Löw93] M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoret. Comput. Sci.*, 109(1&2):181–224, 1993.

[LPV01] C. Laneve, J. Parrow, and B. Victor. Solo diagrams. In N. Kobayashi and B. C. Pierce, editors, *Proc. of TACS'01*, volume 2215 of *Lect. Notes in Comput. Sci.*, pages 127–144. Springer, 2001.

[LT05] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In J.-M. Jacquet and G. P. Picco, editors, *Proc. of Coordination'05*, volume 3454 of *Lect. Notes in Comput. Sci.*, pages 220–235. Springer, 2005.

[LX90] K. G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In M. S. Paterson, editor, *Proc. of ICALP'90*, volume 443 of *Lect. Notes in Comput. Sci.*, pages 526–539. Springer, 1990.

[Lyn89] N. Lynch. A hundred impossibility proofs for distributed computing. In *Proc. of PODC'89*, pages 1–28. ACM Press, 1989.

[Mac71] S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.

[Mes92] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.

[Mil79] R. Milner. Flowgraphs and flow algebras. *Journal of the ACM*, 26(4):794–818, 1979.

[Mil82] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer, 1982.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil94] R. Milner. Pi-nets: A graphical form of π-calculus. In D. Sannella, editor, *Proc. of ESOP'94*, volume 788 of *Lect. Notes in Comput. Sci.*, pages 26–42. Springer, 1994.

[Mil01] R. Milner. Bigraphical reactive systems. In K. G. Larsen and M. Nielsen, editors, *Proc. of CONCUR'01*, volume 2154 of *Lect. Notes in Comput. Sci.*, pages 16–35. Springer, 2001.

[MM90] J. Meseguer and U. Montanari. Petri nets are monoids. *Inform. and Comput.*, 88(2):105–155, 1990.

[MP95] U. Montanari and M. Pistore. Concurrent semantics for the pi-calculus. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Proc. of MFPS'95*, volume 1 of *Elect. Notes in Th. Comput. Sci.* Elsevier Science, 1995.

[MPW92] R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.

[MT97] U. Montanari and C. L. Talcott. Can actors and pi-agents live together? In A. Gordon, A. Pitts, and C. Talcott, editors, *Proc. of HOOTS II*, volume 10 of *Elect. Notes in Th. Comput. Sci.*, 1997.

[MZ03] M. Merro and F. Zappa Nardelli. Bisimulation proof techniques for mobile ambients. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. of ICALP'03*, volume 2719 of *Lect. Notes in Comput. Sci.*, pages 584–598. Springer, 2003.

[Pal90] C. Palamidessi. Algebraic properties of idempotent substitutions. In M. Paterson, editor, *Proc. of ICALP'90*, volume 443 of *Lect. Notes in Comput. Sci.*, pages 386–399. Springer, 1990.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. of Theoretical Computer Science'81*, volume 104 of *Lect. Notes in Comput. Sci.*, pages 167–183. Springer, 1981.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[Plo04] G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[Pri95] C. Priami. Stochastic pi-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[PRO] PROFUNDIS: Proofs of functionality for mobile distributed systems. `http://www.it.uu.se/profundis/`.

[PT97] G. Plotkin and D. Turi. Towards a mathematical operational semantics. In *Proc. of LICS'97*. IEEE Computer Society Press, 1997.

[PT00] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[PV97] J. Parrow and B. Victor. The update calculus. In M. Johnson, editor, *Proc. of AMAST'97*, volume 1349 of *Lect. Notes in Comput. Sci.*, pages 409–423. Springer, 1997. Full version available as Technical report DoCS 97/93, Uppsala University.

[PV98] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS'98*, pages 176–185. IEEE Computer Society Press, 1998.

[Rei85] W. Reisig. *Petri Nets.* Springer, 1985.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Book. *The Unified Modeling Language Reference Manual.* Addison Wesley, 1999.

[RM99] F. Rossi and U. Montanari. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *Applied Categorical Structures*, 7(4):333–370, 1999.

[Roz97] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformations, vol. 1: Foundations*. World Scientific, 1997.

[Rut00] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoret. Comput. Sci.*, 249(1):3–80, 2000.

[San01] D. Sangiorgi. Asynchronous process calculi: the first-order and higher-order paradigms (tutorial). *Theoret. Comput. Sci.*, 253:311–350, 2001.

[SEN] SENSORIA: Software engineering for service-oriented overlay computers. `http://www.pst.ifi.lmu.de/projekte/sensoria/`.

[Sew98] P. Sewell. From rewrite to bisimulation congruences. In D. Sangiorgi and R. de Simone, editors, *Proc. of CONCUR'98*, volume 1466 of *Lect. Notes in Comput. Sci.*, pages 269–284. Springer, 1998.

[Ste94] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, 1994.

[SU01] P. Sewell and A. Unyapoth. Nomadic pict: correct communication infrastructure for mobile computation. In *Proc. of POPL'01*, pages 116–127. ACM Press, 2001.

[Tuo03] E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Computer Science Department, University of Pisa, Italy, 2003.

[Vic98] B. Victor. *The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes*. PhD thesis, Department of Computer Systems, Uppsala University, Sweden, 1998.

[Win84] G. Winskel. Synchronization trees. *Theoret. Comput. Sci.*, 34:33–82, 1984.