Deciding and Axiomatizing Weak ST Bisimulation for a Process Algebra with Recursion and Action Refinement

MARIO BRAVETTI and ROBERTO GORRIERI
University of Bologna

Due to the complex nature of bisimulation equivalences which express some form of history dependence, it turned out to be problematic to decide them over non trivial classes of recursive systems. Moreover, to the best of our knowledge, the problem of axiomatizing them over such classes of systems has never been solved. In this paper we face this problem in the case of weak ST bisimulation, an equivalence which expresses the execution of an action as the combination of the two interdependent events of action start and action termination and which supports the operation of action refinement. We first consider a basic process algebra with CSP multiway synchronization and recursion and we show that a simple technique based on static names is sufficient to decide weak ST bisimulation over processes which are finite state according to the standard interleaving semantics. Then we introduce a different technique based on dynamic names and on the new idea of compositional levelwise renaming of actions (which produces semantic models via SOS such that weak ST bisimulation can be established through standard weak bisimulation) and we show that it can be applied to decide and axiomatize weak ST bisimulation over the same class of processes. Finally we introduce a third technique based on pointers, updated according to a pseudo stack discipline, which preserves the possibility of deciding and axiomatizing weak ST bisimulation also when an action refinement operator is considered.

Categories and Subject Descriptors: F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Operational semantics*; F.1.2 [**Computation by Abstract Devices**]: Models of Computation—*Parallelism and concurrency*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Top-down programming*

General Terms: Languages, Theory.

Additional Key Words and Phrases: Action Refinement, Axiomatization, Noninterleaving Semantics, Process Algebra, ST Bisimulation.

1. INTRODUCTION

ST semantics, originally defined in [van Glabbeek and Vaandrager 1987] over Petri Nets, is one of the most studied noninterleaving semantics. The main reason is that ST semantics is the less informative semantics that is a congruence for

Authors' address: Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy, E-mail: {bravetti, gorrieri}@cs.unibo.it
This paper is an extended and revised version of [Bravetti and Gorrieri 1999].
This research has been partially supported by MURST progetto TOSCA.

action refinement [van Glabbeek 1990; Aceto and Hennessy 1994; Vogler 1993; Gorrieri and Laneve 1991;1995; Busi et al. 1994], thus allowing for the top-down design of concurrent systems. This semantics describes the behavior of a concurrent system in terms of activities which exhibit a duration. In particular, actions are not atomic as in standard interleaving semantics, but the execution of an action is split into the two distinguished events of action start and action termination. Between such events, other system activities may evolve. Moreover, enough information is included in semantic models so that the event of an action termination uniquely determines to which event of action start it refers to, even in the situation of auto-concurrency (i.e., multiple actions of the same type being in execution at the same time). Different implementations of the ST semantics are reported in [van Glabbeek 1990; Busi et al. 1994] and, under other names, also in [Vogler 1993; Aceto and Hennessy 1994; Devillers 1992].

The ST semantics has found application in several different contexts. For example the ST idea of representing durational actions has turned out to be useful to express the semantics of languages with time; e.g., [Vogler 1995] argues that ST bisimulation is the correct equivalence for comparing the time behavior of concurrent systems in an asynchronous setting, while [Bravetti et al. 1998; Bravetti and Gorrieri 2001] shows that, when duration of activities of a concurrent system is specified with a general probability distribution, the use of ST semantics produces suitable semantic models that can be analyzed to obtain performance measures.

Besides ST bisimulation, other bisimulation equivalences expressing some form of history dependence (the fact that in ST semantics action terminations are uniquely related to action starts is a form of history dependence) have been defined in the literature: e.g. history preserving bisimulation [Rabinovich and Trakhtenbrot 1988; Darondeau and Degano 1989; van Glabbeek and Goltz 1989; Degano et al. 1989] and location bisimulation [Boudol et al. 1994; Aceto 1994b; Montanari and Yankelevich 1995; Castellani 1995]. Due to the complex nature of this kind of equivalences it turned out to be problematic to decide them in the presence of recursion (see for instance [Kiehn and Hennessy 1997; Fröschle and Hildebrandt 1999; Fröschle 1999] for positive results and [Jurdziński and Nielsen 2000] for a negative result). Moreover to the best of our knowledge, the problem of axiomatizing them over non trivial classes of systems (e.g. classes which include most recursive systems) has never been solved. For instance, the axiom systems for ST bisimulation presented in [Hennessy 1991; Aceto and Hennessy 1993] only deal with nonrecursive systems.

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

The first approach is based on *static names* [van Glabbeek and Goltz 1989; van Glabbeek 1990; Aceto and Hennessy 1994; Aceto 1994b; Castellani 1995]. According to this approach a unique name is statically assigned (i.e. on the basis of the syntactical structure of the process) to each different "historical element" (e.g. a started action in ST semantics) which must be referred to in labels of process computations. Such references express the history dependence. Then history dependent bisimulation is defined by making associations among static names of historical elements of different processes that are put in correspondence by the equivalence. An advantage of this approach is that it gives the opportunity to produce (as done for location semantics in [Aceto 1994b; Castellani 1995]) finite semantic models also for a wide class of recursive systems. The main drawback of this approach is that each kind of history dependent bisimulation has a different definition which deviates from standard bisimulation. As a consequence, there is no simple way to derive an axiomatization for such history dependent bisimulations by using an approach similar to that employed for standard bisimulation in process algebras with an interleaving semantics (equality of two terms would depend on the associations already established among their historical elements), and the results previously developed for standard bisimulation theory (e.g. tools for verification) cannot be directly exploited.

The second approach is based on *pointers* [Darondeau and Degano 1989; Gorrieri and Laneve 1991;1995; Busi et al. 1994]. According to this approach, a historical dependence is expressed in the label of a computation by a *pointer* that determines the position in the semantic model of the transition that "activated" such "historical element" (i.e. the transition representing an action start in ST semantics). This approach has the advantage that the equivalence of terms, according to a history dependent bisimulation, can be established by simply applying the standard notion of bisimulation on a particular semantic model especially constructed for that kind of history dependent bisimulation. The drawback of the techniques used in [Darondeau and Degano 1989; Gorrieri and Laneve 1991;1995; Busi et al. 1994] is that for most recursive systems (e.g. for $a \,\|\, recX.b.X$ in [Darondeau and Degano 1989; Gorrieri and Laneve 1991; Busi et al. 1994]) a semantic model with an infinite number of states is produced. As a consequence there is no easy way to decide or axiomatize history dependent equivalences between process terms which include recursion.

A further step is made in [Montanari and Pistore 1997]. Here a technique for expressing history preserving semantics is developed, which is based on *dynamic names*. This technique combines the advantages of the first and second approach: finite semantic models are produced for a wide class of recursive systems and history preserving bisimulation is decided simply by applying the standard definition of bisimulation. The idea of the approach of [Montanari and Pistore 1997] is to *dynamically assign* - with a *fixed rule* - a different name to each new "historical element" that becomes active, on the basis of the names of the historical elements already active; in particular, names of obsolete "historical elements" (which are no longer active) are *reused*. Note that this technique is based on names as the approach of [van Glabbeek and Goltz 1989; van Glabbeek 1990; Aceto and Hennessy 1994; Aceto 1994b; Castellani 1995], but here names are not assigned statically (i.e. at compile-time) according to their syntactical position in the initial process, they are instead computed dynamically while the system evolves (i.e. at run-time). [1] Since the method to compute new names is fixed, processes that perform equivalent computations produce the same names for "historical elements". As a consequence the history dependent bisimulation can be decided by applying standard bisimulation. Unfortunately the technique developed in [Montanari and Pistore 1997] is

[1]The idea of dynamic names already appeared in [Boudol et al. 1994], but in that paper no special technique (like reuse of names) is employed in order to obtain finite models.

not compositional, in the sense that, in order to produce the history preserving semantic model of a term, first an intermediate semantic model of the whole term must be computed and then transformed by adjusting history information in transitions. Hence also with the technique of [Montanari and Pistore 1997], being it not compositional, there is no easy way to produce an axiomatization for a history dependent bisimulation.

In this paper, we first show that, considered a process algebra which extends the basic algebra of [Aceto and Hennessy 1994] (which has prefix, choice and CCS [Milner 1989a] parallel composition) with CSP [Hoare 1985] multiway synchronization and recursion, a technique based on static names can be easily employed to decide weak ST bisimulation over processes which are finite state according to the standard interleaving semantics. Then we show that an axiomatization of weak ST bisimulation which is complete over such class of processes can be naturally obtained by combining the idea of dynamic names (which allow us to produce finite semantic models for a wide class of recursive systems and to establish ST bisimulation through standard bisimulation) with the new idea, we introduce here, of *levelwise renaming* of actions which provide us with compositionality. Finally we consider a more complex process algebra which includes in addition an action refinement operator and we face the problem of deciding and axiomatizing weak ST bisimulation for such a process algebra. We solve this problem by considering a third, completely new, idea for referring to historical elements (action starts) which, with respect to the dynamic name technique, maximizes the reuse of names. The resulting technique turns out to be no longer based on (static or dynamic) names, but on a dynamic re-denomination of actions which makes the technique more similar to an approach based on pointers. The main difference with the previous techniques based on pointers is that our new technique, which is called *stack technique*, is not affected by the problem of producing infinite semantic models in the presence of recursion for a wide class of processes. We show that by combining the stack technique with the idea of levelwise renaming it is indeed possible to axiomatize weak ST bisimulation over a wide class of processes of the language with action refinement.

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

1.1 ST Semantics via Static Names

We start by considering a basic process algebra, called *Basic Language* (*BL*), equipped with the CCS prefix and choice operators, the CSP parallel and hiding operators and a recursion operator. We show how to define ST semantics in SOS style [Plotkin 1981] for *BL* by using static names so that finite state semantic models are obtained for processes which are finite state according to the standard interleaving semantics. This is obtained thanks to the observation that, for the purpose of defining an ST semantics, it is not necessary to assign a different static name to *every* action of a process (as done e.g. in [Aceto and Hennessy 1994]), but it is just sufficient to consider action *locations* (syntactical positions with respect to parallel operators) as their static names. Moreover with respect to [Aceto and Hennessy 1994] the definition of operational ST semantics is more complex because of multiway CSP synchronization. As we will see, the fact that visible actions can be obtained by synchronization leads to the necessity of enriching the representation of system states so that they include explicitly the information about the locations of the actions in execution (i.e. started but not terminated yet). The definition of weak ST bisimulation for static names that we employ is just that defined in [Aceto and Hennessy 1994] under the name of "weak refine bisimulation", where equivalence is established by making associations among static names of actions that are put in correspondence by the bisimulation. Note that it is correct to consider the equivalence of [Aceto and Hennessy 1994] as a definition for ST bisimulation since it has been proven in [Gorrieri and Laneve 1992] to correspond to the original definition of ST equivalence [van Glabbeek 1990]. We show that through our technique we can decide weak ST bisimulation over BL processes with a finite state interleaving semantics.

1.2 ST Semantics via Dynamic Names

Then we address the problem of axiomatizing weak ST bisimulation over processes of BL with a finite state interleaving semantics. Our approach is based on the observation that this can be easily done provided that we have a compositional approach for deriving semantic models which express ST semantics via dynamic names (a different name is dynamically assigned to each action that starts execution through a fixed rule and names of terminated actions are reused when new actions start). This because, similarly as in [Montanari and Pistore 1997], such semantic models are finite for processes with a finite state interleaving semantics and ST bisimulation of two processes can be decided simply by applying the standard definition of bisimulation. More precisely what we need is a structural operational semantics (SOS [Plotkin 1981]) which allows us to derive, notably for the parallel composition operator "$\|$", the ST computations of $P \| Q$ (starts or terminations of actions with certain dynamic names) from the ST computations of P and Q. As long as we consider only terms with choice, prefix and termination operators, where ST computations are encoded in prefixes (such terms are just *normal forms*, i.e. representations of a semantic model), axiomatizing standard bisimulation is simply done with the standard axiom set developed by Milner [Milner 1989a]. In order to transform a general term in normal form it is just sufficient to have axioms that reflect the operational rules for the operator $\|$, i.e. they must derive from the computations of P and Q (prefixes representing starts or terminations of actions with certain dynamic names), the computations of $P \| Q$ (in the form of prefixes representing starts or terminations of actions with certain dynamic names).

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

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

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

1.3 ST Semantics via the Stack Technique

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

Moreover we show that with the stack technique it is possible to solve some problems that arise with the dynamic name technique when we consider a *Refinement Language RL* which extends the language *BL* with an action refinement operator $P[a \rightsquigarrow Q]$ which performs the semantic refinement of all actions a executed by P to Q [van Glabbeek 1990; Degano and Gorrieri 1995; Gorrieri and Rensink 2001]. In particular with the dynamic name technique there is no easy way to define a semantics for $P[a \rightsquigarrow Q]$ so that if both P and Q are finite state processes then $P[a \rightsquigarrow Q]$ is finite state, hence to decide and axiomatize ST bisimulation for a wide class of processes containing the refinement operator.

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

1.4 Outline of the Paper

The paper is organized as follows. In Sect. 2 we present the basic language *BL*. In Sect. 3 we present the technique for defining ST semantics of *BL* via static names and we recall the definition of weak ST bisimulation of [Aceto and Hennessy 1994].

In Sect. 4 we present the technique for defining ST semantics of BL via dynamic names and we present a complete axiomatization of weak ST bisimulation of finite state guarded BL processes. In Sect. 5 we analyze the drawbacks of the dynamic name technique and we introduce the refinement language RL. In Sect. 6 we present the stack technique for defining ST semantics of RL and we present a complete axiomatization of weak ST bisimulation of finite state guarded RL processes. In Sect. 7 we compare the size of the ST semantic models produced with our three techniques for a BL process. Finally, in Sect. 8 we report some concluding remarks. In Appendix A we present the definition of strongly guarded processes for both BL and RL, while Appendices B and C are entirely dedicated to the proofs of the two central theorems of the paper: the equivalence of ST bisimulation via dynamic names and via the stack technique with ST bisimulation via static names, respectively.

2. THE BASIC LANGUAGE

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

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

"$\underline{0}$" denotes a process that cannot move. The operators "." and "+" are the CCS [Milner 1989a] prefix and choice. "$\parallel_S$" is the CSP [Hoare 1985] parallel operator, where synchronization over actions in S is required. "$/L$" is the hiding operator which turns the actions in L into τ actions. Finally "$recX$" denotes recursion in the usual way. We denote by BL_C the set of BL processes, i.e. closed terms of BL, and by BL_{CG} the set of strongly guarded processes of BL (see the definition of BL strongly guarded process in Appendix A). In the following, in order to avoid ambiguities, we assume the following operator precedence relation: hiding > prefix > recursion > parallel composition > choice.

3. ST SEMANTICS VIA STATIC NAMES

The *static name* technique is based on the idea of statically assigning a name to each action occurring in a BL process P on the basis of its syntactical position in P. Names must be assigned to actions in such a way that we are guaranteed that any two actions of the same type a that can be executed simultaneously (there may exist a state in which they are both started but not terminated yet) always get different names. In this way, when an action a with a given name terminates, it is always clear which action a (among those in execution) is terminating and we preserve the correct relationship between action starts and action terminations, as required by ST semantics. We point out that, for the purpose of defining an ST semantics, it is not necessary to identify *every* action of a term as a different action (as done, e.g., in [Aceto and Hennessy 1994]). For example two consecutive actions a of a term executed by the same sequential process (with the form $a.a.P'$) cannot be concurrently executed, but are causally related. Thus the two actions cannot overlap during their execution and once an action has started it must terminate before the other one can start. Therefore even if the two actions are not distinguished by the identification mechanism the event of an action termination is still uniquely related to the event of its start (there is no ambiguity about which action is terminating). Also alternative actions a of a term executable by the same sequential process (with the form $a.P' + a.P''$) can be identified in the same way since the execution of one of them excludes the execution of the other one.

The method that we adopt for assigning names to actions is based on *locations.* The name of an action is given by its location defined as the syntactical position (left or right) with respect to the parallel operators of the term P specifying the system. Therefore in the semantic model of a BL process P the event of starting of an observable action a is represented by a transition labeled by $a^+_@$, where @ is the location of the action, while the event of termination of the action is represented by a transition labeled by $a^-_@$, where the location @ is an "identifier" which uniquely determines which action a is terminating. A consequence of using action locations as static names with respect to the approach of [Aceto and Hennessy 1994] is that we can easily deal with recursion and, since action locations are "reused" by the actions of the same sequential process, we can obtain finite semantic models also in the presence of recursion.

For the choice of the CSP operators of multiway synchronization and hiding, we are forced to deviate from the simpler approach of [Aceto and Hennessy 1994]. In particular, since visible actions a can be obtained by synchronization, we need (as we will see) to record the locations of synchronized actions under execution (started but not terminated yet) and so it is necessary to represent system states as pairs $\langle P, ex \rangle$ composed of a term P and a set ex of locations of actions in execution. Another technical difference with [Aceto and Hennessy 1994] regards the treatment of τ actions. Since τ actions are invisible (weak ST bisimulation abstracts from them), we do not have to associate names with τ actions when they are executed, as in [Aceto and Hennessy 1994]. However, differently from [Aceto and Hennessy 1994] where τ actions are executed in a single step, in our approach we represent the execution of a τ action as the combination of two "invisible events" $\dot{\tau}$ of action start and action termination. Both events are represented by $\dot{\tau}$ since they are invisible and we adopt a notion of weak ST bisimulation which abstracts from $\dot{\tau}$ transitions (instead of abstracting from τ transitions as usual). On the other hand splitting τ actions, as we do for visible actions, adheres to the intuition that the semantics of τ should be isomorphic to that of $a/\{a\}$, where the two events a^+ and a^- are separately hidden.

As we already said in the introduction, in the case of the static name technique the definition of ST equivalence (which is taken from [Aceto and Hennessy 1994]) is rather complex because it must somehow abstract from the particular names chosen for the actions of a given BL process P. The problem of name abstraction derives from the fact that the role of static names is just to distinguish actions of the same type executed in parallel by a single process P, but not to distinguish actions of the same type executed by different processes P and P'. Since the mechanism for assigning names to actions is based on syntactical positions, two terms P and P' that behave in an equivalent way may get different names for corresponding actions. For instance in the two terms $a \,\|_\emptyset\, b$ and $b \,\|_\emptyset\, a$, which should be clearly equivalent, the action a of $a \,\|_\emptyset\, b$ gets name a_l (where l stands for "left") while the action a of $b \,\|_\emptyset\, a$ gets name a_r (where r stands for "right") and similarly for actions b. In order to implement abstraction from action names the definition of weak ST bisimulation must somehow associate the names of the actions used by one process with the names of the corresponding actions used by the other one. More precisely, while checking weak ST bisimulation of two processes P and Q, we make use of an *association history* H where we record associations $(a, @', @'')$ between names of corresponding actions, where a is the type of the two actions, $@'$ is the location of the action a of P, and $@''$ is the location of the corresponding action a of Q. We add a new association $(a, @', @'')$ to H whenever P performs an $a^+_{@'}$ transition and Q performs a corresponding $a^+_{@''}$ transition. After the association $(a, @', @'')$ is established, the two processes P and Q are equivalent only if P performs the transition $a^-_{@'}$ in correspondence to the transition $a^-_{@''}$ of Q. Whenever this happens, the association $(a, @', @'')$ is removed from H. In the following we use $a\!:\!(@', @'')$ to stand for $(a, @', @'') \in H$.

3.1 Definition of ST Operational Semantics via Static Names

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

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

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

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

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

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

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

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

Example 3.2. Consider $(a^-.P \,\|_\emptyset\, b^-.Q) \,\|_{\{b\}}\, b^-.R$. Then $ll\bullet$ is the global location of the action a in execution in the leftmost sequential process. Moreover $<r\bullet \mid \bullet>$

is the global location of the action in execution obtained by synchronizing the two actions b. ∎

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

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

The states of the semantic model of a process of BL are pairs $\langle T, ex\rangle$, with $T \in BL_{sn}$ and $ex \in Ex$, where ex includes the global locations of the actions in execution in T. The set of states is denoted by $\Sigma = \mathcal{P}(BL_{sn,C} \times Ex)$, ranged over by s, t. It is necessary to include ex in the representation of a system state because it records the synchronization structure of started actions. Consider for example the term $(a.\underline{0} \,\|_\emptyset\, a.\underline{0}) \,\|_{\{a\}} (a.\underline{0} \,\|_\emptyset\, a.\underline{0})$. In this term two actions may start execution before any action terminates so that we reach the state $(a^-.\underline{0} \,\|_\emptyset\, a^-.\underline{0}) \,\|_{\{a\}} (a^-.\underline{0} \,\|_\emptyset\, a^-.\underline{0})$. In this term we cannot distinguish if the two actions that have started are $a_{\langle l\bullet | l\bullet\rangle}$ and $a_{\langle r\bullet | r\bullet\rangle}$, or $a_{\langle l\bullet | r\bullet\rangle}$ and $a_{\langle r\bullet | l\bullet\rangle}$. Note that the necessity of including global locations of actions in execution in the representation of system states derives from the capability of BL of performing multiway synchronizations. Due to the CSP synchronization mechanism in BL the actions obtained by synchronization are observable and must be assigned a name (a global location). [2]

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

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

Let us briefly comment on the operational rules.

The meaning of the rules for prefix is the following. $\langle a.P, \emptyset\rangle$, where the action a still has to be executed (hence the set of action under execution is empty), becomes

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

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

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

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

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

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

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

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

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

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

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

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

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

Table I. Rules for the Static Name Technique

$\langle a^-.P, \{a : \bullet\}\rangle$ after the event of start of action a, where a is the only action under execution. $\langle a^-.P, \{a : \bullet\}\rangle$ becomes $\langle P, \emptyset\rangle$ after the event of termination of action a (hence the set of action under execution becomes empty again). $\langle \tau.P, \emptyset\rangle$ becomes $\langle \dot{\tau}.P, \emptyset\rangle$ and then $\langle P, \emptyset\rangle$ in a similar way.

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

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

—A termination move $a^-_{l@}$, with $a \notin S$, made by $\langle T \parallel_S U, ex\rangle$ such that $a : l@ \in ex$ is originated from the unique move $a^-_@$ of $\langle T, l(ex)\rangle$, that exists because $a : @ \in l(ex)$. Symmetrically a termination move $a^-_{r@}$, with $a \notin S$, is derived from an $a^-_@$ move of U.

—The silent $\dot{\tau}$ moves of $\langle T \parallel_S U, ex\rangle$ are derived from the $\dot{\tau}$ moves of $\langle T, l(ex)\rangle$ and of $\langle U, r(ex)\rangle$.

—A starting move $a^+_{<@'|@''>}$, with $a \in S$, is made by $\langle T \parallel_S U, ex\rangle$ whenever $\langle T, l(ex)\rangle$ makes an $a^+_@$ move and $\langle U, r(ex)\rangle$ makes an $a^+_@$ move.

—A termination move $a^-_{<@'|@''>}$, with $a \in S$, made by $\langle T \parallel_S U, ex\rangle$ such that $a : <@'|@''> \in ex$ is originated from the unique move $a^-_{@'}$ of $\langle T, l(ex)\rangle$, that exists because $a : @' \in l(ex)$, and from the unique move $a^-_{@''}$ of $\langle U, r(ex)\rangle$, that exists because $a : @'' \in r(ex)$.

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

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

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

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

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

Example 3.3. In Fig. 1 we depict the ST semantic model of $recX.a.X \parallel_\emptyset recX.a.X$ obtained by applying the static name technique. ■

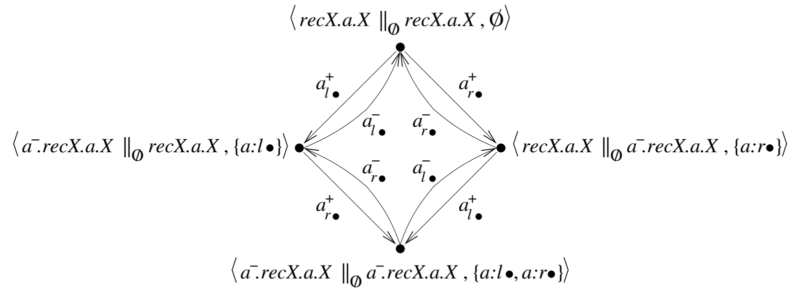

Fig. 1. Example of Recursion with the Static Name Technique

The following theorem shows that our approach is indeed suitable to cope with recursive processes. We consider the interleaving semantics of a process P to be the transition system including the processes in BL_C reachable from P according to the standard operational rules of [Milner 1989a; Hoare 1985], and the transitions among such processes.

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

PROOF.
Proof of "$\Longrightarrow$" Let us consider the term P' obtained by syntactically substituting $\mu.\mu$ for each action μ occurring in the term P. Since the interleaving semantics of P is finite state, it easily derives that also the interleaving semantics of P' is finite state. It is easy to see that if, inside the state terms of the ST semantics of P, we syntactically replace each ST action a^- or $\dot{\tau}$ with its type and we discard the sets ex of locations of the actions in execution from states, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because the number of different locations that the actions of P may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.
Proof of "$\Longleftarrow$" Starting from the ST semantics of P, we consider only those paths where the termination transition for an action immediately follows the starting transition for that action, thus removing all transitions and states not reached in this way. The semantic model that we obtain is just the interleaving model of P where each transition is split into two transitions and a state is added between them. □

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

Now we present a simple syntactic criterion which constitutes a sufficient condition for a process of BL_C to be finite state. In the following corollary we call "static" [Milner 1989a] the operators of parallel composition and hiding only.

COROLLARY 3.5. *Let $P \in BL_C$ be a process s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator. Then P is a finite* *state process.*

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

3.2 Weak ST Bisimulation via Static Names

As we already explained, the equivalence notion we consider over BL terms is that presented in [Aceto and Hennessy 1994]. In particular the equivalence between a pair of states is established depending on a given association history, i.e. a correspondence between their sets of actions in execution.

We denote by la a *location association*, whose elements are associations $(@', @'')$. la ranges over the set LA of partial bijections from $GLoc$ to $GLoc$. An association history H is a relation from A to $GLoc \times GLoc$ such that $\forall a \in A. H_a \in LA$, [3] i.e. H is a set including an independent location association for each different action type. We denote by $\mathcal{H}$ the set of all association histories. [4]

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

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

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

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

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

(4) Silent $\dot{\tau}$ moves must be matched as in standard weak bisimulation [Milner 1989a].

Formally, weak ST bisimulation is defined by means of a family of bisimulations (binary relations) $\mathcal{B}_H$, each indexed by a an association history H, as follows. First of all, let us say that a $\mathcal{H}$-indexed family of binary relations $\{\mathcal{B}_H \mid H \in \mathcal{H}\}$ over

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

[4] Since we use global locations as static names for actions and, as we already observed, it can never happen that, in a state, we have two actions in execution with the same global location (even if they are of different type), it is not strictly necessary to include action types in association histories. We prefer to do this in order to adhere to the definition of ST bisimulation of [Aceto and Hennessy 1994] and for clarity reasons.

Σ_R is *symmetric* if and only if $(s,t) \in \mathcal{B}_H$ implies $(t,s) \in \mathcal{B}_{\overline{H}}$, where $\overline{H} = \{a : (@'', @') \mid a : (@', @'') \in H\}$.

Definition 3.6. A *weak ST bisimulation family* is a symmetric $\mathcal{H}$-indexed family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ of binary relations over Σ_R such that for each $H \in \mathcal{H}$ and $s, t \in \Sigma_R$ we have that $s\,\mathcal{B}_H\,t$ implies:

(1) H is compatible with (s,t)

(2) $s \xrightarrow{a^+_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ t \xRightarrow{a^+_{@'}} t' \wedge (s', t') \in \mathcal{B}_{H \cup \{a:(@,@')\}}$

(3) $s \xrightarrow{a^-_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@, @') \in H \wedge t \xRightarrow{a^-_{@'}} t' \wedge (s', t') \in \mathcal{B}_{H - \{a:(@,@')\}}$

(4) $s \xrightarrow{\dot{\tau}} s'$ implies $\exists t' \in \Sigma_R.\ t \xRightarrow{\varepsilon} t' \wedge (s', t') \in \mathcal{B}_H$ ■

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

Definition 3.7. Given $P, Q \in BL_C$, P is *observationally ST congruent* to Q (written $P \simeq_{sn} Q$), if and only if:

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

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

—$\langle P, \emptyset \rangle \xrightarrow{\dot{\tau}} s$ implies $\exists t \in \Sigma_R.\ \langle Q, \emptyset \rangle \xRightarrow{\dot{\tau}} t \wedge (s,t) \in \approx_{sn,\emptyset}$

—$\langle Q, \emptyset \rangle \xrightarrow{\dot{\tau}} t$ implies $\exists s \in \Sigma_R.\ \langle P, \emptyset \rangle \xRightarrow{\dot{\tau}} s \wedge (s,t) \in \approx_{sn,\emptyset}$ ■

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

THEOREM 3.8. *$\simeq_{sn}$ is a congruence for BL.*

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

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

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

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

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

$r(H)$ is similarly defined as

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

and Id_{ex} is defined as $Id_{ex} = \{a : (@, @) \mid a : @ \in ex\}$.

As far as the recursion operator is concerned, in order to show that for all $P', P'' \in BL$ containing at most the variable X free we have that $P' \simeq_{sn} P''$ (meaning $\forall\, Q \in BL_C .\, P'\{Q/X\} \simeq_{sn} P''\{Q/X\}$, as usual [Milner 1989a]) implies $recX.P' \simeq_{sn} recX.P''$, it suffices to verify that the family (one for each $ex \in Ex$) of binary relations over Σ_R

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

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

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

—$s \xrightarrow{\dot{\tau}} s'$ implies $\exists t', t'' \in \Sigma_R .\, t \stackrel{\dot{\tau}}{\Longrightarrow} t' \wedge s' \;\mathcal{B}_{Id_{ex}}\; t'' \approx_{sn, Id_{ex}} t'$

—$s \xrightarrow{a^-_@} s$ implies $\exists t' \in \Sigma_R .\, t \xrightarrow{a^-_@} t' \wedge s' \;\mathcal{B}_{Id_{ex - \{a:@\}}}\; t'$

This is done by induction on the depth of the inference by which a transition $s \xrightarrow{\theta} s'$ is inferred, similarly as in [Milner 1989a]. It is easy to see that (similarly as for the notion of "bisimulation up to $\approx$" of [Milner 1989a]) the three conditions above (together with the three symmetric conditions for the moves of t) imply that for all $ex \in Ex$ we have $\mathcal{B}_{Id_{ex}} \subseteq \approx_{Id_{ex}}$. Hence, by taking $T \equiv X$ and $ex = \emptyset$, the conditions above imply that $recX.P' \simeq_{sn} recX.P''$. □

From Theorem 3.4 it easily derives the decidability of observational ST congruence over processes possessing a finite state interleaving semantics.

Theorem 3.9. *Observational ST congruence is decidable over processes of BL_C which possess a finite state interleaving semantics.*

Proof. Since the ST semantic model via static names of BL_C processes with a finite state interleaving semantics is finite state, observational ST congruence can be simply decided via a partition refinement algorithm similar to that of Paige and Tarjan [1987]. In particular given $P, Q \in BL_C$ we establish if $P \simeq_{sn} Q$ as follows. First of all we explicitly compute weak transitions among the states of the static name ST semantics of P, and similarly for Q. Then we evaluate $\approx_{sn,H}$ for each association history $H \in \mathcal{H}$ such that there exist a state s of the semantics of P and a state t of the semantics of Q such that H is compatible with (s,t). This is done by modifying partition refinement algorithm of [Paige and Tarjan 1987] as follows. We consider the states in the semantic models of P and Q to be distinguished and for each association history $H \in \mathcal{H}$ satisfying the condition above we consider a partition over the states of P and Q which are compatible with H. Therefore we deal with a set of partitions parameterized over H, each divided into a set of classes C_H. At the beginning of the algorithm each partition consist of a single class C_H containing all the states. Then the algorithm proceeds similarly as in [Paige and Tarjan 1987] by iteratively refining all the classes currently considered (independently on the partition they belong to) until the $\mathcal{H}$-indexed family of partitions is a weak ST bisimulation family. □

4. ST SEMANTICS VIA DYNAMIC NAMES

The *dynamic name* technique is based on the idea of dynamically assigning, during the evolution of the system, a new name to each action that starts execution, on the basis of the names assigned to the actions already started. *Names* are indexes

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

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

4.1 Definition of ST Operational Semantics via Dynamic Names

Now we show how the dynamic name technique can be exploited to give operational ST semantics to our basic language *BL*. We need a richer syntax to represent the states of semantic models. We denote by $STA_{dn} = \{a_i^+ \mid a \in A \wedge i \in \mathbb{N}^+\} \cup \{a_i^- \mid a \in A \wedge i \in \mathbb{N}^+\}$ (where the *dn* stands for the dynamic name technique), the set of observable ST actions, where a_i^+ represents the beginning of the action a identified with i and a_i^- represents the ending of the action a identified with i. γ ranges over STA_{dn}. [5] The set of all ST actions is denoted by $STAct_{dn} = STA_{dn} \cup \{\dot{\tau}\}$. θ ranges over $STAct_{dn}$ and η ranges over $Act \cup STAct_{dn}$. We denote by *ia* an *index association*, whose elements are associations (i, loc_j). *ia* ranges over the set *IA* of partial bijections from $\mathbb{N}^+$ to $Loc \times \mathbb{N}^+$. A mapping M is a relation from A to $\mathbb{N}^+ \times (Loc \times \mathbb{N}^+)$ such that $\forall a \in A.\, M_a \in IA$, i.e. M is a set including an

[5] For simplicity and reuse, we usually denote the meta-variable for a set, which is the version with dynamic names of a set already used in Sect. 3 (e.g. STA_{dn} corresponds to STA_{sn}), by the same name used in the case of static names.

independent index association for each different action type.

The set BL_{dn} of state terms of BL is generated by the following syntax: [6]

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

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

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

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

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

Table II. Standard Rules

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

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

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

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

—since actions of a given type $a \in S$ are numbered independently from actions of other types,

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

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

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

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

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

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

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

Table III. Rules for the Dynamic Name Technique

—since the rule for generating new indexes for actions a starting in T and U is the same, and

—since actions of type a are required to start and terminate in T and U at the same time and with the same index,

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

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

Example 4.1. In Fig. 2 we depict the ST semantic model of $recX.a.X \,\|_{\emptyset}\, recX.a.X$ obtained by applying the dynamic name technique. ■

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

THEOREM 4.2. *The interleaving semantics of a process $P \in BL_C$ is finite state if and only if the ST semantics of P via dynamic names is finite state.*

PROOF.

Proof of "$\Longrightarrow$" As in the proof of Theorem 3.4 we consider the finite state term P' obtained by syntactically substituting $\mu.\mu$ for each action μ occurring in

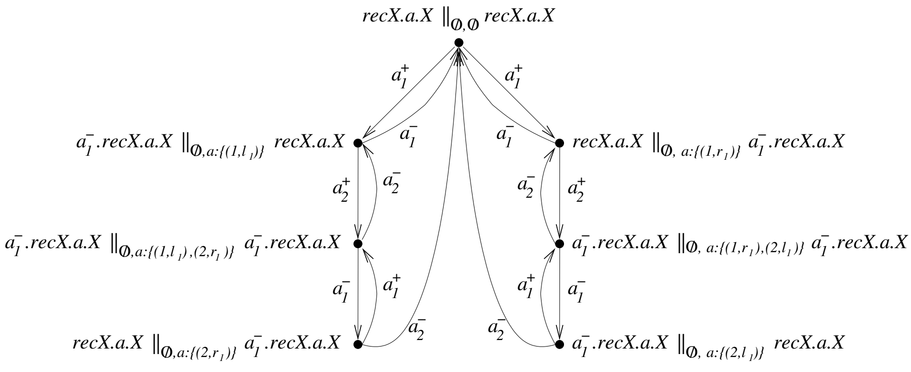

Fig. 2. Example of Recursion with the Dynamic Name Technique

the term P. If, inside the state terms of the ST semantics of P, we syntactically replace each ST action with its type and we discard the sets M in the operators $\|_{S,M}$, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because, due to the policy of using the minimum index available for a starting action, the maximum index an action may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.
Proof of "$\Longleftarrow$" The same as in the proof of Theorem 3.4. $\square$

As a consequence the simple syntactic criterion for BL processes that are guaranteed to be finite state we considered in Sect. 3.1 (processes P s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator), which strictly includes the class of nets of automata, is valid also for the dynamic name technique.

4.2 Weak ST Bisimulation via Dynamic Names

The equivalence notion we consider over terms of BL_{dn}, denoted by $\simeq_{dn}$, is the standard notion of *observational congruence* extended to open terms [Milner 1989a], where the alphabet of visible actions is STA_{dn} and hidden actions are $\dot{\tau}$ actions.

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

THEOREM 4.3. *Given two processes P and Q of BL_C we have that $P \simeq_{dn} Q$ if and only if $P \simeq_{sn} Q$.*

PROOF. See Appendix B. $\square$

From Theorems 4.3 and 3.8 it immediately derives that $\simeq_{dn}$ is preserved by all the operators of BL, including recursion.

COROLLARY 4.4. *$\simeq_{dn}$ is a congruence for BL.* ■

As far as decidability of $\simeq_{dn}$ is concerned, obviously (since we rely on the standard notion of observational congruence) $\simeq_{dn}$ is decidable over processes of BL_C which possess a finite-state interleaving semantics. This can be seen also as a corollary of Theorems 4.3 and 3.9.

4.3 Axiomatization of Weak ST Bisimulation via Dynamic Names

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

The axiom system $\mathcal{A}_{BL}$ for $\simeq_{dn}$ on BL_{dn} terms is formed by the standard axioms presented in Table IV and by the axioms of Table V, which are specific for the dynamic name technique. "$\lfloor\!\lfloor$" and "$|$" denote the left merge and synchronization merge operators [Bergstra and Klop 1985; Aceto 1994a], respectively.

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

Table IV. Standard Axioms

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

Table V. Axioms for the Dynamic Name Technique

The axiom (Par) is the standard one except that when the position of processes T and U is exchanged we must invert left and right inside M. More precisely, the inverse $\overline{M}$ of a mapping M is defined by $\overline{M} = \{a : (i, r_j) \mid a : (i, l_j) \in M\} \cup \{a : (i, l_j) \mid a : (i, r_j) \in M\}$. Axioms ($LM5$) and ($LM6$) just reflect the operational rules of the parallel operator for an independent move of the lefthand process. As far as axiom ($Rec2$) is concerned, we define a variable X to be sequential in a term if each free occurrence of X in that term is in the scope of operators $\theta.T$, $T' + T''$ and $recX.T$ only. Moreover we assume the standard definition of [Milner 1989a] for strong guardedness of sequential variables.

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

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

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

Table VI. Rules for the Left and Synchronization Merge Operators

In Table VI we present the operational rules for "$\lfloor\!\lfloor_{S,M}$" and "$|_{S,M}$", which are obtained by applying levelwise renaming of actions (as for the parallel operator) to operational rules similar to those considered in [Aceto 1994a]. In particular the definition of the operational rule for "$|_{S,M}$" allows for actions "$\dot{\tau}$" to be skipped [Aceto 1994a], as reflected by axiom ($SM5$).

THEOREM 4.5. *The axioms of $\mathcal{A}_{BL}$ are sound for $\simeq_{dn}$ over BL_{dn} terms.*

PROOF. A simple consequence of the soundness of the axiomatization of [Aceto 1994a]. □

Once defined a sequential state to be a state that can be built using only $\underline{0}$, X and operators $\theta.T$, $T' + T''$, $recX.T$, and assumed the standard definition of [Milner 1989a] for strong guardedness of sequential states, we have the following theorem.

LEMMA 4.6. *If a process $P \in BL_{CG}$ is finite state, then $\exists T \in BL_{dn,C}$. $\mathcal{A}_{BL} \vdash P = T$ with T strongly guarded sequential state.*

PROOF. Let $T_1 \dots T_n$ be the states of the ST semantics of P, $T_n \equiv P$. Since P is strongly guarded, each state T_i of the semantics of P is finitely branching. It can be easily seen that, for each $i \in \{1 \dots n\}$, there exist $m_i \in \mathbb{N}$, $\{\theta^i_j\}_{j \le m_i}$, $\{k^i_j\}_{j \le m_i}$ s.t. $\mathcal{A}_{BL} \vdash T_i = \sum_{j \le m_i} \theta^i_j . T_{k^i_j}$ where $\sum_{\emptyset} \equiv \underline{0}$. Hence we can characterize the behavior of P by means of a set of equations similarly as in [Milner 1984]. Moreover, similarly as for the unique solution of equations theorem of [Milner 1984], we have that there is a sequential term T such that $\mathcal{A}_{BL} \vdash T = T_n \equiv P$. This can be shown as follows. For each i, from 1 to n, we do the following. If i is such that $\exists j \leq m_i : k^i_j = i$ we have, by applying ($Rec2$), that $T_i = recX.(\sum_{j \leq m_i : k^i_j \neq i} \theta^i_j.T_{k^i_j} + \sum_{j \leq m_i : k^i_j = i} \theta^i_j.X)$. Then we replace each subterm T_i occurring in the equations for $T_{i+1} \ldots T_n$ with its equivalent term. When, in the equation for $T_n \equiv P$, we have replaced T_{n-1}, we are done. □

Since for sequential states the ST semantics coincide with the standard interleaving semantics and the axioms of $\mathcal{A}_{BL}$ involved are just the standard axioms for CCS, from [Milner 1989b] and Lemma 4.6 we derive the completeness of $\mathcal{A}_{BL}$.

THEOREM 4.7. *$\mathcal{A}_{BL}$ is complete for $\simeq_{dn}$ over finite state processes of BL_{CG}.*

PROOF. From [Milner 1989b] we have that given two sequential states T and T' such that $T \approx_{dn} T'$ then $\mathcal{A}_{BL} \vdash T = T'$, hence the result directly follows from Lemma 4.6. □

5. EXTENDING THE LANGUAGE WITH REFINEMENT

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

Example 5.1. A hole in the ordered sequence of indexes of actions a started by a term $T \,\|_{S,M}\, U$ can be generated as follows. Consider $a.\underline{0} \,\|_\emptyset\, a.\underline{0}$. After the righthand a starts, the state is $a.\underline{0} \,\|_{\emptyset, a:\{(1,r_1)\}}\, a^-_1.\underline{0}$. After the lefthand a starts, the state is $a^-_1.\underline{0} \,\|_{\emptyset, a:\{(1,r_1),(2,l_1)\}}\, a^-_1.\underline{0}$. Finally, after the righthand a terminates, the state is $a^-_1.\underline{0} \,\|_{\emptyset, a:\{(2,l_1)\}}\, \underline{0}$ where only index 2 is being used. Therefore we have a hole in position 1. ■

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

Example 5.2. An index permutation occurs when an association function does not preserve the order of indexes. For instance consider the process $(a.\underline{0} \,\|_\emptyset\, a.\underline{0}) \,\|_\emptyset\, a.\underline{0}$. After the rightmost a starts, the state is $(a.\underline{0} \,\|_{\emptyset,\emptyset}\, a.\underline{0}) \,\|_{\emptyset, a:r_1}\, a^-_1.\underline{0}$. After the leftmost a starts, the state is $(a^-_1.\underline{0} \,\|_{\emptyset, a:l_1}\, a.\underline{0}) \,\|_{\emptyset, a:r_1 l_1}\, a^-_1.\underline{0}$. After the rightmost a terminates, the state is $(a^-_1.\underline{0} \,\|_{\emptyset, a:l_1}\, a.\underline{0}) \,\|_{\emptyset, a:* l_1}\, \underline{0}$. Finally after the central a starts, the state is $(a^-_1.\underline{0} \,\|_{\emptyset, a:l_1 r_1}\, a^-_1.\underline{0}) \,\|_{\emptyset, a:l_2 l_1}\, \underline{0}$. Therefore we have an index permutation: the action with index 1 is mapped to l_2 and the action with index 2 is mapped to l_1. ■

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

Moreover, the two phenomena above cause problems when we try to extend our basic language with an action refinement operator $P[a \rightsquigarrow Q]$ which performs the *semantic* refinement [Gorrieri and Rensink 2001] of all occurrences of a in P by Q. This means that every time an action a is executed by P, $P[a \rightsquigarrow Q]$ executes instead a new copy of Q in parallel with P. As suggested also in [Busi et al. 1994], $P[a \rightsquigarrow Q]$ can be defined in term of the parallel operator and other basic operators. As we will see, if we want to obtain a refinement operator with the desirable property that if both P and Q are finite state processes then $P[a \rightsquigarrow Q]$ is finite state, then we must have the possibility to define an elimination rule for the parallel operator such that $P \,\|_\emptyset\, Q$ is turned into P if Q is terminated. With the name technique such a rule cannot be implemented in general. For instance $a.\underline{0} \,\|_\emptyset\, a.\underline{0}$ reaches the state $a_1^-.\underline{0} \,\|_{\emptyset,a:\{(2,l_1)\}}\, \underline{0}$ which is not equivalent to $a_1^-.\underline{0}$ as well as $(a.P \,\|_\emptyset\, a.Q) \,\|_\emptyset\, a.\underline{0}$ reaches the state $(a_1^-.P \,\|_{\emptyset,a:l_1r_1}\, a_1^-.Q) \,\|_{\emptyset,a:l_2l_1}\, \underline{0}$ which is not equivalent to $a_1^-.P \,\|_{\emptyset,a:l_1r_1}\, a_1^-.Q$. The point is that a parallel operator may be safely eliminated only if the related mapping is a simple identity, which is rarely the case due to the possible presence of holes or index permutations.

Note that the problem above arises because we consider semantic action refinement instead of, e.g., *syntactic* action refinement (see [Gorrieri and Rensink 2001]), where a restricted form of refinement is simply obtained by syntactically replacing terms for actions. But it is not due to the particular implementation of semantic action refinement that we consider in this paper with respect to, e.g., other implementations of semantic action refinement. This because, according to the notion of semantic action refinement, if actions a of a process P are refined by a term Q, every time a new a is executed by P, a new instance of Q must be put in parallel with P via a static operator. So no matter if this operator is a special operator different from the parallel operator used in the algebra (as, e.g., for the algebras of [Busi et al. 1994; Degano and Gorrieri 1995]), since it is static we still have the problem of removing it when the execution of an instance of Q is terminated.

In the rest of the paper we will consider a language which extends our basic language with an action refinement operator $P[a \rightsquigarrow Q]$, where we distinguish deadlock, denoted by δ, from successful termination, denoted by ε (otherwise ST bisimulation could not be a congruence for the refinement operator) and we employ the ACP [Baeten and Weijland 1990] sequential composition operator ";" instead of the CCS prefix operator ".".

Let A, Act and Var be defined as for the basic language BL of Sect. 2. The terms of RL (Refinement Language) are generated by the following syntax:

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

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

6. ST SEMANTICS VIA THE STACK TECHNIQUE

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

Since the index of a started action change dynamically while other actions start and terminate, this technique is not based on names (seen as identifiers for actions) but is more similar to the approach [Gorrieri and Laneve 1991;1995; Busi et al. 1994] based on pointers. In particular, the event of starting of an action a is represented in semantic models by a transition labeled with a^+ (so no index is observable) whilst the event of termination of an action a is represented by a transition labeled with a_i^- where i is the current position of the action on the stack. The event of action start referred to by a transition a_i^- can be uniquely determined by going back in the history of process computations (reconstructing the history of the stack state at each backward step) until the transition a^+ that pushed on the stack the action that now is at position i is reached. More precisely, the procedure for determining which transition a^+ is pointed by a_i^- is the following.

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

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

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

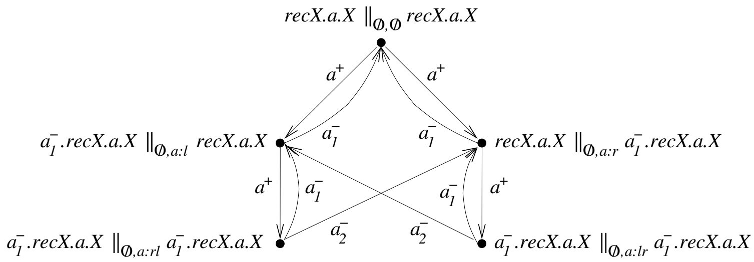

Fig. 3. Example of Recursion with the Stack Technique

be checked by applying standard bisimilarity.

Let w range over the set $IAS = \{w : \mathbb{N}^+ \nrightarrow Loc \mid \exists k \geq 1.\, dom(w) = \{1 \dots k\}\}$ of index association strings, i.e. nonempty strings over the alphabet of locations $Loc = \{l, r\}$ [7], ranged over by loc. M includes an association string $w \in IAS$ for each action type a. A string w associated to a type a represents a stack of started a actions, where the action a_1 on the top of the stack corresponds to the leftmost position in w and the action a_i corresponds to the i-th position in w. The location in the i-th position of w determines if the action a_i is executed by the lefthand term (if the location is "l") or righthand term (if the location is "r") of the parallel composition operator. The index j of the action a of the lefthand (or righthand) term associated to a_i is determined as follows. Index j is given by the position of the l (or r) in the string obtained from w by removing all locations r (or l).

Example 6.1. In Fig. 3 we depict the ST semantic model of $recX.a.X \,\|_{\emptyset}\, recX.a.X$ obtained by applying the stack technique. [8] By comparing the semantic model of Fig. 3 with that of Fig. 2 we can see that the phenomenon of holes of the dynamic name technique generates only two additional states. If we consider $(recX.a.X \,\|_{\emptyset}\, recX.a.X) \,\|_{\emptyset}\, recX.a.X$ we have that, due to the combined effects of holes and index permutations, the ST semantic model obtained with the dynamic name technique has 42 states, whilst that obtained with the stack technique has only 16 states. ■

6.1 Definition of ST Operational Semantics via the Stack Technique

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

[7]Even if the set of nonempty strings over Loc is usually denoted by Loc^+ we prefer to stick to this notation to be consistent with the notation of the dynamic name technique.
[8]We apply the stack technique to a simple language with prefixing instead of general sequential composition in the running example only, for ease of comparison.

ranged over by γ; [9] and $STAct_s = STA_s \cup \{\dot{\tau}\}$. θ ranges over $STAct_s$ and η ranges over $Act_e \cup STAct_s$. The meta-variable χ ranges over $STAct_s \cup \{\surd\}$, where $\surd$ is a distinguished action representing successful termination. A mapping M is a relation from A_e to $\mathbb{N}^+ \times Loc$ such that $\forall \alpha \in A_e. M_\alpha \in IAS$, i.e. M is a set including an independent index association string for each different state action type. Similarly as for the dynamic name technique we will use $a : (i, loc)$ to stand for $(a, i, loc) \in M$ and we will use the shorthand notation $a : w$, where $w \in IAS$ is an index association string, to stand for the set of triples $\{a : (i, loc) \mid (i, loc) \in w\}$ referring to the same action a. For instance we will use $a : lr$ to stand for $\{a : (1, l), a : (2, r)\}$. Finally, let φ range over the bijections over A_e. The set RL_s of state terms of RL is generated by the following syntax:

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

We denote by $RL_{s,C}$ the set of processes, i.e. closed terms, of RL_s. The bang operator "!" and the (bijective) relabeling operator "$[\varphi]$" are auxiliary operators that are used for the definition of the refinement operator. [10] In the following, in order to avoid ambiguities, we assume the following operator precedence relation: hiding = bang = relabeling > sequential composition > recursion > parallel composition > choice > refinement.

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

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

The following auxiliary definitions are used in the definition of the operational semantics of RL. The function $type : STAct_s \cup \{\surd\} \longrightarrow Act_e \cup \{\surd\}$ is defined in the obvious way. We define $M[\alpha \mapsto w]$ with $\alpha \in A_e, w \in IAS$, which modifies a mapping M so that w becomes the new index association string for actions of type α, as follows: $M[\alpha \mapsto w] = \{\alpha' : (i, loc) \in M \mid \alpha' \neq \alpha\} \cup \{\alpha : (i, loc) \mid (i, loc) \in w\}$. The termination predicate $\surd$ is defined as $T\surd \Leftrightarrow (\exists U \in RL_{s,C}.\ T \xrightarrow{\surd} U) \wedge (\nexists \theta \in STAct_s, U' \in RL_{s,C}. T \xrightarrow{\theta} U')$. The expression $\#_{i,l}(w)$ computes the position of the i-th l in the string w. We have that $\#_{i,l}(w)$ is the only $j \in \mathbb{N}^+$ such that $w(j) = l$ and $|\{k \leq j \mid w(k) = l\}| = i$. Similarly for $\#_{i,r}(w)$. Finally we define $w \ominus i$ as the string obtained by removing the i-th element from the string w, i.e. $w \ominus i = \{(j, loc) \in w \mid j < i\} \cup \{(j-1, loc) \mid (j, loc) \in w \wedge j > i\}$. [11]

---

[9] For simplicity and reuse, we usually denote the meta-variable for a set, which is the version with the stack technique of a set already used in Sect. 3 and 4 (e.g. STA_s corresponds to STA_{dn} and STA_{sn}), by the same name used in the case of static and dynamic names.

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

[11] To be precise both $\#_{i,loc}(w)$ and $w' \ominus j$ should be defined to assume a default value when w does not have an i-th symbol loc or j is greater than the length of w'. Since these situations cannot

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

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

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

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

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

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

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

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

Table VII. Rules for the Stack Technique

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

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

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

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

---

arise for terms reachable from a term $P \in RL$ or, as we will see, derivable via the axiomatization of weak ST bisimulation from a term $P \in RL$, the actual default value is not important.

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

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

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

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

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

Table VIII. Rules for Refinement

the way it works:

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

where the bijective relabeling $\alpha \leftrightarrow \alpha'$ is defined by $\alpha \leftrightarrow \alpha' = \{(\alpha, \alpha'), (\alpha', \alpha)\} \cup \{(\alpha'', \alpha'') \mid \alpha'' \in A_e \wedge \alpha'' \notin \{\alpha, \alpha'\}\}$. For each a executed by the process T a corresponding process U is activated by the bang operator in the righthand term. In this way if T executes several auto-concurrent actions a then a corresponding number of processes U are executed in parallel by the righthand term. The correct association between actions a and processes U is guaranteed by the fact that the events of starting and termination of each auto-concurrent action e are uniquely related by the ST semantics. [12]

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

The operational rules for $!T$ just say that we can execute zero or more parallel copies of T. The number of copies depends on the initial choice performed by $!T$ between termination (action $\surd$) and execution of T in parallel with a new instance of $!T$. Such new instance is generated only after the execution of the first action of T. This avoids the immediate generation of an unbounded number of instances of $!T$ which would lead to infinite state models. Note that, in the definition of the refinement operator, this interpretation of the bang operator leads to the execution of the correct number of instances of its argument, because we exploit the fact that a process in parallel with another process may actually perform the terminating action $\surd$ only when both processes are ready to terminate (actions $\surd$ of parallel processes must synchronize).

[12] This definition of semantic action refinement is slightly different from the usual definition [van Glabbeek 1990; Degano and Gorrieri 1995; Gorrieri and Rensink 2001] in that in $T[a \rightsquigarrow U]$ each execution of U is preceded and followed by the occurrence of a silent transition $\dot{\tau}$. In order to obtain a definition which adheres completely to the usual one it is simply sufficient to "skip" e transitions (instead of just hiding them with "$/\{e\}$"), similarly as done in [Busi et al. 1994].

THEOREM 6.2. *The interleaving semantics of a RL process P not including the refinement operator is finite state if and only if the ST semantics of P obtained with the stack technique is finite state.*

PROOF.
Proof of "$\Longrightarrow$" Similarly as in the proof of Theorems 3.4 and 4.2 we consider the finite state term P' obtained by syntactically substituting $\mu;\mu$ for each action μ occurring in the term P. If, inside the state terms of the ST semantics of P, we syntactically replace each ST action with its type and we discard the sets M in the operators $\|_{S,M}$, then we obtain a subset of the states of the interleaving semantics of the term P'. In particular, the number of states of the ST semantics of P that collapse in the same state is finite because, since holes in the ordered sequence of started action indices generated by action termination are immediately eliminated by the stack policy, the maximum index an action may assume is bounded by the maximum number of processes that may run in parallel in a state. Such quantity is finite because the number of states of the interleaving semantics of P' is finite.
Proof of "$\Longleftarrow$" The same as in the proof of Theorem 3.4. □

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

THEOREM 6.3. *If P and Q are finite state RL processes, then $P[a \rightsquigarrow Q]$ is a finite state process.*

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

Example 6.4. In Fig. 4 we present the finite ST semantic model of $recX.a;X[a \rightsquigarrow b;c]$ obtained with the stack technique. In Fig. 5 we show an initial fragment of the infinite ST semantic model of the same term obtained with the dynamic name technique. Note that ST semantics via the dynamic name technique over the whole language RL is simply obtained as follows. The operational rules for the refinement, relabeling and bang operators are the same as for the stack technique, except that e^+ is replaced by e_1^+ in the premise for refinement, α^+ is replaced by α_i^+ in the rules for relabeling and bang operators and $\varphi(\alpha)^+$ is replaced by $\varphi(\alpha)_i^+$ in the rule for relabeling. Fig. 5 makes clear that, in the absence of an elimination rule for the parallel operator, the number of parallel operators generated by the bang operator grows as new actions to be refined start and terminate. Therefore even refining a simple recursive term such as $recX.a;X$ leads to an infinite semantic model. [13] ■

[13] For the simple example $recX.a;X[a \rightsquigarrow b;c]$, the execution of a refinement by means of $!(e_1^+;b;c;e_1^-)$ always leads to $\varepsilon \|_{\emptyset,\emptyset} !(e_1^+;b;c;e_1^-)$, where the parallel operator could in fact be eliminated. The fact that, with the name technique, we cannot apply the elimination rule to the parallel operators generated by refinement, can be seen by considering a refinement (e.g. $a \rightsquigarrow b;c$) of the term $recX.a;X \,\|\, recX.a;X$, whose semantic model (see Fig. 2) includes states which exhibit "holes" in the index sequences of started actions.

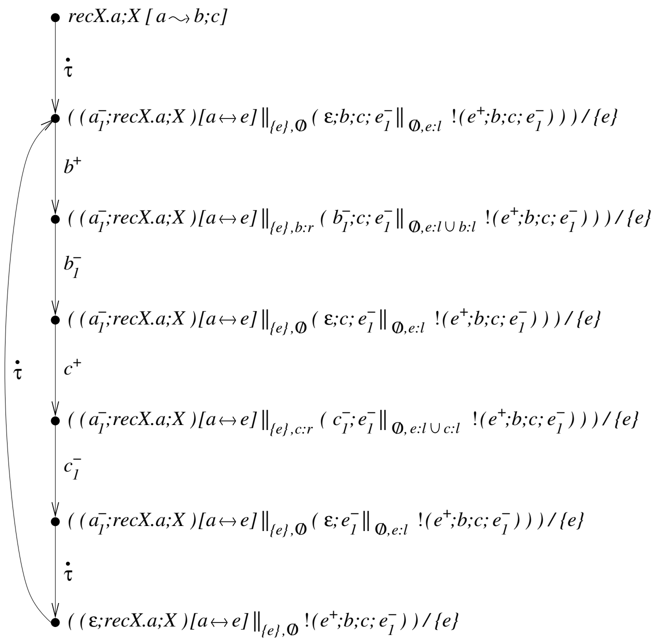

Fig. 4. Example of Refinement with the Stack Technique

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

COROLLARY 6.5. *Let P be a process of RL_C s.t. for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a static operator or in the lefthand side of a ";". Then P is a finite state process.*

6.2 Weak ST Bisimulation via the Stack Technique

The equivalence notion we consider over terms of RL_s, denoted by $\simeq_s$, is again observational congruence extended to open terms [Milner 1989a], where the alphabet of visible actions is STA_s and hidden actions are $\dot{\tau}$ actions.

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

THEOREM 6.6. *Given two processes P and Q of BL_C we have that $P \simeq_s Q$ if and only if $P \simeq_{sn} Q$.*

PROOF. See Appendix C. □

From Theorems 6.6 and 4.3 we immediately derive also the equivalence between the stack technique and the dynamic name technique.

COROLLARY 6.7. *Given two processes P and Q of BL_C we have that $P \simeq_s Q$ if and only if $P \simeq_{dn} Q$.*

Moreover from Theorems 6.6 and 3.8 it easily derives that $\simeq_s$ is preserved by all the operators of RL, including recursion.

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

Fig. 5. Example of Refinement with the Name Technique

COROLLARY 6.8. *$\simeq_s$ is a congruence for RL.*

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

THEOREM 6.9. *Observational ST congruence is decidable over processes of RL_C which possess a finite state stack technique ST semantics.*

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

6.3 Axiomatization of Weak ST Bisimulation via the Stack Technique

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

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

Table IX. Axioms for the Stack Technique

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

Table X. Axioms for Refinement

The axiom system $\mathcal{A}_{RL}$ for $\simeq_s$ on RL_s terms is formed by: the standard axioms presented in Table IV where we replace the operator ";" for the operator ".", [14] "δ" for "$\underline{0}$", $\mathcal{L}$ for L and $\mathcal{S}$ for S; the axioms of Table IX which are specific for the stack technique; and the axioms of Table X which deal with the refinement operator.

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

In Table XI we present the operational rules for "$\lfloor\!\lfloor_{\mathcal{S},M}$" and "$|_{\mathcal{S},M}$", which are obtained from the rules of the parallel operator by following an approach similar to that of [Aceto 1994a].

THEOREM 6.10. *The axioms of $\mathcal{A}_{RL}$ are sound for $\simeq_s$ over RL_s terms.*

PROOF. A simple consequence of the soundness of the axiomatization of [Aceto 1994a] and of the fact that the semantic model of $!P$ is isomorphic to that of $recX.(\varepsilon + P \,\lfloor\!\lfloor_{\emptyset,\emptyset}\, X)$ where X does not occur free in P. □

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

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

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

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

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

Table XI. Rules for the Left and Synchronization Merge Operators

Once defined a sequential state to be a state that can be built using only ε, δ, X and operators $\theta;T$, $T' + T''$, $recX.T$, and assumed the standard definition of [Milner 1989a] for strong guardedness of sequential states, we have the following theorem.

LEMMA 6.11. *If a process $P \in RL_{CG}$ is finite state then $\exists T \in RL_{s,C}.\, \mathcal{A}_{RL} \vdash P = T$ with T strongly guarded sequential state.*

PROOF. The proof of this theorem is completely analogous to that of Lemma 4.6, in particular ";" has the same role as "." and "δ" has the same role as "$\underline{0}$". The only difference is that a term "ε" must be added to the summation in the equation for a state T_i if T_i has an outgoing transition labeled with "$\surd$". $\square$

Since for sequential states the ST semantics coincide with the standard interleaving semantics and the axioms of $\mathcal{A}_{RL}$ involved are just the standard axioms for CCS (it suffices to consider "$\eta;T$" as being "$\eta.T$", "δ" as being "$\underline{0}$" and "ε" as being "$\surd.\underline{0}$"), from [Milner 1989b] and Lemma 6.11 we derive the completeness of $\mathcal{A}_{RL}$.

THEOREM 6.12. *$\mathcal{A}_{RL}$ is complete for $\simeq_s$ over finite state processes of RL_{CG}.*

PROOF. From [Milner 1989b] we have that given two sequential states T and T' such that $T \approx_s T'$ then $\mathcal{A}_{RL} \vdash T = T'$, hence the result directly follows from Lemma 6.11. $\square$

7. COMPARISON OF THE THREE TECHNIQUES

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

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

THEOREM 7.1. *Given a term $P \in BL_C$, the number of states of the semantic model of P produced with the static name technique is less or equal to the number of states of the semantic model of P produced with the dynamic name and stack techniques. Moreover if the behavior of P is such that it may execute at least two concurrent actions of the same type, then the number of states of the semantic model of P produced with the static name technique is strictly smaller than the number of states of the semantic model of P produced with the dynamic name and stack techniques.* ■

PROOF. It is easy to see that, in both the dynamic name and the stack technique ST semantic models of P, the number of states corresponding to the same state $\langle T, ex\rangle$ of the ST static name semantic model of P (i.e. states U such that $sn(U) = \langle T, ex\rangle$ according to Definitions B.4 and C.4) is at least $\sum_{a\in A:\, ex_a\neq\emptyset}(|ex_a|!)$, i.e. for each action type we consider the factorial of the number of the action of that type in execution in the state. This because, being the actions in ex in parallel execution in $\langle T, ex\rangle$, they are independent (not causally related), hence they can be started in any order. Therefore both the dynamic name and the stack technique ST semantic models of P include at least a different state U such that $sn(U) = \langle T, ex\rangle$ for each possible permutation of the actions in ex of the same type. Formally, if for each action type a such that $ex_a \neq \emptyset$ we consider a bijection $Perm_a$ from $\{1 \ldots |ex_a|\}$ to ex_a, then both in the dynamic name and the stack technique ST semantic models of P there exists a state U such that, for each action type a considered above, we have (according to Definitions B.1 and C.1) $dsmap(U)_a = Perm_a$ for the dynamic name technique and $ssmap(U)_a = Perm_a$ for the stack technique. Finally note that states of the dynamic name and the stack technique semantic models of P corresponding to different states of the static name semantic model of P are obviously different, otherwise sn would not be a function. □

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

THEOREM 7.2. *Given a term $P \in BL_C$, the number of states of the semantic model of P produced with the stack technique is less or equal to the number of states of the semantic model of P produced with the dynamic name technique. Moreover if the behavior of P is such that it may execute at least two concurrent actions of the same type, then the number of states of the semantic model of P produced with the stack technique is strictly smaller than the number of states of the semantic model of P produced with the dynamic name technique.* ■

PROOF. As we showed in the proof of Theorem 7.1 the number of states of the dynamic name semantic model of P in correspondence with a given state $\langle T, ex\rangle$ of the static name semantic model of P is at least $\sum_{a\in A:\, ex_a\neq\emptyset}(|ex_a|!)$. It is easy to see that, since in the stack technique the actions in execution of the same type are always recorded in a stack whose dimension is equal to the number of such actions, the number of states of the stack technique semantic model of P in correspondence with $\langle T, ex \rangle$ is at most $\sum_{a \in A: ex_a \neq \emptyset}(|ex_a|!)$, i.e. the number of possible configurations (permutations of actions of the same type) for all the stacks. Formally, this derives from the fact that in each state U of the stack technique semantic model of a process $P \in BL_C$, the indexes (representing stack positions) associated with the actions in execution of a given type in the state always range from 1 to k for some k, i.e. for each action type a we have $dom(ssmap(U)_a) = \{1 \ldots k_a\}$ according to Definition C.1. Moreover it is easy to see that for each ordering of the actions of ex in the stacks, there is at most one state U of the stack technique semantic model of P in correspondence with $\langle T, ex \rangle$ implementing that ordering, i.e. such that $ssmap(U)_a$ yields the desired ordering for the actions of ex of type a. Finally, if the behavior of P is such that it may execute at least two concurrent actions of the same type, then in the dynamic name semantics of P there is a state U such that we have two actions in execution, both with the same type a, with associated indexes 1 and 2. Moreover U must have an outgoing transition representing the termination of the action a_1 leading to a state U' where we have a "hole" in position 1. Therefore, considered the state $sn(U')$ of the static name semantics of P corresponding to U' (see Definition B.4) which has a single action in execution, we have that in the stack technique semantics of P there is a single state corresponding to $sn(U')$, while in the dynamic name semantics of P there are at least two states corresponding to $sn(U')$ (U' and the state U'' such that $sn(U'') = sn(U')$ and the action a in execution has index 1). $\square$

8. CONCLUSION

Even if the stack technique is more adequate in the context of ST semantics because it allows to decide and axiomatize ST bisimulation also in the presence of an action refinement operator, we believe that the three techniques for expressing ST semantics that we have presented have different features that may make one of them more suitable than another one depending on the context of application. We have introduced the static name technique first of all because it allows us to use the equivalence of [Aceto and Hennessy 1994] which has been proven in [Gorrieri and Laneve 1992] to coincide to the original definition of ST equivalence [van Glabbeek 1990]. In this way we have that all three equivalences considered in this paper actually coincide with ST bisimulation. Secondly because it is a technique producing much smaller semantic models with respect to the other two techniques. Moreover we have introduced the dynamic name technique for the following reasons. First of all, it shows how dynamic names, which have been previously used in the literature for producing finite models in the presence of recursion, can be managed compositionally, thus giving rise to structural operational semantics for terms of process algebras and axiomatizations. Secondly, the dynamic name technique is more intuitive than the stack technique because it identifies an action with the same name throughout all its execution and makes it possible to understand the (quite complex) notion of levelwise renaming, without getting at the same time into the intricate behavior of the stack technique. Thirdly, for certain problems it may constitute the unique solution, i.e. the stack technique may not be applicable. For example when ST semantics is used to express execution of activities in timed systems (see e.g. [Bravetti et al. 1998; Bravetti and Gorrieri 2001]), the models used in the literature (e.g., Timed Automata or Generalized Semi-Markov Processes for probabilistic time) require an activity to maintain its name throughout all its execution (name of the clock representing the activity) so the stack technique cannot be applied to derive models of this kind from an algebraic specification. Moreover, even if the dynamic name technique is not practically applicable to a language including an action refinement operator (semantic action refinement [Gorrieri and Rensink 2001]), we could still rely on syntactic action refinement [Gorrieri and Rensink 2001], i.e. we could perform a limited form of action refinement by syntactically replacing terms for actions.

The finiteness result for ST semantics and refinement makes the top-down design of concurrent systems actually performable. This result also finds application in other areas. For instance, in the context of probabilistically timed process algebras where time is expressed by associating a general probability distribution of duration to actions, we have developed two languages, which can be used to specify and analyze real-time and performance properties of systems, one based on ST semantics via static names [Bravetti et al. 1998] and another one based on ST semantics via dynamic names [Bravetti and Gorrieri 2001]. The use of the ST semantics via static and dynamic names (as opposed to the use of the stack technique) gives rise to semantic models which are very close to Generalized Semi-Markov Processes (GSMPs) which can be analyzed to evaluate performance measures of systems. In particular names assigned to actions by ST semantics become the names of the elements of the GSMP deriving from an algebraic specification. Since in such process algebras the mathematical evaluation of performance measures of systems rely on the analysis of the stationary behavior of cyclic systems, it is crucial that we can obtain finite state models also in the presence of recursion. Moreover in this context the finiteness result for the refinement operator gives rise to the new possibility of "refining" phase-type distributions (a special class of duration probability distributions which are very useful in many applications and can be used to approximate every kind of duration distribution) into combinations of exponential distributions that are easily analyzable through well-known mathematical techniques (i.e. analysis of Continuous Time Markov Chains) for deriving performance measures of systems.

Finally, we would like to note that, if we just consider the process algebras commonly used in the literature which can be endowed with an interleaving semantics (and we do not consider ad-hoc operators which exploit the fact of splitting actions through ST semantics), our results are not language dependent. For example, considering ACP [Baeten and Weijland 1990] instead of CSP parallel composition would only lead to more complex mappings (parameterizing the parallel operators) for representing the associations between the computations of $P \,||\, Q$ and the computations of P and Q that originate them. This because in ACP an action of type a of $P \,||\, Q$ can be originated from the synchronization of two actions of P and Q with a different type, e.g. b and c. Therefore we have to parameterize parallel operators with mappings associating (indexed) actions of different types. Nevertheless, apart from the implementation details of levelwise renaming, the static name, dynamic name and stack technique can still be used to finitely represent the behavior of systems. In particular concerning the dynamic name and stack technique, the fact that we obtain a finite ST semantic model for terms having a finite interleaving semantic model is not at all related to the operators used in the term, but is due to the way we generate names for action started at the top level by the term and reuse names of actions terminated at the top level by the term.

ACKNOWLEDGMENT

We thank the anonymous referees for the helpful comments and suggestions.

APPENDIX

A. DEFINITION OF STRONG GUARDEDNESS

In this appendix we will provide the precise definition of strongly guarded process for both the Basic Language and the Refinement Language.

We start with the definition of strong guardedness for Basic Language process. If we exclude static operators from the language the definition that we present coincides with the definition given in [Milner 1989a].

Definition A.1. A process P of BL_C is strongly guarded if for every subterm $recX.Q$ of P the following holds: for each free occurrence of X in Q we have that there exists $a \in A$ and a subterm $a.Q'$ of Q such that the same occurrence of X lies inside Q' and $a.Q'$ occurs in P not in the scope of an hiding operator "$/L$" with $a \in L$. ■

As far as the definition of strong guardedness for the Refinement Language is concerned, the presence of the ACP sequential composition operator ";" (instead of the CCS prefix operator) and of the action refinement operator make the definition of strongly guarded processes slightly more complex with respect to the case of the basic language, due to technical reasons.

Similarly as for the basic language, in order for a process P of RL_C to be strongly guarded we must have that every occurrence of a variable X in a subterm $recX.Q$ of P is guarded, i.e. there exists a subterm $R; Q'$ of Q such that X lies inside Q' and we are guaranteed that R executes a visible action before terminating. First of all note that, due the adoption of ACP sequential composition, we can have that a variable X in Q can be guarded by several visible actions (e.g. in $(a + b); X$ the variable X is guarded by the set of actions $\{a, b\}$), while in the case of prefix a strongly guarded variable X is always guarded by a single action. Secondly, we note that, differently from the case of the Basic Language, the execution of X in Q may be preceded by the occurrence of a free variable $Y \neq X$ as, e.g., in process "$recY.a; recX.Y; X$". Therefore in order to understand if X is strongly guarded in $recX.Q$, we have to evaluate the closure of subterm Q by replacing each free variable with the corresponding definition. Thirdly, due to the presence of the action refinement operator we have to account for the combined effect of the refinement and hiding operators which include term R preceding variable X in their scope, e.g. X is not strongly guarded in $((a; X)[a \rightsquigarrow b])/b$.

In the following we present three auxiliary definitions which are necessary for the definition of strong guardedness in the Refinement Language.

Definition A.2. Given $P \in RL_C$ and a finite set $M \subset A$, we define the predicate $G_M(P)$, stating that the successful termination of P is strongly guarded by the set of visible actions M, by structural induction as follows:

$$
\begin{array}{ll}
G_M(X),\ G_M(\delta) & \forall\, M \subset_{\mathit{fin}} A \\
G_M(a) & \text{if } a \in M \\
G_M(Q';Q''),\ G_M(Q' \parallel_S Q'') & \text{if } G_M(Q') \vee G_M(Q'') \\
G_M(Q' + Q'') & \text{if } G_M(Q') \wedge G_M(Q'') \\
G_M(Q/L) & \text{if } G_{M-L}(Q) \\
G_M(\mathit{rec}X.Q) & \text{if } G_M(Q) \\
G_M(Q'[a \rightsquigarrow Q'']) & \text{if } (G_M(Q') \wedge a \notin M) \vee (G_{M\cup\{a\}}(Q') \wedge G_M(Q''))
\end{array}
$$

■

Note that it is correct to consider the successful termination of a subterm X of P to be strongly guarded because P is a closed term, hence a variable X occurs in P always in the scope of its definition $\mathit{rec}X.Q$. Moreover, the successful termination of $\mathit{rec}X.Q$ is not strongly guarded only if in $\mathit{rec}X.Q$ the successful termination may be reached without encountering neither visible actions nor X occurrences.

Definition A.3. Given $P \in RL_C$ and a subterm Q of P, we define the closure of Q to be the closed term obtained, starting from Q, by subsequently replacing any occurrence of any free variable X with the subterm $\mathit{rec}X.R$ binding such variable in P. ■

Note that, since variable occurences which are bound by inner recursive definitions including Q in their scope are not free inside outer recursive definitions including Q in their scope, the sequence of substitutions considered in Definition A.3 is always finite.

Definition A.4. Given $P \in RL$ and a variable $X \in \mathit{Var}$ such that P includes one free occurrence of X, we define $\mathit{RefHi}_X(P)$, the term obtained from P by considering only the action refinement and hiding operators including X in their scope, as follows:

$$
\begin{array}{ll}
\mathit{RefHi}_X(X) = X & \\
\mathit{RefHi}_X(Y) = \perp & \text{if } Y \neq X \\
\mathit{RefHi}_X(\mathit{rec}X.P) = \mathit{RefHi}_X(\delta) = \mathit{RefHi}_X(\varepsilon) = \mathit{RefHi}_X(\mu) = \perp & \\
\mathit{RefHi}_X(\mathit{rec}Y.P) = \mathit{RefHi}_X(P) & \text{if } Y \neq X \\
\mathit{RefHi}_X(P + Q) = \mathit{RefHi}_X(P \parallel_S Q) = \mathit{RefHi}_X(P;Q) \ = & \\
\qquad\qquad \max(\mathit{RefHi}_X(P), \mathit{RefHi}_X(Q)) & \\
\mathit{RefHi}_X(P/L) = \mathit{RefHi}_X(P)/L & \\
\mathit{RefHi}_X(P[a \rightsquigarrow Q]) = \mathit{RefHi}_X(P)[a \rightsquigarrow Q] &
\end{array}
$$

where we assume $\perp \leq P \ \ \forall P \in RL$ and $\perp/L = \perp[a \rightsquigarrow P] = \perp$. ■

We are now in a position to present the definition of strongly guarded processes of RL_C.

Definition A.5. A process P of RL_C is strongly guarded if for every subterm $\mathit{rec}X.Q$ of P the following holds: for each free occurrence of X in Q we have that there exists $M \subset_{fin} A$ and a subterm $R; Q'$ of Q such that the same occurrence of X lies inside Q' and, accounting for the refinement and hiding operators which include R in their scope, the successful termination of term $Cl(R)$, representing the closure of R as a subterm of P, is strongly guarded by the set of actions M. More precisely, defined R^c to be the context of R in P expressed by means of a fresh free variable Y, i.e. $P \equiv R^c\{R/Y\}$, we must have that the successful termination of term $RefHi_Y(R^c)\{Cl(R)/Y\}$ is strongly guarded by M, i.e. $G_M(RefHi_Y(R^c)\{Cl(R)/Y\})$ holds true. ■

B. EQUIVALENCE OF THE DYNAMIC NAME AND STATIC NAME TECHNIQUES

In this appendix we will provide the proof of Theorem 4.3, i.e. the equivalence of ST bisimulation expressed through dynamic names with the definition of ST bisimulation based on static names. In the following we will denote by $\longrightarrow_{sn}$ the state transitions of the static name operational semantics in order to distinguish them from those of the dynamic name operational semantics (simply denoted by $\longrightarrow$).

We denote by $BL_{dn,CR}$ the subset of $BL_{dn,C}$ of the state terms reachable from a process of BL_C, i.e. $BL_{dn,CR} = \{T \in BL_{dn,C} \mid \exists P \in BL_C, k \in \mathbb{N}, \{T_i\}_{i<k}, \{\theta_i\}_{i\leq k}. P \xrightarrow{\theta_1} T_1 \xrightarrow{\theta_2} \dots \xrightarrow{\theta_{k-1}} T_{k-1} \xrightarrow{\theta_k} T\}$.

The proof of Theorem 4.3 is based on a mapping *dsmap* between the dynamic and static names assumed by the actions of a process according to the dynamic name and static name ST semantics, respectively. An association between the dynamic name $i \in \mathbb{N}^+$ and the static name $@ \in GLoc$ assumed by an action a according to the two semantics is represented inside *dsmap* by the triple $(a, i, @)$. In the following we use $a : (i, @)$ to stand for $(a, i, @) \in dsmap$.

We denote by *dsa*, a *dynamic-static association*, whose elements are associations $(i, @)$. *dsa* ranges over the set *DSA* of partial bijections from $\mathbb{N}^+$ to *GLoc*. A *dynamic-static mapping dsmap* is a relation from A to $\mathbb{N}^+ \times GLoc$ such that $\forall a \in A.\, dsmap_a \in DSA$, i.e. *dsmap* is a set including an independent dynamic-static association for each different action type. We denote by *DSMAP* the set of all dynamic-static mappings.

Definition B.1. The function $dsmap : BL_{dn,CR} \longrightarrow DSMAP$, which evaluates a dynamic-static mapping from a state term of a dynamic ST semantic model, is defined as follows:

—$dsmap(\underline{0}) = dsmap(\mu.P) = dsmap(\dot{\tau}.P) = dsmap(P + Q) = dsmap(recX.P) = \emptyset$

—$dsmap(a_1^-.P) = \{a : (1, \bullet)\}$

—$dsmap(T \parallel_{S,M} U) =$

$\{a : (j, l@) \mid a \notin S \wedge a : (i, @) \in dsmap(T) \wedge a : (j, l_i) \in M\} \cup$

$\{a : (j, r@) \mid a \notin S \wedge a : (i, @) \in dsmap(U) \wedge a : (j, r_i) \in M\} \cup$

$\{a : (i, <@'|@''>) \mid a \in S \wedge a : (i, @') \in dsmap(T) \wedge a : (i, @'') \in dsmap(U)\}$

—$dsmap(T/L) = \{a : (i, @) \in dsmap(T) \mid a \notin L\}$ ■

The following proposition determine how the dynamic-static mapping $dsmap(T)$ changes during the evolution of a dynamic state T in the three cases of an action start, an action termination and a silent transition.

PROPOSITION B.2. *Let T be a dynamic state of $BL_{dn,CR}$. We have:*

—*If* $T \xrightarrow{a_i^+} T'$ *then* $i = \min\{k \mid k \notin dom(dsmap(T)_a)\}$ *and there exists* $@ \in GLoc$ *such that* $@ \notin range(dsmap(T)_a)$ *and* $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$. *Moreover* $dsmap(T')_b = dsmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$.

—*If* $T \xrightarrow{a_i^-} T'$ *then there exists* $@ \in GLoc$ *such that* $(i, @) \in dsmap(T)_a$ *and* $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$. *Moreover* $dsmap(T')_b = dsmap(T)_b$ *for all* $b \in A$ *such that* $b \neq a$.

—*If* $T \xrightarrow{\dot{\tau}} T'$ *then* $dsmap(T') = dsmap(T)$.

PROOF. By induction on the derivation of a transition $T \xrightarrow{\theta} T'$, by showing that whenever all the premises of a dynamic name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T, the conclusion satisfies it. □

Now we will show how to derive the state of the static name semantics corresponding to a given state of the dynamic name semantics.

Definition B.3. The function $CL : BL_{dn,CR} \longrightarrow BL_{sn}$, where CL stands for "clean", is defined as follows. Given $T \in BL_{dn,CR}$, $CL(T)$ is obtained by syntactically replacing each ST action a_1^- occurring in T with a^- and by discarding the sets M in the operators $\|_{S,M}$. ■

Definition B.4. The function $sn : BL_{dn,CR} \longrightarrow \Sigma_R$, which evaluates the state $sn(T)$ of the static name ST semantics corresponding to a state T of the dynamic name ST semantics, is defined as follows:

$$sn(T) = \langle CL(T), \bigcup_{a \in A} \{a : @ \mid @ \in range(dsmap(T)_a)\}\rangle$$

■

Another preliminary result which is needed before proving the equivalence of the dynamic name and static name approaches is to establish a correspondence between the dynamic name and static name transitions.

PROPOSITION B.5. *Let T be a dynamic state of $BL_{dn,CR}$. We have:*

$$—T \xrightarrow{a_i^+} T' \quad \textit{implies} \quad \exists @ \in GLoc.\ dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\} \wedge sn(T) \xrightarrow{a_@^+}_{sn} sn(T')$$

$$—T \xrightarrow{a_i^-} T' \quad \textit{implies} \quad \exists @ \in GLoc.\ dsmap(T')_a = dsmap(T)_a - \{(i, @)\} \wedge sn(T) \xrightarrow{a_@^-}_{sn} sn(T')$$

$$—T \xrightarrow{\dot{\tau}} T' \quad \textit{implies} \quad sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$$

PROOF. By induction on the derivation of a dynamic name transition from T by showing that whenever all the premises of a dynamic name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T and static name transitions are defined by structural semantics, the conclusion satisfies it. $\square$

Proposition B.6. *Let T be a dynamic state of $BL_{dn,CR}$. We have:*

$$—sn(T) \xrightarrow{a^+_@}_{sn} s \quad \textit{implies} \quad \exists i \in \mathbf{N}^+, T' \in BL_{dn,CR}.\ T \xrightarrow{a^+_i} T' \wedge s = sn(T') \wedge dsmap(T')_a = dsmap(T)_a \cup \{(i,@)\}$$

$$—sn(T) \xrightarrow{a^-_@}_{sn} s \quad \textit{implies} \quad \exists i \in \mathbf{N}^+, T' \in BL_{dn,CR}.\ T \xrightarrow{a^-_i} T' \wedge s = sn(T') \wedge dsmap(T')_a = dsmap(T)_a - \{(@,i)\}$$

$$—sn(T) \xrightarrow{\dot\tau}_{sn} s \quad \textit{implies} \quad \exists T' \in BL_{dn,CR}.\ T \xrightarrow{\dot\tau} T' \ \wedge\ s = sn(T')$$

Proof. By induction on the derivation of a static name transition from $sn(T)$ by showing that whenever all the premises of a static name operational rule satisfy the proposition, then, exploiting the fact that $dsmap(T)$ is defined by structural induction over T and dynamic name transitions are defined by structural semantics, the conclusion satisfies it. $\square$

Finally, we introduce function $H(T,U)$ which computes a (static name) association history H from two dynamic name states T and U which are put in correspondence by a dynamic name equivalence.

Definition B.7. The function $H : BL_{dn,CR} \times BL_{dn,CR} \longrightarrow \mathcal{H}$, which evaluates an association history H between the global locations of the actions in execution of a pair of dynamic name state terms (T,U) such that $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$, i.e. the set of dynamic names of the actions in execution in T and U is the same, is defined as follows:

$$H(T,U) = \{a : (@', @'') \mid a \in A \wedge dsmap(T)_a^{-1}(@') = dsmap(U)_a^{-1}(@'')\} \qquad \blacksquare$$

Now we are in a position to present the two main lemmas needed for the proof of Theorem 4.3. The first lemma shows that from a dynamic name ST bisimulation we can derive a static name ST bisimulation, the second lemma shows the converse.

Definition B.8. Let $\mathcal{B}$ be a weak bisimulation [Milner 1989a] over dynamic name state terms of $BL_{dn,C}$. The corresponding $\mathcal{H}$-indexed family of binary relations $\mathbf{sn}(\mathcal{B}) = \{sn(\mathcal{B})_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ among states of a static name ST semantic model is defined as follows:

$$sn(\mathcal{B})_H = \{(sn(T), sn(U)) \mid T,U \in BL_{dn,CR} \wedge (\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)) \wedge T\,\mathcal{B}\,U \wedge H(T,U) = H\} \qquad \blacksquare$$

Lemma B.9. *Given a weak bisimulation $\mathcal{B}$ [Milner 1989a] over dynamic name state terms of $BL_{dn,C}$, we have that $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family.*

Proof. First of all let us observe that $\mathbf{sn}(\mathcal{B})$ is a a symmetric $\mathcal{H}$-indexed family of binary relations over Σ_R because $\mathcal{B}$ is a symmetric binary relation.

Supposed that $H \in \mathcal{H}$ and $(s,t) \in sn(\mathcal{B})_H$ we show that:

(1) H is compatible with (s,t)

(2) $s \xrightarrow{a^+_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ t \xRightarrow{a^+_{@'}} t' \wedge (s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$

(3) $s \xrightarrow{a^-_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@, @') \in H \wedge t \xRightarrow{a^-_{@'}} t' \wedge$
$(s', t') \in sn(\mathcal{B})_{H - \{a:(@,@')\}}$

(4) $s \xrightarrow{\dot\tau} s'$ implies $\exists t' \in \Sigma_R.\ t \xRightarrow{\varepsilon} t' \wedge (s', t') \in sn(\mathcal{B})_H$

In the following we will separately prove each item. Note that, by definition of $\mathbf{sn}(\mathcal{B})$, we have that there exist $T, U \in BL_{dn,CR}$ such that $s = sn(T)$ and $t = sn(U)$.

(1) Since, by definition of $\mathbf{sn}(\mathcal{B})$, we have that $H(T, U) = H$, then $dom(H_a) = range(dsmap(T)_a)$ and $range(H_a) = range(dsmap(U)_a)$, hence H is compatible with $(sn(T), sn(U))$.

(2) If $sn(T) \xrightarrow{a^+_@}_{sn} s'$ then, from Prop. B.6 and B.2, we have that there exist $i \in \mathbf{N}^+$ and $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{a^+_i} T'$. Moreover, we have $dsmap(T')_a = dsmap(T)_a \cup \{(i, @)\}$, with $(i, @) \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\ dsmap(T')_b = dsmap(T)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\, \mathcal{B}\, U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \xRightarrow{a^+_i} U'$ and $T'\, \mathcal{B}\, U'$.
Then, from repeated use of Prop. B.5 and B.2, we derive that there exists $@' \in GLoc$ such that $sn(U) \xRightarrow{a^+_{@'}}_{sn} sn(U')$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a \cup \{(i, @')\}$, with $(i, @') \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\ dsmap(U')_b = dsmap(U)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\ dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) \cup \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$.

(3) If $sn(T) \xrightarrow{a^-_@}_{sn} s'$ then, from Prop. B.6 and B.2, we have that there exist $i \in \mathbf{N}^+$ and $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $(i, @) \in dsmap(T)_a$ and $T \xrightarrow{a^-_i} T'$. Moreover, we have $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$ and $\forall b \in A, b \neq a.\ dsmap(T')_b = dsmap(T)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\, \mathcal{B}\, U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \xRightarrow{a^-_i} U'$ and $T'\, \mathcal{B}\, U'$.
Then, from repeated use of Prop. B.5 and B.2, we derive that there exists $@' \in GLoc$ such that $(j, @') \in dsmap(U)_a$ and $sn(U) \xRightarrow{a^-_{@'}}_{sn} sn(U')$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a - \{(i, @')\}$ and $\forall b \in A, b \neq a.\ dsmap(U')_b = dsmap(U)_b$.

Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T',U') = H(T,U) - \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s',t') \in sn(\mathcal{B})_{H-\{a:(@,@')\}}$.

(4) If $sn(T) \xrightarrow{\dot{\tau}}_{sn} s'$ then, from Prop. B.6 and B.2, we have that there exists $T' \in BL_{dn,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{\dot{\tau}} T'$ and $dsmap(T') = dsmap(T)$. Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{dn,CR}$ such that $U \stackrel{\varepsilon}{\Longrightarrow} U'$ and $T'\,\mathcal{B}\,U'$.
Then, from repeated use of Prop. B.5 and B.2, we derive that $sn(U) \stackrel{\varepsilon}{\Longrightarrow}_{sn} sn(U')$ and $dsmap(U') = dsmap(U)$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, we have $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$. Moreover $H(T',U') = H(T,U)$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s',t') \in sn(\mathcal{B})_H$.

□

Definition B.10. Let $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ be a weak ST bisimulation family. The corresponding binary relation $dn(\mathbf{B}) \subseteq BL_{dn,CR} \times BL_{dn,CR}$ among states of a dynamic name ST semantic model is defined as follows:

$$dn(\mathbf{B}) = \{(T,U) \in BL_{dn,CR} \times BL_{dn,CR} \mid (\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)) \wedge sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)\}$$

■

LEMMA B.11. *Given a weak ST bisimulation family* $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$, *we have that* $dn(\mathbf{B})$ *is a weak bisimulation [Milner 1989a].*

PROOF. First of all let us observe that $dn(\mathbf{B})$ is a symmetric binary relation because $\mathbf{B}$ is a symmetric $\mathcal{H}$-indexed family of binary relations.
Supposed that $(T,U) \in dn(\mathbf{B})$ we show that:

$$T \xrightarrow{\theta} T' \text{ implies } \exists U' \in BL_{dn,CR}.\ U \stackrel{\hat{\theta}}{\Longrightarrow} U' \wedge (T',U') \in dn(\mathbf{B})$$

where $\hat{\theta} = \theta$ if $\theta \in STA_{dn}$ and $\hat{\theta} = \varepsilon$ if $\theta = \dot{\tau}$ [Milner 1989a].
We have three cases depending on whether θ is an action start, an action termination or a silent transition:

—If $T \xrightarrow{a_i^+} T'$ then, from Prop. B.5 and B.2, we have that there exists $@ \in GLoc$ such that $sn(T) \xrightarrow{a_@^+}_{sn} sn(T')$. Moreover, we have $dsmap(T')_a = dsmap(T)_a \cup \{(i,@)\}$, with $(i,@) \notin dsmap(T)_a$, and $\forall b \in A, b \neq a.\, dsmap(T')_b = dsmap(T)_b$.
Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $sn(U) \stackrel{a_{@'}^+}{\Longrightarrow}_{sn} s$ and $sn(T')\,\mathcal{B}_{H(T,U)\cup\{a:(@,@')\}}\,s$.

Then, from repeated use of Prop. B.6 and B.2, we derive that there exist $j \in \mathbf{N}^+$ and $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $U \stackrel{a_j^+}{\Longrightarrow} U'$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a \cup \{(j, @')\}$, with $(j, @') \notin dsmap(U)_a$, and $\forall b \in A, b \neq a.\, dsmap(U')_b = dsmap(U)_b$.
Since, from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from repeated use of Prop. B.2, we derive that $i = j$. Moreover, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) \cup \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

—If $T \xrightarrow{a_i^-} T'$ then, from Prop. B.5 and B.2, we have that there exists $@ \in GLoc$ such that $(i, @) \in dsmap(T)_a$ and $sn(T) \xrightarrow{a_@^-}_{sn} sn(T')$. Moreover, we have $dsmap(T')_a = dsmap(T)_a - \{(i, @)\}$ and $\forall b \in A, b \neq a.\, dsmap(T')_b = dsmap(T)_b$.
Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ and $@' \in GLoc$, with $a : (@, @') \in H(T, U)$, such that $sn(U) \stackrel{a_{@'}^-}{\Longrightarrow}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)-\{a:(@,@')\}}\, s$.
Then, from repeated use of Prop. B.6 and B.2, we derive that there exist $j \in \mathbf{N}^+$ and $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $(j, @') \in dsmap(U)_a$ and $U \stackrel{a_j^-}{\Longrightarrow} U'$. Moreover, we derive $dsmap(U')_a = dsmap(U)_a - \{(j, @')\}$ and $\forall b \in A, b \neq a.\, dsmap(U')_b = dsmap(U)_b$.
Since $a : (@, @') \in H(T, U)$, from $(i, @) \in dsmap(T)_a$ and $(j, @') \in dsmap(U)_a$ we derive $i = j$. Moreover, since from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, from the relationship between $dsmap(T')$ and $dsmap(T)$, and the relationship between $dsmap(U')$ and $dsmap(U)$ that we derived, we have: $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$ and $H(T', U') = H(T, U) - \{a : (@, @')\}$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

—If $T \xrightarrow{\dot{\tau}} T'$ then, from Prop. B.5 and B.2, we have that $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$ and $dsmap(T') = dsmap(T)$.
Since, from the definition of $dn(\mathbf{B})$, we know that $sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ such that $sn(U) \stackrel{\varepsilon}{\Longrightarrow}_{sn} s$ and $sn(T')\, \mathcal{B}_{H(T,U)}\, s$.
Then, from repeated use of Prop. B.6 and B.2, we derive that there exists $U' \in BL_{dn,CR}$, with $s = sn(U')$, such that $U \stackrel{\varepsilon}{\Longrightarrow} U'$ and $dsmap(U') = dsmap(U)$.
Since from the definition of $dn(\mathbf{B})$, we know that $\forall a \in A.\, dom(dsmap(T)_a) = dom(dsmap(U)_a)$, we have $\forall a \in A.\, dom(dsmap(T')_a) = dom(dsmap(U')_a)$. Moreover $H(T', U') = H(T, U)$. Therefore $sn(T')\, \mathcal{B}_{H(T',U')}\, sn(U')$ and, by definition of $dn(\mathbf{B})$, we have $(T', U') \in dn(\mathbf{B})$.

□

The proof of Theorem 4.3, i.e. given $P, Q \in BL_C$ we have $P \simeq_{dn} Q$ if and only if $P \simeq_{sn} Q$, is now straightforward.

PROOF OF THEOREM 4.3.

Proof of "$\Longrightarrow$" Let us suppose that $\langle P, \emptyset \rangle \xrightarrow{a^+_@}_{sn} s$. We have to show that $\exists t \in \Sigma_R, @' \in GLoc.\ \langle Q, \emptyset \rangle \xRightarrow{a^+_{@'}} t \wedge (s, t) \in \approx_{sn,\{a:(@,@')\}}$.

From Prop. B.6 and B.2, we have that there exist $i \in \mathbf{N}^+$ and $T \in BL_{dn,CR}$, with $s = sn(T)$, such that $P \xrightarrow{a^+_i} T$. Moreover, we have $dsmap(T)_a = \{(i, @)\}$ and $\forall b \in A, b \neq a.\ dsmap(T)_b = \emptyset$.

From the definition of observational congruence [Milner 1989a] we derive that there $\exists U \in BL_{dn,CR}$ such that $Q \xRightarrow{a^+_i} U$ and $(T, U) \in \approx_{dn}$, where $\approx_{dn}$ denotes weak bisimulation equivalence [Milner 1989a], hence there exists a weak bisimulation $\mathcal{B}$ such that $T\,\mathcal{B}\,U$.

Then, from repeated use of Prop. B.5 and B.2, we derive that there exists $@' \in GLoc$ such that $Q \xRightarrow{a^+_{@'}}_{sn} sn(U)$. Moreover, we derive $dsmap(U)_a = \{(i, @')\}$ and $\forall b \in A, b \neq a.\ dsmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$ and $H(T, U) = \{a : (@, @')\}$, $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family such that, defined $t = sn(U)$, $(s, t) \in sn(\mathcal{B})_{\{a:(@,@')\}}$, hence $(s, t) \in \approx_{sn,\{a:(@,@')\}}$.

In the case of a start move of $\langle Q, \emptyset \rangle$ or a silent move of $\langle P, \emptyset \rangle$ or $\langle Q, \emptyset \rangle$ the proof is similar.

Proof of "$\Longleftarrow$" Let us suppose that $P \xrightarrow{a^+_i} T$. We have to show that $\exists U \in BL_{dn,CR}.\ Q \xRightarrow{a^+_i} U \wedge (T, U) \in \approx_{dn}$, where $\approx_{dn}$ denotes weak bisimulation equivalence [Milner 1989a].

From Prop. B.5 and B.2, we have that there exists $@ \in GLoc$ such that $\langle P, \emptyset \rangle \xrightarrow{a^+_@}_{sn} sn(T)$. Moreover, we have $dsmap(T)_a = \{(i, @)\}$ and $\forall b \in A, b \neq a.\ dsmap(T)_b = \emptyset$.

From the definition of (static name) observational ST congruence we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $\langle Q, \emptyset \rangle \xRightarrow{a^+_{@'}}_{sn} s$ and $(sn(T), s) \in \approx_{sn,\{a:(@,@')\}}$, hence there exists a weak ST bisimulation family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ such that $(sn(T), s) \in \mathcal{B}_{\{a:(@,@')\}}$.

Then, from repeated use of Prop. B.6 and B.2, we derive that there exists $j \in \mathbf{N}^+$ and $U \in BL_{dn,CR}$, with $s = sn(U)$, such that $Q \xRightarrow{a^+_j} U$. Moreover, we derive $dsmap(U)_a = \{(j, @')\}$ and $\forall b \in A, b \neq a.\ dsmap(U)_b = \emptyset$.

Since $\forall a \in A.\ dom(dsmap(P)_a) = dom(dsmap(Q)_a) = \emptyset$, from repeated use of Prop. B.2, we derive that $i = j = 1$. Finally, since we have $\forall a \in A.\ dom(dsmap(T)_a) = dom(dsmap(U)_a)$ and $H(T, U) = \{a : (@, @')\}$, $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$ and $dn(\mathbf{B})$ is a weak bisimulation [Milner 1989a] such that $(T, U) \in dn(\mathbf{B})$, hence $(T, U) \in \approx_{dn}$.

In the case of a start move of Q or a silent move of P or Q the proof is similar.

□

C. EQUIVALENCE OF THE STACK AND STATIC NAME TECHNIQUES

In this appendix we will provide the proof of Theorem 6.6, i.e. the equivalence of ST bisimulation expressed via the stack technique with the definition of ST bisimulation based on static names. Similarly as in Appendix B, we will denote by $\longrightarrow_{sn}$ the state transitions of the static name operational semantics in order to distinguish them from those of the stack technique operational semantics (simply denoted by $\longrightarrow$). Since Theorem 6.6 compares the stack and the static name approaches over processes of BL_C, in this appendix we will just consider the stack technique applied to a simple language with prefixing instead of general sequential composition. Therefore we will consider a set of stack technique state terms BL_s generated by a syntax with prefixing like that of state terms of the dynamic name technique ($BL_{s,C}$ as usual denotes the set of processes of BL_s). Moreover we will consider the stack technique transition system to be generated by the set of operational rules in Table VII, where we perform the obvious translation of the rules for sequential composition into those for prefixing and we do not consider the elimination rules for the parallel operator (we just treat processes which, after performing an a_i^-, become terminated processes in the same way as we do for those which do not terminate).

We denote by $BL_{s,CR}$ the subset of $BL_{s,C}$ of the state terms reachable from a process of BL_C, i.e. $BL_{s,CR} = \{T \in BL_{s,C} \mid \exists P \in BL_C, k \in \mathbb{N}, \{T_i\}_{i<k}, \{\theta_i\}_{i\leq k}. P \xrightarrow{\theta_1} T_1 \xrightarrow{\theta_2} \dots \xrightarrow{\theta_{k-1}} T_{k-1} \xrightarrow{\theta_k} T\}$.

The proof of Theorem 6.6 is based on a mapping *ssmap* between the stack positions and the static names assumed by the actions of a process according to the ST semantics via stack technique and static name ST semantics, respectively. An association between the stack position $i \in \mathbb{N}^+$ and the static name $@ \in GLoc$ assumed by an action a according to the two semantics is represented inside *ssmap* by the triple $(a, i, @)$. In the following we use $a : (i, @)$ to stand for $(a, i, @) \in ssmap$.

We denote by W a *stack-static association string.* W ranges over the set $SSAS$ of partial bijections from $\mathbb{N}^+$ to $GLoc$ such that $\exists k \geq 1.\, dom(W) = \{1 \dots k\}$, i.e. non-empty strings over the alphabet of global locations $GLoc$ without multiple occurrences of the same global location. A stack-static association string W can be seen as a set whose elements are associations $(i, @)$. A *stack-static mapping ssmap* is a relation from A to $\mathbb{N}^+ \times GLoc$ such that $\forall a \in A.\, ssmap_a \in SSAS$, i.e. *ssmap* is a set including an independent stack-static association string for each different action type. We denote by $SSMAP$ the set of all stack-static mappings.

Definition C.1. The function $ssmap : BL_{s,CR} \longrightarrow SSMAP$, which evaluates a stack-static mapping from a state term of stack technique ST semantic model, is defined as follows:

—$ssmap(\underline{0}) = ssmap(\mu.P) = ssmap(\dot{\tau}.P) = ssmap(P + Q) = ssmap(recX.P) = \emptyset$

—$ssmap(a_1^-.P) = \{a : (1, \bullet)\}$

—$ssmap(T \parallel_{S,M} U) =$
$\{a : (j, l@) \mid a \notin S \wedge a : (i, @) \in ssmap(T) \wedge j = \#_{i,l}(M_a)\} \cup$

$\{a : (j, r@) \mid a \notin S \wedge a : (i, @) \in ssmap(U) \wedge j = \#_{i,r}(M_a)\} \cup$
$\{a : (i, <@'|@''>) \mid a \in S \wedge a : (i, @') \in ssmap(T) \wedge a : (i, @'') \in ssmap(U)\}$

—$ssmap(T/L) = \{a : (i, @) \in ssmap(T) \mid a \notin L\}$ ■

The following proposition determine how the stack-static mapping $ssmap(T)$ changes during the evolution of a stack technique state T in the three cases of an action start, an action termination and a silent transition.

PROPOSITION C.2. *Let T be a state of $BL_{s,CR}$. We have:*

—*If $T \xrightarrow{a^+} T'$ there exists $@ \in GLoc$ such that $@ \notin range(ssmap(T)_a)$ and $ssmap(T')_a = @ssmap(T)_a$. Moreover $ssmap(T')_b = ssmap(T)_b$ for all $b \in A$ such that $b \neq a$.*

—*If $T \xrightarrow{a_i^-} T'$ then there exists $@ \in GLoc$ such that $(i, @) \in ssmap(T)_a$ and $ssmap(T')_a = ssmap(T)_a \ominus i$. Moreover $ssmap(T')_b = ssmap(T)_b$ for all $b \in A$ such that $b \neq a$.*

—*If $T \xrightarrow{\dot{\tau}} T'$ then $ssmap(T') = ssmap(T)$.*

PROOF. By induction on the derivation of a transition $T \xrightarrow{\theta} T'$, by showing that whenever all the premises of a stack technique operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T, the conclusion satisfies it. □

The state of the static name semantics corresponding to a given state of the semantics via the stack technique is derived, as in Appendix B, as follows.

Definition C.3. The function $CL : BL_{s,CR} \longrightarrow BL_{sn}$, where CL stands for "clean", is defined as follows. Given $T \in BL_{s,CR}$, $CL(T)$ is obtained by syntactically replacing each ST action a_1^- occurring in T with a^- and by discarding the sets M in the operators $\|_{S,M}$. ■

Definition C.4. The function $sn : BL_{s,CR} \longrightarrow \Sigma_R$, which evaluates the state $sn(T)$ of the static name ST semantics corresponding to a state T of the ST semantics via the stack technique, is defined as follows:

$$sn(T) = \langle CL(T), \bigcup_{a \in A} \{a : @ \mid @ \in range(ssmap(T)_a)\}\rangle$$

■

As in Appendix B, another preliminary result which is needed before proving the equivalence of the stack and static name approaches is to establish a correspondence between the stack and static name transitions.

PROPOSITION C.5. *Let T be a state of $BL_{s,CR}$. We have:*

—$T \xrightarrow{a^+} T'$ *implies* $\exists @ \in GLoc.\ ssmap(T')_a = @\,ssmap(T)_a \wedge sn(T) \xrightarrow{a^+_@}_{sn} sn(T')$

—$T \xrightarrow{a_i^-} T'$ *implies* $\exists @ \in GLoc.\ ssmap(T')_a = ssmap(T)_a \ominus i \wedge sn(T) \xrightarrow{a^-_@}_{sn} sn(T')$

—$T \xrightarrow{\dot\tau} T'$ *implies* $sn(T) \xrightarrow{\dot\tau}_{sn} sn(T')$

PROOF. By induction on the derivation of a stack technique transition from T by showing that whenever all the premises of a stack technique operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T and static name transitions are defined by structural semantics, the conclusion satisfies it. □

PROPOSITION C.6. *Let T be a state of $BL_{s,CR}$. We have:*

—$sn(T) \xrightarrow{a^+_@}_{sn} s$ *implies* $\exists T' \in BL_{s,CR}.\ T \xrightarrow{a^+} T' \wedge s = sn(T') \wedge ssmap(T')_a = @\, ssmap(T)_a$

—$sn(T) \xrightarrow{a^-_@}_{sn} s$ *implies* $\exists i \in \mathbb{N}^+, T' \in BL_{s,CR}.\ T \xrightarrow{a^-_i} T' \wedge s = sn(T') \wedge ssmap(T')_a = ssmap(T)_a \ominus i$

—$sn(T) \xrightarrow{\dot\tau}_{sn} s$ *implies* $\exists T' \in BL_{s,CR}.\ T \xrightarrow{\dot\tau} T' \wedge s = sn(T')$

PROOF. By induction on the derivation of a static name transition from $sn(T)$ by showing that whenever all the premises of a static name operational rule satisfy the proposition, then, exploiting the fact that $ssmap(T)$ is defined by structural induction over T and stack technique transitions are defined by structural semantics, the conclusion satisfies it. □

Finally, we introduce function $H(T, U)$ which computes a (static name) association history H from two stack technique states T and U which are put in correspondence by a stack technique equivalence.

Definition C.7. The function $H : BL_{s,CR} \times BL_{s,CR} \longrightarrow \mathcal{H}$, which evaluates an association history H between the global locations of the actions in execution of a pair of stack technique state terms (T, U) such that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, i.e. the number of stack positions of the actions in execution in T and U is the same, is defined as follows:

$$H(T, U) = \{a : (@', @'') \mid a \in A \wedge ssmap(T)^{-1}_a(@') = ssmap(U)^{-1}_a(@'')\}$$ ■

Now we are in a position to present the two main lemmas needed for the proof of Theorem 6.6. The first lemma shows that from a stack technique ST bisimulation we can derive a static name ST bisimulation, the second lemma shows the converse.

Definition C.8. Let $\mathcal{B}$ be a weak bisimulation [Milner 1989a] over stack technique state terms of $BL_{s,C}$. The corresponding $\mathcal{H}$-indexed family of binary relations $\mathbf{sn}(\mathcal{B}) = \{sn(\mathcal{B})_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ among states of a static name ST semantic model is defined as follows:

$$sn(\mathcal{B})_H = \{(sn(T), sn(U)) \mid T, U \in BL_{s,CR} \wedge (\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)) \wedge T\,\mathcal{B}\,U \wedge H(T, U) = H\}$$ ■

LEMMA C.9. *Given a weak bisimulation $\mathcal{B}$ [Milner 1989a] over stack technique state terms of $BL_{s,C}$, we have that $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family.*

PROOF. First of all let us observe that $\mathbf{sn}(\mathcal{B})$ is a a symmetric $\mathcal{H}$-indexed family of binary relations over Σ_R because $\mathcal{B}$ is a symmetric binary relation.

Supposed that $H \in \mathcal{H}$ and $(s,t) \in sn(\mathcal{B})_H$ we show that:

(1) H is compatible with (s,t)

(2) $s \xrightarrow{a^+_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ t \xRightarrow{a^+_{@'}} t' \wedge (s',t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$

(3) $s \xrightarrow{a^-_@} s'$ implies $\exists t' \in \Sigma_R, @' \in GLoc.\ a : (@,@') \in H \wedge t \xRightarrow{a^-_{@'}} t' \wedge$
$(s',t') \in sn(\mathcal{B})_{H - \{a:(@,@')\}}$

(4) $s \xrightarrow{\dot{\tau}} s'$ implies $\exists t' \in \Sigma_R.\ t \xRightarrow{\varepsilon} t' \wedge (s',t') \in sn(\mathcal{B})_H$

In the following we will separately prove each item. Note that, by definition of $\mathbf{sn}(\mathcal{B})$, we have that there exist $T, U \in BL_{s,CR}$ such that $s = sn(T)$ and $t = sn(U)$.

(1) Since, by definition of $\mathbf{sn}(\mathcal{B})$, we have that $H(T,U) = H$, then $dom(H_a) = range(ssmap(T)_a)$ and $range(H_a) = range(ssmap(U)_a)$, hence H is compatible with $(sn(T), sn(U))$.

(2) If $sn(T) \xrightarrow{a^+_@}_{sn} s'$ then, from Prop. C.6 and C.2, we have that there exists $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{a^+} T'$. Moreover, we have $ssmap(T')_a = @\, ssmap(T)_a$, with $@ \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\ ssmap(T')_b = ssmap(T)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{s,CR}$ such that $U \xRightarrow{a^+} U'$ and $T'\,\mathcal{B}\,U'$.
Then, from repeated use of Prop. C.5 and C.2, we derive that there exists $@' \in GLoc$ such that $sn(U) \xRightarrow{a^+_{@'}}_{sn} sn(U')$. Moreover, we derive $ssmap(U')_a = @\, ssmap(U)_a\}$, with $@ \notin range(ssmap(U)_a)$, and $\forall b \in A, b \neq a.\ ssmap(U')_b = ssmap(U)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\ dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T',U') = H(T,U) \cup \{a : (@,@')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s',t') \in sn(\mathcal{B})_{H \cup \{a:(@,@')\}}$.

(3) If $sn(T) \xrightarrow{a^-_@}_{sn} s'$ then, from Prop. C.6 and C.2, we have that there exist $i \in \mathbf{N}^+$ and $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $(i,@) \in ssmap(T)_a$ and $T \xrightarrow{a^-_i} T'$. Moreover, we have $ssmap(T')_a = ssmap(T)_a \ominus i$ and $\forall b \in A, b \neq a.\ ssmap(T')_b = ssmap(T)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T\,\mathcal{B}\,U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{s,CR}$ such that $U \xRightarrow{a^-_i} U'$ and $T'\,\mathcal{B}\,U'$.

Then, from repeated use of Prop. C.5 and C.2, we derive that there exists $@' \in GLoc$ such that $(j, @') \in ssmap(U)_a$ and $sn(U) \overset{a^-_{@'}}{\Longrightarrow}_{sn} sn(U')$. Moreover, we derive $ssmap(U')_a = ssmap(U)_a \ominus i$ and $\forall b \in A, b \neq a.\, ssmap(U')_b = ssmap(U)_b$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T', U') = H(T, U) - \{a : (@, @')\}$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_{H-\{a:(@,@')\}}$.

(4) If $sn(T) \xrightarrow{\dot{\tau}}_{sn} s'$ then, from Prop. C.6 and C.2, we have that there exists $T' \in BL_{s,CR}$, with $s' = sn(T')$, such that $T \xrightarrow{\dot{\tau}} T'$ and $ssmap(T') = ssmap(T)$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$ we know that $T \,\mathcal{B}\, U$, from the definition of weak bisimulation [Milner 1989a] we derive that there exists $U' \in BL_{s,CR}$ such that $U \overset{\varepsilon}{\Longrightarrow} U'$ and $T' \,\mathcal{B}\, U'$.
Then, from repeated use of Prop. C.5 and C.2, we derive that $sn(U) \overset{\varepsilon}{\Longrightarrow}_{sn} sn(U')$ and $ssmap(U') = ssmap(U)$.
Since, from the definition of $\mathbf{sn}(\mathcal{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, we have $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$. Moreover $H(T', U') = H(T, U)$. Therefore, by definition of $\mathbf{sn}(\mathcal{B})$, we have that, defined $t' = sn(U')$, $(s', t') \in sn(\mathcal{B})_H$.

□

Definition C.10. Let $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ be a weak ST bisimulation family. The corresponding binary relation $s(\mathbf{B}) \subseteq BL_{s,CR} \times BL_{s,CR}$ among states of a stack technique ST semantic model is defined as follows:

$$s(\mathbf{B}) = \{(T, U) \in BL_{s,CR} \times BL_{s,CR} \mid (\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)) \wedge sn(T)\, \mathcal{B}_{H(T,U)}\, sn(U)\}$$

■

LEMMA C.11. *Given a weak ST bisimulation family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$, we have that $s(\mathbf{B})$ is a weak bisimulation [Milner 1989a].*

PROOF. First of all let us observe that $s(\mathbf{B})$ is a symmetric binary relation because $\mathbf{B}$ is a symmetric $\mathcal{H}$-indexed family of binary relations.
Supposed that $(T, U) \in s(\mathbf{B})$ we show that:

$$T \xrightarrow{\theta} T' \text{ implies } \exists U' \in BL_{s,CR}.\, U \overset{\hat{\theta}}{\Longrightarrow} U' \wedge (T', U') \in s(\mathbf{B})$$

where $\hat{\theta} = \theta$ if $\theta \in STA_s$ and $\hat{\theta} = \varepsilon$ if $\theta = \dot{\tau}$ [Milner 1989a].
We have three cases depending on whether θ is an action start, an action termination or a silent transition:

—If $T \xrightarrow{a^+} T'$ then, from Prop. C.5 and C.2, we have that there exists $@ \in GLoc$ such that $sn(T) \xrightarrow{a^+_@}_{sn} sn(T')$. Moreover, we have $ssmap(T')_a = @\, ssmap(T)_a$, with $@ \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\, ssmap(T')_b = ssmap(T)_b$.

Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $sn(U) \overset{a^+_{@'}}{\Longrightarrow}_{sn} s$ and $sn(T')\,\mathcal{B}_{H(T,U)\cup\{a:(@,@')\}}\,s$.
Then, from repeated use of Prop. C.6 and C.2, we derive that there exists $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $U \overset{a^+}{\Longrightarrow} U'$. Moreover, we derive $ssmap(U')_a = @'\,ssmap(U)_a$, with $@' \notin range(ssmap(T)_a)$, and $\forall b \in A, b \neq a.\, ssmap(U')_b = ssmap(U)_b$.
Since, from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T',U') = H(T,U) \cup \{a : (@,@')\}$. Therefore $sn(T')\,\mathcal{B}_{H(T',U')}\,sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

—If $T \xrightarrow{a^-_i} T'$ then, from Prop. C.5 and C.2, we have that there exists $@ \in GLoc$ such that $(i,@) \in ssmap(T)_a$ and $sn(T) \xrightarrow{a^-_@}_{sn} sn(T')$. Moreover, we have $ssmap(T')_a = ssmap(T)_a \ominus i$ and $\forall b \in A, b \neq a.\, ssmap(T')_b = ssmap(T)_b$.
Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$, from the definition of weak ST bisimulation family we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$, with $a : (@,@') \in H(T,U)$, such that $sn(U) \overset{a^-_{@'}}{\Longrightarrow}_{sn} s$ and $sn(T')\,\mathcal{B}_{H(T,U)-\{a:(@,@')\}}\,s$.
Then, from repeated use of Prop. C.6 and C.2, we derive that there exist $j \in \mathbf{N}^+$ and $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $(j,@') \in ssmap(U)_a$ and $U \overset{a^-_j}{\Longrightarrow} U'$. Moreover, we derive $ssmap(U')_a = ssmap(U)_a \ominus j$ and $\forall b \in A, b \neq a.\, ssmap(U')_b = ssmap(U)_b$.
Since $a : (@,@') \in H(T,U)$, from $(i,@) \in ssmap(T)_a$ and $(j,@') \in ssmap(U)_a$ we derive $i = j$. Moreover, since from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, from the relationship between $ssmap(T')$ and $ssmap(T)$, and the relationship between $ssmap(U')$ and $ssmap(U)$ that we derived, we have: $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$ and $H(T',U') = H(T,U) - \{a : (@,@')\}$. Therefore $sn(T')\,\mathcal{B}_{H(T',U')}\,sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

—If $T \xrightarrow{\dot{\tau}} T'$ then, from Prop. C.5 and C.2, we have that $sn(T) \xrightarrow{\dot{\tau}}_{sn} sn(T')$ and $ssmap(T') = ssmap(T)$.
Since, from the definition of $s(\mathbf{B})$, we know that $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$, from the definition of weak ST bisimulation family we derive that there exists $s \in \Sigma_R$ such that $sn(U) \overset{\varepsilon}{\Longrightarrow}_{sn} s$ and $sn(T')\,\mathcal{B}_{H(T,U)}\,s$.
Then, from repeated use of Prop. C.6 and C.2, we derive that there exists $U' \in BL_{s,CR}$, with $s = sn(U')$, such that $U \overset{\varepsilon}{\Longrightarrow} U'$ and $ssmap(U') = ssmap(U)$.
Since from the definition of $s(\mathbf{B})$, we know that $\forall a \in A.\, dom(ssmap(T)_a) = dom(ssmap(U)_a)$, we have $\forall a \in A.\, dom(ssmap(T')_a) = dom(ssmap(U')_a)$. Moreover $H(T',U') = H(T,U)$. Therefore $sn(T')\,\mathcal{B}_{H(T',U')}\,sn(U')$ and, by definition of $s(\mathbf{B})$, we have $(T',U') \in s(\mathbf{B})$.

$\square$

The proof of Theorem 6.6, i.e. given $P, Q \in BL_C$ we have $P \simeq_s Q$ if and only if $P \simeq_{sn} Q$, is now straightforward.

PROOF OF THEOREM 6.6.

Proof of "$\Longrightarrow$" Let us suppose that $\langle P, \emptyset \rangle \xrightarrow{a^+_{@}}_{sn} s$. We have to show that $\exists t \in \Sigma_R, @' \in GLoc.\ \langle Q, \emptyset \rangle \xRightarrow{a^+_{@'}} t \wedge (s,t) \in \approx_{sn,\{a:(@,@')\}}$.

From Prop. C.6 and C.2, we have that there exists $T \in BL_{s,CR}$, with $s = sn(T)$, such that $P \xrightarrow{a^+} T$. Moreover, we have $ssmap(T)_a = \{(1, @)\}$ and $\forall b \in A, b \neq a.\ ssmap(T)_b = \emptyset$.

From the definition of observational congruence [Milner 1989a] we derive that there $\exists U \in BL_{s,CR}$ such that $Q \xRightarrow{a^+} U$ and $(T,U) \in \approx_s$, where $\approx_s$ denotes weak bisimulation equivalence [Milner 1989a], hence there exists a weak bisimulation $\mathcal{B}$ such that $T\,\mathcal{B}\,U$.

Then, from repeated use of Prop. C.5 and C.2, we derive that there exists $@' \in GLoc$ such that $Q \xRightarrow{a^+_{@'}}_{sn} sn(U)$. Moreover, we derive $ssmap(U)_a = \{(1, @')\}$ and $\forall b \in A, b \neq a.\ ssmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$ and $H(T,U) = \{a : (@, @')\}$, $\mathbf{sn}(\mathcal{B})$ is a weak ST bisimulation family such that, defined $t = sn(U)$, $(s,t) \in sn(\mathcal{B})_{\{a:(@,@')\}}$, hence $(s,t) \in \approx_{sn,\{a:(@,@')\}}$.

In the case of a start move of $\langle Q, \emptyset \rangle$ or a silent move of $\langle P, \emptyset \rangle$ or $\langle Q, \emptyset \rangle$ the proof is similar.

Proof of "$\Longleftarrow$" Let us suppose that $P \xrightarrow{a^+} T$. We have to show that $\exists U \in BL_{s,CR}.\ Q \xRightarrow{a^+} U \wedge (T,U) \in \approx_s$, where $\approx_s$ denotes weak bisimulation equivalence [Milner 1989a].

From Prop. C.5 and C.2, we have that there exists $@ \in GLoc$ such that $\langle P, \emptyset \rangle \xrightarrow{a^+_{@}}_{sn} sn(T)$. Moreover, we have $ssmap(T)_a = \{(1, @)\}$ and $\forall b \in A, b \neq a.\ ssmap(T)_b = \emptyset$.

From the definition of (static name) observational ST congruence we derive that there exist $s \in \Sigma_R$ and $@' \in GLoc$ such that $\langle Q, \emptyset \rangle \xRightarrow{a^+_{@'}}_{sn} s$ and $(sn(T), s) \in \approx_{sn,\{a:(@,@')\}}$, hence there exists a weak ST bisimulation family $\mathbf{B} = \{\mathcal{B}_H \subseteq \Sigma_R \times \Sigma_R \mid H \in \mathcal{H}\}$ such that $(sn(T), s) \in \mathcal{B}_{\{a:(@,@')\}}$.

Then, from repeated use of Prop. C.6 and C.2, we derive that there exists $U \in BL_{s,CR}$, with $s = sn(U)$, such that $Q \xRightarrow{a^+} U$. Moreover, we derive $ssmap(U)_a = \{(1, @')\}$ and $\forall b \in A, b \neq a.\ ssmap(U)_b = \emptyset$.

Finally, since we have $\forall a \in A.\ dom(ssmap(T)_a) = dom(ssmap(U)_a)$ and $H(T,U) = \{a : (@, @')\}$, $sn(T)\,\mathcal{B}_{H(T,U)}\,sn(U)$ and $s(\mathbf{B})$ is a weak bisimulation [Milner 1989a] such that $(T,U) \in s(\mathbf{B})$, hence $(T,U) \in \approx_s$.

In the case of a start move of Q or a silent move of P or Q the proof is similar.

□

REFERENCES

ACETO, L. 1994a. On "axiomatising finite concurrent processes". *SIAM Journal on Computing 23,* 4, 852–863.

ACETO, L. 1994b. A static view of localities. *Formal Aspects of Computing 6*, 201–222.

ACETO, L. AND HENNESSY, M. 1993. Towards action refinement in process algebras. *Information and Computation 103*, 204–269.

ACETO, L. AND HENNESSY, M. 1994. Adding action refinement to a finite process algebra. *Information and Computation 115*, 179–247.

BAETEN, J. C. M. AND WEIJLAND, W. P. 1990. *Process Algebra.* Cambridge University Press.

BERGSTRA, J. A. AND KLOP, J. W. 1985. Algebra of communicating processes with abstraction. *Theoretical Computer Science 37,* 1, 77–121.

BOUDOL, G., CASTELLANI, I., HENNESSY, M., AND KIEHN, A. 1994. A theory of processes with localities. *Formal Aspects of Computing 6,* 2, 165–200.

BRAVETTI, M., BERNARDO, M., AND GORRIERI, R. 1998. Towards performance evaluation with general distributions in process algebras. In *Proc. of the 9th Int. Conf. on Concurrency Theory (CONCUR '98), LNCS 1466.* Nice (France), 405–422.

BRAVETTI, M. AND GORRIERI, R. 1999. Axiomatizing st-bisimulation for a process algebra with recursion and action refinement (extended abstract). In *Proc. of the 6th Int. Workshop on Expressiveness in Concurrency (EXPRESS '99), ENTCS 27.* Eindhoven (The Netherlands).

BRAVETTI, M. AND GORRIERI, R. 2001. The theory of interactive generalized semi-markov processes. *Theoretical Computer Science.* To appear.

BUSI, N., VAN GLABBEEK, R. J., AND GORRIERI, R. 1994. Axiomatising st-bisimulation equivalence. In *Proc. of the IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '94).* S. Miniato (Italy), 169–188.

CASTELLANI, I. 1995. Observing distribution in processes: Static and dynamic localities. *Int. Journal of Foundations of Computer Science 6*, 353–393.

DARONDEAU, P. AND DEGANO, P. 1989. Causal trees. *Automata, Languages and Programming LNCS 372*, 234–248.

DEGANO, P., DE NICOLA, R., AND MONTANARI, U. 1989. Partial ordering description of nondeterministic concurrent systems. In *Proc. of the REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354.* Noordwijkerhout, 438–466.

DEGANO, P. AND GORRIERI, R. 1995. A causal operational semantics of action refinement. *Information and Computation 122*, 97–119.

DEVILLERS, R. 1992. Maximality preserving bisimulation. *Theoretical Computer Science 102*, 165–183.

FRÖSCHLE, S. B. 1999. Decidability of plain and hereditary history-preserving bisimilarity for bpp. In *Proc. of the 6th Int. Workshop on Expressiveness in Concurrency (EXPRESS '99), ENTCS 27.* Eindhoven (The Netherlands).

FRÖSCHLE, S. B. AND HILDEBRANDT, T. T. 1999. On plain and hereditary history-preserving bisimulation. In *Proc. of the 24th Int. Symp. on Mathematical Foundations of Computer Science (MFCS '99), LNCS 1672.* Szklarska Poreba (Poland), 354–365.

GORRIERI, R. AND LANEVE, C. 1991. The limit of split_n bisimulations for ccs agents. In *Proc. of the Int. Symp. on Mathematical Foundations of Computer Science (MFCS '91), LNCS 520.* 170–180.

GORRIERI, R. AND LANEVE, C. 1992. An extensional formulation of st bisimulation. In *Proc. of the 4th Italian Conference on Theoretical Computer Science (ICTCS '92)*, P. M. A. Marchetti Spaccamela and M. V. Zilli, Eds. World Scientific, L'Aquila (Italy), 239–253.

GORRIERI, R. AND LANEVE, C. 1995. Split and st bisimulation semantics. *Information and Computation 118*, 272–288.

GORRIERI, R. AND RENSINK, A. 2001. *Handbook of Process Algebra.* Elsevier, Chapter Action Refinement.

HENNESSY, M. 1991. A proof system for weak st-bisimulation over a finite process algebra. Tech. Rep. 6/91, University of Sussex (UK).

HOARE, C. A. R. 1985. *Communicating Sequential Processes.* Prentice Hall.

JURDZIŃSKI, M. AND NIELSEN, M. 2000. Hereditary history preserving bisimilarity is undecidable. In *Proc. of the 17th Int. Symp. on Theoretical Aspects of Computer Science (STACS 2000), LNCS 1770.* Lille (France), 358–369.

KIEHN, A. AND HENNESSY, M. 1997. On the decidability of non-interleaving process equivalences. *Fundamenta Informaticae 30,* 1, 18–33.

MILNER, R. 1984. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences 28*, 439–466.

MILNER, R. 1989a. *Communication and Concurrency.* Prentice Hall.

MILNER, R. 1989b. A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation 81*, 227–247.

MONTANARI, U. AND PISTORE, M. 1997. Minimal transition systems for history-preserving bisimulation. In *Proc. of the 14th Symp. on Theoretical Aspects of Computer Science (STACS'97), LNCS 1200.*

MONTANARI, U. AND YANKELEVICH, D. 1995. Location equivalence in parametric setting. *Theoretical Computer Science 149,* 2, 299–332.

PAIGE, R. AND TARJAN, R. E. 1987. Three partition refinement algorithms. *SIAM Journal of Computing 16*, 973–989.

PLOTKIN, G. 1981. A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, Aarhus University, Department of Computer Science.

RABINOVICH, A. AND TRAKHTENBROT, B. 1988. Behaviour structures and nets. *Fundamenta Informaticae 11*, 357–404.

VAN GLABBEEK, R. J. 1990. The refinement theorem for st-bisimulation semantics. In *Proc. of the IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '90).* Sea of Gallilea (Israel), 27–52.

VAN GLABBEEK, R. J. AND GOLTZ, U. 1989. Equivalence notions for concurrent systems and refinement of actions. Arbeitspapiere der GMD 366. Extended abstract in: Proc. of the 14th Int. Symp. on Mathematical Foundations of Computer Science (MFCS '89), A. Kreczmar and G. Mirkowska editors, LNCS 379:237-248, Poræbka-Kozubnik (Poland), 1989.

VAN GLABBEEK, R. J. AND VAANDRAGER, F. W. 1987. Petri net models for algebraic theories of concurrency. In *Proc. of the Conf. on Parallel Architectures and Languages Europe (PARLE '87), LNCS 259.* Eindhoven (The Netherlands), 224–242.

VOGLER, W. 1993. Bisimulation and action refinement. *Theoretical Computer Science 114*, 173–200.

VOGLER, W. 1995. Timed testing of concurrent systems. *Information and Computation 121*, 149–171.

Received November 2000; revised April 2001; accepted June 2001