URL: http://www.elsevier.nl/locate/entcs/volume27.html 20 pages

Axiomatizing ST Bisimulation for a Process Algebra with Recursion and Action Refinement (Extended Abstract) [1]

Mario Bravetti, Roberto Gorrieri

Dipartimento di Scienze dell'Informazione
Università di Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
E-mail: {bravetti, gorrieri}@cs.unibo.it

Abstract

Due to the complex nature of bisimulation equivalences which express some form of history dependence, it turned out to be problematic to axiomatize them for non trivial classes of systems. Here we introduce the idea of "compositional level-wise renaming" which gives rise to the new possibility of axiomatizing the class of history dependent bisimulations with slight modifications to the machinery for standard bisimulation. We propose two techniques, which are based on this idea, in the special case of the ST semantics, defined for terms of a process algebra with recursion. The first technique, which is more intuitive, is based on dynamic names, allowing weak ST bisimulation to be decided and axiomatized for all processes that possess a finite state interleaving semantics. The second technique, which is based on pointers, preserves the possibility of deciding and axiomatizing weak ST bisimulation also when an action refinement operator $P[a \rightsquigarrow Q]$ is considered.

1 Introduction

Many bisimulation equivalences expressing some form of history dependence have been defined in the literature: history preserving bisimulation [24,10,13], ST bisimulation [12,3,25,16,17,8] and location bisimulation [5,2,22,9]. Due to the complex nature of this kind of equivalences it turned out to be not simple to decide (see for instance [18]) and, especially, axiomatize them for non trivial classes of systems (e.g. classes which include most recursive systems).

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

The first approach is based on *static names* [13,12,3,2,9]. According to this approach a unique name is statically assigned (i.e. on the basis of the syntac-

[1] This research has been partially supported by MURST and CNR.

tical structure of the process) to each different "historical element" (e.g. a location in location bisimulation) 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 historical elements of different terms that are considered to be equivalent. An advantage of this approach is that it produces (at least in our formulation of ST semantics of [4] and in the location semantics of [2,9]) 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 easy way to axiomatize these history dependent bisimulations (it is necessary to rely on equality parametrized on associations among 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* [10,16,17,8]. 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 [10,16,17,8] is that for most recursive systems (e.g. for $a \,\|\, recX.b.X$ in [10,16,8]) 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 [21]. 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 [21] 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 more active) are *reused*. Note that this technique is based on names as the approach of [13,12,3,2,9], 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). [2] 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.

[2] The idea of dynamic names already appeared in [5], but here no special technique (like reuse of names) is employed in order to obtain finite models.

Unfortunately the technique developed in [21] is not compositional, in the sense that, in order to produce the history preserving semantic model of a term, first an intermediate semantic model of the whole term must be computed and then transformed by adjusting history information in transitions.

In fact, we could easily develop an axiomatization for a history dependent bisimulation, which is complete over a wide class of processes, if we had a compositional approach for deriving semantic models which encode history with dynamic names (i.e. names dynamically assigned with a fixed rule and reuse of obsolete names). What we need is a structured operational semantics (SOS [23]) which allows to derive, e.g. in the case of the parallel composition operator "$\|$", from the history dependent computations of P and Q, the history dependent computations of $P \,\|\, Q$. As long as we consider only terms with choice, prefix and termination operators, where history dependences 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 [19]. 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 which encode history dependences), the computations of $P \,\|\, Q$ (in the form of prefixes with history dependences).

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 names used for identifying historical elements at the level of $P \,\|\, Q$, generated according to the fixed rule for creating new names, to the names used for identifying the same historical elements inside P or Q (which in general are different) generated according to the same rule. In this way when a future reference to a historical element is made by a computation 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 *level-wise renaming technique* where historical elements 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.

In this paper we tackle this problem for one of these history dependent bisimulations: weak ST bisimulation equivalence. We present two techniques, based on our idea of level-wise renaming, for defining the ST semantics via SOS rules of a language with recursion (for the second technique we consider also a semantic action refinement operator). Both these techniques can be used for deciding ST equivalence via standard bisimulation (namely observational congruence [19]). Moreover they produce finite ST semantic models for all processes that possess a finite state interleaving semantics. As a consequence we show that both techniques can be used for axiomatizing ST bisimulation over the wide class of (interleaving) finite state processes.

With ST semantics, originally defined in [15] over Petri Nets, the execution of an action gives rise to 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 (this is a form of history dependence), even in the situation of auto-concurrency (i.e., multiple actions of the same type being in execution at the same time).

Initially we consider a basic process algebra equipped with a recursion operator and the CSP parallel operator. We show how to define ST semantics for such a language via SOS rules, by employing our idea of level-wise renaming, when we encode history dependence through dynamic names: a different name is dynamically assigned to each action that starts execution and names of terminated actions are reused when new actions start. Hence with this technique, hereafter called *name technique*, the reuse of names is "on demand", i.e. it is delayed until new names need to be activated.

Then 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 level-wise re-mapping of pointers. The main difference with the previous technique is that 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. Moreover we show that with this new pointer-based technique it is possible to solve some problems that arise with the name technique when we extend our language with an action refinement operator $P[a \leadsto Q]$ which performs the semantic refinement of all a executed by P to Q [12,11,14]. In particular with the new technique we have that if both P and Q are finite state processes then $P[a \leadsto Q]$ is finite state. Through the new technique we define the ST semantics in SOS style for an extended language which includes the refinement operator and we produce a complete axiomatization for ST bisimulation over finite state processes.

The paper is organized as follows. In Sect. 2 we briefly present the name technique and we show how it can be used to define ST semantics via SOS rules of a basic process algebra. In Sect. 3 we analyze the defects of the name technique and the decidability problems that arise when an action refinement operator is considered. In Sect. 4 we present the stack technique and we use it to define the ST semantics via SOS rules of a process algebra with action refinement and recursion. In Sect. 5 we present a complete axiomatization for weak ST bisimulation equivalence over finite state processes of such a process algebra. Finally, in Sect. 6 we report some concluding remarks.

The full version of the paper, which includes a detailed presentation of the name technique and proofs of theorems, is available at:

ftp://ftp.cs.unibo.it/pub/techreports/99-01.ps.gz.

2 ST Semantics via the Name Technique

We start by briefly presenting the technique based on dynamic names. Here we consider a simple process algebra with actions taken from a set A, ranged over by $a, b, c, \ldots$, the CCS prefix, choice and recursion operators, and the CSP parallel composition operator "$\|_S$", where synchronization over actions of type S is required.

The *name technique* is based on the idea of dynamically assigning, at the semantic level, a new name to each action that starts execution. *Names* are indices $i \in \mathbb{N}$ 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 indices 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 name i uniquely determines which action a is terminating.

In order to express this behavior compositionally it is necessary to parameterize in state terms each parallel operator with a mapping M. For every action a started in $P \|_{S,M} Q$, M records the association between the name i, generated according to the rule above for identifying a at the level of $P \|_{S,M} Q$, and the name j (which in general is different from i), generated according to the same rule for identifying the same action a inside P (or Q). In this way when such action a terminates in P (or Q) the name j can be re-mapped to the correct name 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 name i is uniquely determined 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 pair (i, loc_j) with indices $i, j \in \mathbb{N}$ and location $loc \in Loc = \{l, r\}$, where "l" stands for left and "r" for right. We denote an *association function*, whose elements are associations (i, loc_j), with *afun* which ranges over the set *Afun* of partial bijections from $\mathbb{N}$ to $Loc \times \mathbb{N}$. Finally M ranges over the set $\{M \mid M : A \mathrel{-\!\!\circ\!\!\!\rightarrow} Afun\}$ of mappings, i.e. sets including independent association functions for different action types.

Now we show how the name technique can be exploited to give operational ST semantics to our simple language. We need a richer syntax to represent the states of semantic models where prefixing is extended to semi-actions "a_1^-" and parallel operators are parametrized with mappings M (denoted by "$\|_{S,M}$"). [3]

The following notations are used in the definition of the operational rules.

Definition 2.1 Given a partial function $f : I \mathrel{-\!\!\circ\!\!\!\rightarrow} J$ we define f_i with $i \in I$

[3] The operators "$\|_S$" occurring in a term P of the process algebra are considered as being "$\|_{S,\emptyset}$" when P is regarded as a state.

as follows:

$$f_i = f(i) \text{ if } i \in dom(f); f_i = \emptyset \text{ otherwise}$$

Moreover we define $f[i \mapsto j]$ with $i \in I, j \in J$, which modifies f so that f maps i into j as follows:

$$f[i \mapsto j] = f - {<}i, f_i{>} \cup {<}i, j{>}$$

where:

$${<}k, h{>} = \{(k, h)\} \text{ if } h \neq \emptyset; {<}k, h{>} = \emptyset \text{ otherwise}$$

In the following we describe the operational rules that deviate from the standard interleaving ones.

The operational rules for observable action prefixing are:

$$a.P \xrightarrow{a_1^+} a_1^-.P \qquad\qquad a_1^-.P \xrightarrow{a_1^-} P$$

The rules for computing the starting moves of "$P \,\|_{S,M}\, Q$" for observable actions $a \notin S$ are as follows.

$$\frac{P \xrightarrow{a_i^+} P'}{P \,\|_{S,M}\, Q \xrightarrow{a^+_{n(M_a)}} P' \,\|_{S,M[a \mapsto M_a \cup \{(n(M_a), l_i)\}]}\, Q}$$

When P performs a_i^+ then a new index $n(M_a)$ is determined for identifying the action a at the level of "$\|_{S,M}$" 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: $n(\mathit{afun}) = \mathit{min}\{k \mid k \notin \mathit{dom}(\mathit{afun})\}$, where $\mathit{afun} \in \mathit{Afun}$. Symmetrically for a move a_i^+ of Q.

$$\frac{Q \xrightarrow{a_i^+} Q'}{P \,\|_{S,M}\, Q \xrightarrow{a^+_{n(M_a)}} P \,\|_{S,M[a \mapsto M_a \cup \{(n(M_a), r_i)\}]}\, Q'}$$

The rules for computing the termination moves of "$P \,\|_{S,M}\, Q$" for observable actions $a \notin S$ are as follows, where $j = M_a^{-1}(l_i)$.

$$\frac{P \xrightarrow{a_i^-} P'}{P \,\|_{S,M}\, Q \xrightarrow{a_j^-} P' \,\|_{S,M[a \mapsto M_a \text{-} \{(j, l_i)\}]}\, Q}$$

When P performs a_i^- the action of type a with index j associated to l_i in M terminates at the level of the parallel operator. Symmetrically the rule for a move a_i^- of Q is the following one, where $j = M_a^{-1}(r_i)$.

$$\frac{Q \xrightarrow{a_i^-} Q'}{P \,\|_{S,M}\, Q \xrightarrow{a_j^-} P \,\|_{S,M[a \mapsto M_a \text{-} \{(j, r_i)\}]}\, Q'}$$

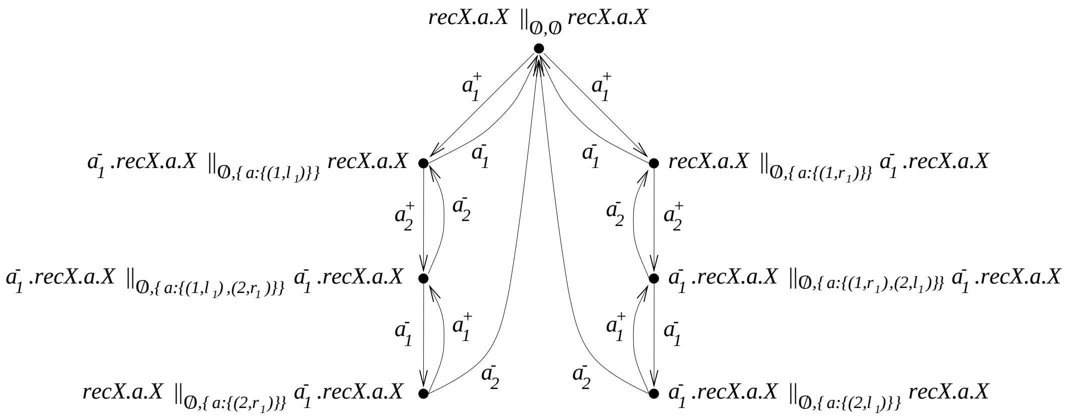
$recX.a.X \parallel_{\emptyset,\emptyset} recX.a.X$
a_1^+
a_1^+
a_1^-
a_1^-
$a_1^- .recX.a.X \parallel_{\emptyset,\{a:\{(1,l_1)\}\}} recX.a.X$
$recX.a.X \parallel_{\emptyset,\{a:\{(1,r_1)\}\}} a_1^- .recX.a.X$
a_2^+
a_2^-
a_2^-
a_2^+
$a_1^- .recX.a.X \parallel_{\emptyset,\{a:\{(1,l_1),(2,r_1)\}\}} a_1^- .recX.a.X$
$a_1^- .recX.a.X \parallel_{\emptyset,\{a:\{(1,r_1),(2,l_1)\}\}} a_1^- .recX.a.X$
a_1^-
a_1^+
a_1^+
a_1^-
a_2^-
a_2^-
$recX.a.X \parallel_{\emptyset,\{a:\{(2,r_1)\}\}} a_1^- .recX.a.X$
$a_1^- .recX.a.X \parallel_{\emptyset,\{a:\{(2,l_1)\}\}} recX.a.X$

Fig. 1. Example of Recursion with the Name Technique

The rule for computing the moves γ of "$\parallel_{S,M}$", where γ is a semi-action a_i^+ or a_i^- with $a \in S$, is:

$$\frac{P \xrightarrow{\gamma} P' \quad Q \xrightarrow{\gamma} Q'}{P \parallel_{S,M} Q \xrightarrow{\gamma} P' \parallel_{S,M} Q'}$$

This rule requires that the two synchronizing actions have the same index and produces an action with that index. Note that:

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

then the set of indices of actions a in execution in P and Q is always the same and it is never possible for P and Q to start actions with different indices.

Example 2.2 In Fig. 1 we depict the semantic model of $recX.a.X \parallel_{\emptyset} recX.a.X$, where $recX$ denotes recursion in the usual way. [4] ■

As we show in [6], with the name technique we have that the ST semantics of a process P is finite state if and only if the interleaving semantics of P is finite state. A simple syntactical characterization for processes that are guaranteed to be finite state is the following one. P is finite state if for each subterm $recX.Q$ of P, X does not occur free in Q in the context of a *static operator* [19] (for our basic language, the parallel operator only). Note that this class of processes includes strictly the class of nets of automata, i.e. terms where no static operator occurs in the scope of any recursion.

The equivalence notion we consider over terms, denoted with $\simeq_n$ (where the n stands for the name technique), is the standard notion of *observational congruence* extended to open terms [19].

[4] We use $a : I$ as a shorthand notation for (a, I).

As we show in detail in [6], a complete axiomatization for $\simeq_n$ over finite state terms of our process algebra is simply obtained by modifying the axioms related to the parallel operator, reflecting the new operational rules, in the standard axiom set of [20].

3 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 large semantic models. The reason for this is the intricate structure of mappings M in terms $P \,\|_{S,M}\, Q$. The problem is that, for any type a, the association function M_a, may present *holes* and *index permutations*. The exact nature of these two phenomena is explained by the following two examples, the first showing how holes can be generated and the second describing a computation that leads to an index permutation.

Example 3.1 A hole in the ordered sequence of indices can be generated as follows. Consider $a.\underline{0} \,\|_\emptyset\, a.\underline{0}$. After the right-hand a starts, the state is $a.\underline{0} \,\|_{\emptyset,\{a:\{(1,r_1)\}\}}\, a_1^- .\underline{0}$. After the left-hand a starts, the state is $a_1^- .\underline{0} \,\|_{\emptyset,\{a:\{(1,r_1),(2,l_1)\}\}}\, a_1^- .\underline{0}$. Finally, after the right-hand 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.

For the next example, we denote the association function M_a for a given type a as a string on the alphabet $Loc \cup \{*\}$, where $*$ denotes a hole in the ordered sequence of indices of actions a started in $P \,\|_{S,M}\, Q$. For example $\{(1,l_1),(3,r_1)\}$ is represented by the string "$l_1 {*} r_1$".

Example 3.2 An index permutation occurs when an association function does not preserve the order of indices. 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 indices and which could guarantee that the order of indices is preserved by mappings (so that indices 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 of all occurrences of a in P by Q. As suggested in [8], $P[a \rightsquigarrow Q]$ 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 $P \parallel_{\emptyset} Q$ is turned into P if Q is terminated. With the name technique such a rule cannot be implemented in general. For instance $a \parallel_{\emptyset} a$ reaches the state $a_1^- .\underline{0} \parallel_{\emptyset,\{a:\{(2,l_1)\}\}} \underline{0}$ which is not equivalent to a_1^- as well as $(a.\underline{0} \parallel_{\emptyset} a.\underline{0}) \parallel_{\emptyset} a.\underline{0}$ reaches the state $(a_1^- .\underline{0} \parallel_{\emptyset,\{a:l_1 r_1\}} a_1^- .\underline{0}) \parallel_{\emptyset,\{a:l_2 l_1\}} \underline{0}$ which is not equivalent to $a_1^- .\underline{0} \parallel_{\emptyset,\{a:l_1 r_1\}} a_1^- .\underline{0}$. The point is that, due to the possible presence of holes or index permutations, it may be that, when a parallel operator should be eliminated, the related mapping is not a simple identity.

4 ST Semantics via the Stack Technique

In order to solve all the problems we reported in the previous paragraph and to give a satisfactory operational semantics to a language including a semantic action refinement operator, we introduce another technique for representing the ST semantics. This technique is based on the idea of eliminating the holes in the sequences of started action indices. 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 [16,17,8] 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 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 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 indices in the case of an action start or termination is fixed, actions of processes that perform equivalent computations get the same indices when they terminate and ST bisimilarity can simply be checked by applying standard bisimilarity.

Let w range over the set $AStr = \{w : \mathbb{N}^+ \mathrel{-\!\!\circ\!\!\!\rightarrow} Loc \mid \exists k \geq 1 : dom(w) = \{1 \ldots k\}\}$ of association strings, i.e. non-empty strings over the alphabet of locations $Loc = \{l, r\}$ [5]. M is a partial function mapping action types in association strings $w \in AStr$. 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 left-most 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 left-hand term (if the location is "l") or right-hand term (if the location is "r") of the parallel composition operator. The index j of the action a of the left-hand (or right-hand) 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 4.1 In Fig. 2 we depict the ST semantic model of $recX.a.X \parallel_\emptyset recX.a.X$ obtained by applying the stack technique. [6] By comparing the semantic model of Fig. 2 with that of Fig. 1 we can see that the phenomenum of holes of the name technique generates only two additional states. If we consider $(recX.a.X \parallel_\emptyset recX.a.X) \parallel_\emptyset recX.a.X$ we have that, due to the combined effects of holes and index permutations, the ST semantic model obtained with the name technique has 42 states, whilst that obtained with the stack technique has only 16 states.

We consider a language 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 sequential composition operator ";" instead of the CCS prefix operator ".".

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

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

[6] We apply the stack technique to a simple language with prefixing instead of general sequential composition in this preliminary example only.

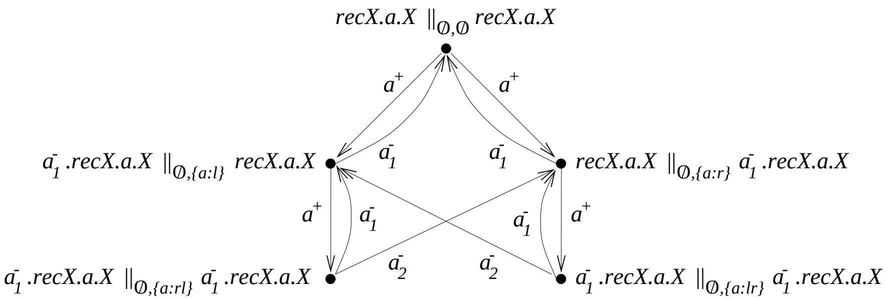
recX.a.X ||∅,∅ recX.a.X
a+
a+
ā1 .recX.a.X ||∅,{a:l} recX.a.X
ā1
ā1
recX.a.X ||∅,{a:r} ā1 .recX.a.X
a+
ā1
ā1
a+
ā1 .recX.a.X ||∅,{a:rl} ā1 .recX.a.X
ā2
ā2
ā1 .recX.a.X ||∅,{a:lr} ā1 .recX.a.X

Fig. 2. Example of Recursion with the Stack Technique

$$\epsilon \xrightarrow{\surd} \delta$$

$$\frac{P \xrightarrow{\theta} P'}{P;Q \xrightarrow{\theta} P';Q} \qquad \frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\chi} Q'}{P;Q \xrightarrow{\chi} Q'}$$

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

$$\frac{P \xrightarrow{\chi} P'}{P/\mathcal{L} \xrightarrow{\chi} P'/\mathcal{L}} \; \mathit{type}(\chi) \notin \mathcal{L} \qquad \frac{P \xrightarrow{\gamma} P'}{P/\mathcal{L} \xrightarrow{\hat{\tau}} P'/\mathcal{L}} \; \mathit{type}(\gamma) \in \mathcal{L}$$

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

Table 1 Standard Rules

computation. Let μ range over Act. Moreover let Var be a set of process variables ranged over by X, Y, Z. The terms of RL (refinement language) are generated by the following syntax:

$$P ::= \epsilon \mid \delta \mid X \mid \mu \mid P;P \mid P+P \mid P \parallel_S P \mid P/L \mid \mathit{recX}.P \mid P[a \leadsto P]$$

"$\parallel_S$" is the CSP 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. A RL process is a closed term of RL. We denote RL_G the set of strongly guarded processes of RL.

To define the operational semantics of RL processes, we need a richer syntax to represent states. We denote with $SA = A \cup \{e\}$ 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 SA and $\mathcal{S}, \mathcal{L}$ range over the subsets of SA. The set of all state action types is denoted by $SAct = SA \cup \{\tau\}$, where τ is a distinguished type representing an internal computation. Let $SA_{ST} = \{\alpha^+ \mid \alpha \in SA\} \cup \{\alpha_i^- \mid \alpha \in SA \wedge i \in \mathbb{N}^+\}$, ranged over by γ; and $SAct_{ST} = SA_{ST} \cup \{\hat{\tau}\}$, where $\hat{\tau}$ is a distinguished semi-action representing an internal computation. [7] Let θ range over $SAct_{ST}$ and η range over $SAct \cup SAct_{ST}$. The meta-variable χ ranges over $SAct_{ST} \cup \{\surd\}$, where $\surd$ is a distinguished action representing successful termination. Moreover M ranges over the set $\{M \mid M : SA \mathrel{-\!\!\circ\!\!\!\rightarrow} AStr\}$ of mappings, i.e. sets including the association strings for all the state actions currently in execution. Finally, let φ range over the bijections over SA. The state terms are generated by the following syntax:

$$P ::= \epsilon \mid \delta \mid X \mid \eta \mid P;P \mid P+P \mid P \,\|_{\mathcal{S},M}\, P \mid P/\mathcal{L} \mid recX.P \mid P[a \rightsquigarrow P] \mid !P \mid P[\varphi]$$

The bang operator "!" and the (bijective) relabeling operator "$[\varphi]$" are auxiliary operators that are necessary for the definition of the refinement operator. [8] 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 "$\|_{\mathcal{S}}$" as being "$\|_{\mathcal{S},\emptyset}$" when an RL process P is regarded as a state.

The semantics of state terms produces a transition system labeled over $SAct_{ST}$. The operational rules for "ϵ" and the operators ";", "+", "$/\mathcal{L}$" and "$recX$" are the standard ones and are presented in Table 1. The operational rules for "η" and the operator "$\|_{\mathcal{S},M}$" are presented in Table 2.

The function $type : SAct_{ST} \cup \{\surd\} \longrightarrow SAct \cup \{\surd\}$ is defined in the obvious way. The termination predicate $\surd$ is defined as $P\surd \Leftrightarrow (\exists Q : P \xrightarrow{\;\surd\;} Q) \wedge (\nexists \theta \in SAct_{ST}, Q' : P \xrightarrow{\;\theta\;} Q')$. 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. $w \ominus i = \{(j, loc) \in w \mid j < i\} \cup \{(j-1, loc) \mid (j, loc) \in w \wedge j > i\}$.

The meaning of the operational rules for "$P \,\|_{\mathcal{S},M}\, Q$" is the following.

When P 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 Q.

When P 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 2 because we eliminate the parallel operator in the case P becomes a successfully terminated process. Symmetrically for a move α_i^- of Q.

[7] Introducing an invisible semi-action $\hat{\tau}$ is not strictly necessary. 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 $a/\{a\}$.

[8] The restriction to bijective relabelings allows us to give a simple operational semantics to the operator "$P[\varphi]$". 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.

$$\alpha \xrightarrow{\alpha^+} \alpha_1^- \qquad \tau \xrightarrow{\hat{\tau}} \hat{\tau} \qquad \theta \xrightarrow{\theta} \epsilon$$

$$\frac{P \xrightarrow{\alpha^+} P'}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\alpha^+} P' \,\|_{\mathcal{S},M[\alpha \mapsto l M_\alpha]}\, Q} \quad \alpha \notin \mathcal{S}$$

$$\frac{Q \xrightarrow{\alpha^+} Q'}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\alpha^+} P \,\|_{\mathcal{S},M[\alpha \mapsto r M_\alpha]}\, Q} \quad \alpha \notin \mathcal{S}$$

$$\frac{P \xrightarrow{\alpha_i^-} P' \quad \neg(P'\surd \,\wedge\, \mathcal{S} = \emptyset)}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\alpha^-_{\#_{i,l}(M_\alpha)}} P' \,\|_{\mathcal{S},M[\alpha \mapsto M_\alpha \ominus \#_{i,l}(M_\alpha)]}\, Q} \quad \alpha \notin \mathcal{S} \qquad \frac{P \xrightarrow{\alpha_1^-} P' \quad P'\surd}{P \,\|_{\emptyset,M}\, Q \xrightarrow{\alpha^-_{\#_{1,l}(M_\alpha)}} Q}$$

$$\frac{Q \xrightarrow{\alpha_i^-} Q' \quad \neg(Q'\surd \,\wedge\, \mathcal{S} = \emptyset)}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\alpha^-_{\#_{i,r}(M_\alpha)}} P \,\|_{\mathcal{S},M[\alpha \mapsto M_\alpha \ominus \#_{i,r}(M_\alpha)]}\, Q'} \quad \alpha \notin \mathcal{S} \qquad \frac{Q \xrightarrow{\alpha_1^-} Q' \quad Q'\surd}{P \,\|_{\emptyset,M}\, Q \xrightarrow{\alpha^-_{\#_{1,r}(M_\alpha)}} P}$$

$$\frac{P \xrightarrow{\hat{\tau}} P' \quad \neg(P'\surd \,\wedge\, \mathcal{S} = \emptyset)}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\hat{\tau}} P' \,\|_{\mathcal{S},M}\, Q} \qquad \frac{P \xrightarrow{\hat{\tau}} P' \quad P'\surd}{P \,\|_{\emptyset,M}\, Q \xrightarrow{\hat{\tau}} Q}$$

$$\frac{Q \xrightarrow{\hat{\tau}} Q' \quad \neg(Q'\surd \,\wedge\, \mathcal{S} = \emptyset)}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\hat{\tau}} P \,\|_{\mathcal{S},M}\, Q'} \qquad \frac{Q \xrightarrow{\hat{\tau}} Q' \quad Q'\surd}{P \,\|_{\emptyset,M}\, Q \xrightarrow{\hat{\tau}} P}$$

$$\frac{P \xrightarrow{\chi} P' \quad Q \xrightarrow{\chi} Q'}{P \,\|_{\mathcal{S},M}\, Q \xrightarrow{\chi} P' \,\|_{\mathcal{S},M}\, Q'} \quad \mathit{type}(\chi) \in \mathcal{S} \cup \{\surd\}$$

Table 2 Rules for the Stack Technique

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 $P[a \rightsquigarrow Q]$ that closely adheres to the intuition of the way it works:

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

where the bijective relabeling $\alpha \leftrightarrow \alpha'$ is defined by $\alpha \leftrightarrow \alpha' = \{(\alpha, \alpha'), (\alpha', \alpha)\} \cup \{(\alpha'', \alpha'') \mid \alpha'' \in \mathit{SA} \wedge \alpha'' \notin \{\alpha, \alpha'\}\}$. For each a executed by the process P a corresponding process Q is activated by the bang operator in the right-hand term. In this way if P executes several auto-concurrent actions a then a corresponding number of processes Q are executed in parallel by the right-hand term. The correct association between actions a and processes Q 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. [9]

[9] This definition of semantic action refinement is slightly different from the usual definition [12,11,14] in that in $P[a \rightsquigarrow Q]$ each execution of Q is preceded and followed by the

$$\frac{(\ P[a \leftrightarrow e] \ \|_{\{e\},\emptyset} \ !(e^+; Q; e_1^-)\ )/\{e\} \xrightarrow{\ \chi\ } P'}{P[a \rightsquigarrow Q] \xrightarrow{\ \chi\ } P'}$$

$$\frac{P \xrightarrow{\ \alpha^+\ } P'}{P[\varphi] \xrightarrow{\ \varphi(\alpha)^+\ } P'[\varphi]} \qquad \frac{P \xrightarrow{\ \alpha_i^-\ } P'}{P[\varphi] \xrightarrow{\ \varphi(\alpha)_i^-\ } P'[\varphi]}$$

$$\frac{P \xrightarrow{\ \chi\ } P'}{P[\varphi] \xrightarrow{\ \chi\ } P'[\varphi]} \quad \mathit{type}(\chi) \in \{\tau, \surd\}$$

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

$$\frac{P \xrightarrow{\ \alpha^+\ } P'}{!P \xrightarrow{\ \alpha^+\ } P' \|_{\emptyset,\{\alpha:l\}} !P} \qquad \frac{P \xrightarrow{\ \hat{\tau}\ } P'}{!P \xrightarrow{\ \hat{\tau}\ } P' \|_{\emptyset,\emptyset} !P}$$

Table 3 Rules for Refinement

The operational rules for the refinement operator, "!" and "$[\varphi]$" are presented in Table 3.

Theorem 4.2 *The interleaving semantics of a RL process P not including the refinement operator is finite state* [10] *iff the ST semantics of P obtained with the stack technique is finite state.*

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

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

Theorem 4.3 *If P and Q are finite state RL processes, then $P[a \rightsquigarrow Q]$ is a finite state process.*

Example 4.4 In Fig. 3 we present the finite ST semantic model of $recX.a; X[a \rightsquigarrow b; c]$ obtained with the stack technique. In Fig. 4 we show an initial fragment of the infinite ST semantic model of the same term obtained with the name technique. Note that ST semantics via the 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_i^+ 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. 4 makes clear that, in the absence of an elimination rule for the parallel operator, the number of parallel

occurrence of a silent transition $\hat{\tau}$. 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 [8].

[10] The interleaving operational rules are the standard ones, therefore they are omitted.

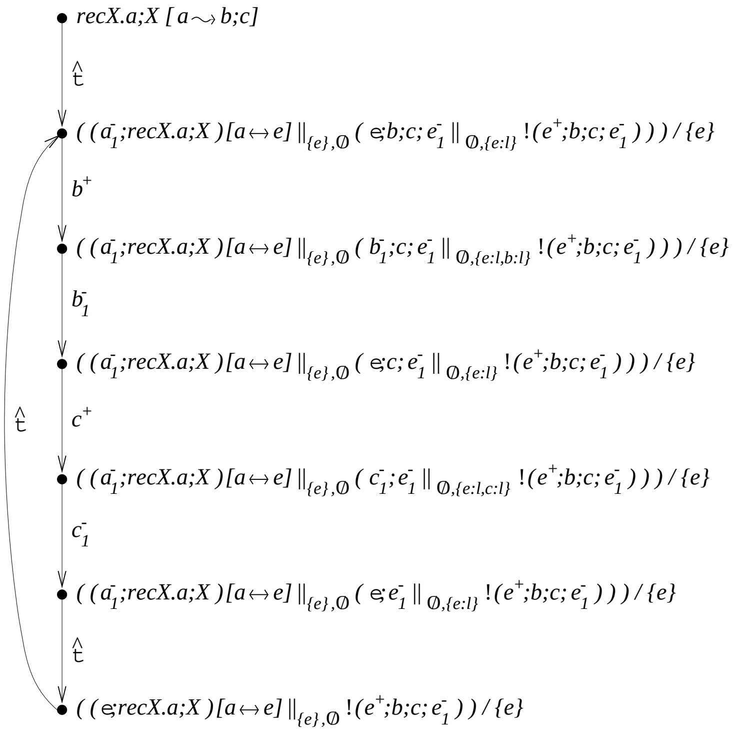
recX.a;X [a ⇝ b;c]
τ̂
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε;b;c; e⁻₁ || ∅,{e:1} !(e⁺;b;c; e⁻₁))) / {e}
b⁺
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (b⁻₁;c; e⁻₁ || ∅,{e:1,b:1} !(e⁺;b;c; e⁻₁))) / {e}
b⁻₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε;c; e⁻₁ || ∅,{e:1} !(e⁺;b;c; e⁻₁))) / {e}
τ̂
c⁺
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (c⁻₁; e⁻₁ || ∅,{e:1,c:1} !(e⁺;b;c; e⁻₁))) / {e}
c⁻₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε; e⁻₁ || ∅,{e:1} !(e⁺;b;c; e⁻₁))) / {e}
τ̂
((ε;recX.a;X)[a↔e] ||{e},∅ !(e⁺;b;c; e⁻₁)) / {e}

Fig. 3. Example of Refinement with the Stack Technique

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. [11]

Now we give a simple syntactical characterization for RL processes that are guaranteed to be finite state. In the following corollary we consider as static the operators of parallel composition, hiding and action refinement.

Corollary 4.5 *Let P be a RL process 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 left-hand side of a "$;$". Then P is a finite state process.*

The equivalence notion we consider over RL processes, denoted with $\simeq_s$ (where the s stands for the stack technique) is again observational congruence where the alphabet of visible actions is SA_{ST} and hidden actions are $\hat{\tau}$ actions.

Once extended the application of the name technique to the whole language RL, as explained in the previous example, we have the following theorem of consistency.

Theorem 4.6 *Given two RL processes P and Q we have that $P \simeq_n Q$ iff $P \simeq_s Q$.*

[11] For the simple example $recX.a;X[a \rightsquigarrow b;c]$, the execution of a refinement by means of $!(e_i^+;b;c;e_i^-)$ always leads to $\epsilon \,\|_{\emptyset,\emptyset} !(e_i^+;b;c;e_i^-)$, 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. 1) includes states which exhibit "holes" in the index sequences of started actions.

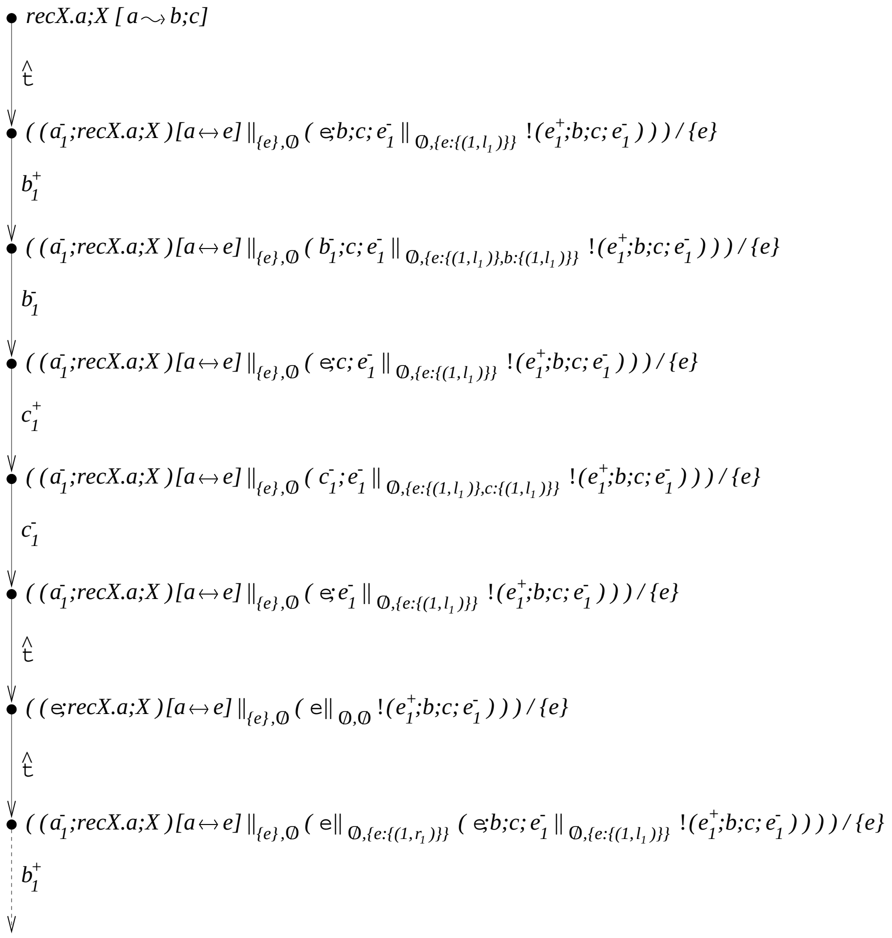
recX.a;X [a ⇝ b;c]
t̂
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε;b;c; e⁻₁ || ∅,{e:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁))) / {e}
b⁺₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (b⁻₁;c; e⁻₁ || ∅,{e:{(1,l₁)},b:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁))) / {e}
b⁻₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε;c; e⁻₁ || ∅,{e:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁))) / {e}
c⁺₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (c⁻₁; e⁻₁ || ∅,{e:{(1,l₁)},c:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁))) / {e}
c⁻₁
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε; e⁻₁ || ∅,{e:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁))) / {e}
t̂
((ε;recX.a;X)[a↔e] ||{e},∅ (ε|| ∅,∅ !(e⁺₁;b;c; e⁻₁))) / {e}
t̂
((a⁻₁;recX.a;X)[a↔e] ||{e},∅ (ε|| ∅,{e:{(1,r₁)}} (ε;b;c; e⁻₁ || ∅,{e:{(1,l₁)}} !(e⁺₁;b;c; e⁻₁)))) / {e}
b⁺₁

Fig. 4. Example of Refinement with the Name Technique

Theorem 4.7 *$P \simeq_s Q$ is a congruence w.r.t. all the operators of RL, including recursion.*

5 Axiomatization via the Stack Technique

The axiom system $\mathcal{A}_{RL}$ for $\simeq_s$ on RL_G terms is formed by: the standard axioms presented in Table 4 (where "$\lfloor\!\lfloor$" and "$|$" denote, respectively, the left merge and synchronization merge operators and the axiom $(LM2)$ reflects the elimination rules for the parallel operator), the axioms of Table 5 which are specific for the stack technique, and the axioms of Table 6 which deal with the refinement operator.

The axiom (Par) is the standard one except that when the position of processes P and Q is exchanged we must invert left and right inside M. The inverse $\overline{M}$ of a mapping M is defined by $\overline{M} = \{(a, \overline{w}) \mid (a, w) \in M\}$ where $\overline{w} = \{(i, r) \mid (i, l) \in w\} \cup \{(i, l) \mid (i, r) \in w\}$. Axioms $(LM7)$ and $(LM8)$ just reflect the operational rules of the parallel operator for an independent move of the left-hand process.

If we consider the obvious operational rules for "$\lfloor\!\lfloor_{\mathcal{S},M}$" and "$|_{\mathcal{S},M}$" that

$(A1)$	$P + Q = Q + P$	$(A2)$	$(P + Q) + R = P + (Q + R)$
$(A3)$	$P + P = P$	$(A4)$	$P + \delta = P$
$(Tau1)$	$\theta; \hat{\tau} = \theta$	$(Tau2)$	$P + \hat{\tau}; P = \hat{\tau}; P$
$(Tau3)$	$\gamma; (P + \hat{\tau}; Q) + \gamma; Q = \gamma; (P + \hat{\tau}; Q)$		
$(Seq1)$	$(P; Q); R = P; (Q; R)$	$(Seq2)$	$(P + Q); R = P; R + Q; R$
$(Seq3)$	$\epsilon; P = P$	$(Seq4)$	$P; \epsilon = P$
$(Seq5)$	$\delta; P = \delta$		
$(LM1)$	$(P + Q) \lfloor\!\lfloor_{\mathcal{S},M} R = P \lfloor\!\lfloor_{\mathcal{S},M} R + Q \lfloor\!\lfloor_{\mathcal{S},M} R$		
$(LM2)$	$(\theta; P) \lfloor\!\lfloor_{\emptyset,M} \epsilon = \theta; P$		
$(LM3)$	$(\hat{\tau}; P) \lfloor\!\lfloor_{\mathcal{S},M} Q = \hat{\tau}; (P \parallel_{\mathcal{S},M} Q)$		
$(LM4)$	$(\gamma; P) \lfloor\!\lfloor_{\mathcal{S},M} Q = \delta$	$\mathit{type}(\gamma) \in \mathcal{S}$	
$(LM5)$	$\epsilon \lfloor\!\lfloor_{\mathcal{S},M} P = \delta$		
$(LM6)$	$\delta \lfloor\!\lfloor_{\mathcal{S},M} P = \delta$		
$(SM1)$	$P \,|_{\mathcal{S},M}\, Q = Q \,|_{\mathcal{S},M}\, P$		
$(SM2)$	$(P + Q) \,|_{\mathcal{S},M}\, R = P \,|_{\mathcal{S},M}\, R + Q \,|_{\mathcal{S},M}\, R$		
$(SM3)$	$(\gamma; P) \,|_{\mathcal{S},M} (\gamma; Q) = \gamma; (P \parallel_{\mathcal{S},M} Q)$	$\mathit{type}(\gamma) \in \mathcal{S}$	
$(SM4)$	$(\hat{\tau}; P) \,|_{\mathcal{S},M}\, Q = P \,|_{\mathcal{S},M}\, Q$		
$(SM5)$	$(\gamma; P) \,|_{\mathcal{S},M} (\gamma'; Q) = \delta$	$\mathit{type}(\gamma) \notin \mathcal{S} \ \vee \ \gamma \neq \gamma'$	
$(SM6)$	$\epsilon \,|_{\mathcal{S},M}\, \epsilon = \epsilon$		
$(SM7)$	$\epsilon \,|_{\mathcal{S},M} (\theta; P) = \delta$		
$(SM8)$	$\delta \,|_{\mathcal{S},M}\, P = \delta$		
$(Hi1)$	$(P + Q)/\mathcal{L} = P/\mathcal{L} + Q/\mathcal{L}$		
$(Hi2)$	$(P; Q)/\mathcal{L} = P/\mathcal{L}; Q/\mathcal{L}$		
$(Hi3)$	$\theta/\mathcal{L} = \theta$	$\mathit{type}(\theta) \notin \mathcal{L}$	
$(Hi4)$	$\gamma/\mathcal{L} = \hat{\tau}$	$\mathit{type}(\gamma) \in \mathcal{L}$	
$(Hi5)$	$\epsilon/\mathcal{L} = \epsilon$		
$(Hi6)$	$\delta/\mathcal{L} = \delta$		
$(Rec1)$	$recX.P = P\{recX.P/X\}$		
$(Rec2)$	$Q = P\{Q/X\} \Rightarrow Q = recX.P$	provided that X is strongly guarded in P	

Table 4 Standard Axioms

$(Act1)$	$\alpha = \alpha^+; \alpha_1^-$	$(Act2)$	$\tau = \hat{\tau}; \hat{\tau}$
(Par)	$P \parallel_{\mathcal{S},M} Q = P \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} Q + Q \mathbin{\lfloor\!\lfloor}_{\mathcal{S},\overline{M}} P + P \mid_{\mathcal{S},M} Q$		
$(LM7)$	$(\alpha^+; P) \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} Q = \alpha^+; (P \parallel_{\mathcal{S},M[a \mapsto l M_\alpha]} Q)$		$\alpha \notin \mathcal{S}$
$(LM8)$	$(\alpha_i^-; P) \mathbin{\lfloor\!\lfloor}_{\mathcal{S},M} Q = \alpha_{l(w,i)}^-; (P \parallel_{\mathcal{S},M[a \mapsto M_\alpha \ominus \#_{i,l}(M_\alpha)]} Q)$		$\alpha \notin \mathcal{S}$

Table 5 Axioms for the Stack Technique

(Ref)	$P[a \rightsquigarrow Q] = (\ P[a \leftrightarrow e] \parallel_{\{e\},\emptyset} !(e^+; Q; e_1^-)\ )/\{e\}$		
$(Rel1)$	$(P + Q)[\varphi] = P[\varphi] + Q[\varphi]$	$(Rel2)$	$(P; Q)[\varphi] = P[\varphi]; Q[\varphi]$
$(Rel3)$	$\alpha^+[\varphi] = \varphi(\alpha)^+$	$(Rel4)$	$\alpha_i^-[\varphi] = \varphi(\alpha)_i^-$
$(Rel5)$	$\hat{\tau}[\varphi] = \hat{\tau}$	$(Rel6)$	$\epsilon[\varphi] = \epsilon$
$(Rel7)$	$\delta[\varphi] = \delta$		
$(Bang)$	$!P = recX.(\epsilon + P \mathbin{\lfloor\!\lfloor}_{\emptyset,\emptyset} X)$ provided that X is not free in P		

Table 6 Axioms for Refinement

derive from those we presented for the parallel operator, [12] then the axioms of $\mathcal{A}_{RL}$ are sound.

We have the following theorem, where a sequential state is a state that includes only "ϵ", "δ", "X" and operators "$\eta; P$", "$P + P$", "$recX.P$".

Theorem 5.1 *If a RL_G process P is finite state then $\exists P' : \mathcal{A}_{RL} \vdash P = P'$ with P' sequential state.*

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 "$\eta; P$" as being "$\eta.P$", "δ" as being "$\underline{0}$" and "ϵ" as being "$\surd.\underline{0}$"), from [20] and Theorem 5.1 we derive the completeness of $\mathcal{A}_{RL}$.

Theorem 5.2 *$\mathcal{A}_{RL}$ is complete for $\simeq_s$ over finite state RL_G processes.*

6 Conclusion

We think that the two techniques for expressing ST semantics, which are based on the new idea of compositional level-wise re-indexing, that we have introduced can be exploited also for deciding and axiomatizing other forms of history dependent bisimulations over processes that possess a finite interleaving semantics as well as bisimilarity for name-passing calculi (e.g. π-calculus). For example [21] uses a technique that is very similar to our name technique, even if not in a compositional way, to express history preserving bisimulation. As far as location bisimulation is concerned, the two techniques collapse in a

[12] The definition of the operational rule for "$|_{\mathcal{S},M}$" must allow for actions "$\hat{\tau}$" to be skipped [1], as reflected by axiom $(SM4)$.

single one because locations never become obsolete and the problems related to the reuse of names do not arise. Even if the stack technique is more adequate in the context of ST semantics because it allows to decide ST bisimulation also in the case of action refinement, we believe that both techniques have different features that may make one of them more suitable than the other one depending on the context of application. For example in the language of [7], where (probabilistically) timed actions are given a semantics similar to ST semantics, the name technique (as opposed to the stack technique) gives rise to semantic models which are very close to Generalized Semi-Markov Processes (GSMPs), where names assigned to actions correspond to the elements of a GSMP.

Acknowledgement

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

References

[1] L. Aceto, *"On "Axiomatising Finite Concurrent Processes" "* in SIAM Journal on Computing 23(4):852-863, 1994

[2] L. Aceto, *"A Static View of Localities"*, in Formal Aspects of Computing, 6:201-222, 1994

[3] L. Aceto, M. Hennessy, *"Adding Action Refinement to a Finite Process Algebra"*, in Information and Computation 115:179-247, 1994

[4] M. Bravetti, M. Bernardo, R. Gorrieri, *"Towards Performance Evaluation with General Distributions in Process Algebras"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:405-422, Nice (France), 1998

[5] G. Boudol, I. Castellani, M. Hennesy, A. Kiehn, *"A theory of processes with localities"*, in Formal Aspects of Computing 6(2):165-200, 1994

[6] M. Bravetti, R. Gorrieri, *"Deciding and Axiomatizing ST Bisimulation for a Process Algebra with Recursion and Action Refinement"*, Technical Report UBLCS-99-1, University of Bologna (Italy), 1999

[7] M. Bravetti, R. Gorrieri, *"Interactive Generalized Semi-Markov Processes"*, to appear in Proc. of the *7th Int. Workshop on Process Algebras and Performance Modeling (PAPM '99)*, Zaragoza (Spain), September 1999

[8] N. Busi, R.J. van Glabbeek, R. Gorrieri, *"Axiomatising ST-Bisimulation Equivalence"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '94)*, pp. 169-188, S. Miniato (Italy), 1994

[9] I. Castellani, *"Observing Distribution In Processes: Static and Dynamic Localities"*, in Int. Journal of Foundations of Computer Science 6:353-393, 1995

[10] P. Darondeau, R. Degano, *"Causal Trees"*, in Automata, Languages and Programming, LNCS 372:234-248, 1989

[11] P. Degano, R. Gorrieri, *"A causal operational semantics of action refinement"*, in Information and Computation 122:97-119, 1995

[12] R.J. van Glabbeek, *"The refinement theorem for ST-bisimulation semantics"*, in Proc. of the *IFIP Working Conf. on Programming Concepts, Methods and Calculi (PROCOMET '90)*, pp. 27-52, Sea of Gallilea (Israel), 1990

[13] R.J. van Glabbeek, U. Goltz, *"Equivalence Notions for Concurrent Systems and Refinement of Actions"*, Arbeitspapiere der GMD 366, Gesellschaft fur Mathematik und Datenverarbeitung MBH, 1989

[14] R. Gorrieri, A. Rensink, *"Action Refinement"*, to appear in Handbook of Process Algebra, Elsevier, 2000

[15] R.J. van Glabbeek, F.W. Vaandrager, *"Petri Net Models for Algebraic Theories of Concurrency"*, in Proc. of the *Conf. on Parallel Architectures and Languages Europe (PARLE '87)*, LNCS 259:224-242, Eindhoven (The Netherlands), 1987

[16] R. Gorrieri, C. Laneve, *"The limit of split$_n$ bisimulations for CCS agents"*, in Proc. of the *Symp. on Mathematical Foundations of Computer Science (MFCS '91)*, LNCS 520:170-180, 1991

[17] R. Gorrieri, C. Laneve, *"Split and ST Bisimulation Semantics"*, in Information and Computation 118:272-288, 1995

[18] A. Kiehn, M. Hennessy, *"On the decidability of non-interleaving process equivalences"*, in Fundamenta Informaticae 30(1):18-33, 1997.

[19] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[20] R. Milner, *"A complete axiomatization for observational congruence of finite-state behaviours"*, in Information and Computation 81:227-247, 1989

[21] U. Montanari, M. Pistore, *"Minimal Transition Systems for History-Preserving Bisimulation"*, in Proc. of the *14th Symp. on Theoretical Aspects of Computer Science (STACS'97)*, LNCS 1200, 1997

[22] U. Montanari, D. Yankelevich, *"Location Equivalence in Parametric Setting"*, in Theoretical Computer Science 149(2):299-332, 1995

[23] G. Plotkin, *"A Structural Approach to Operational Semantics"*, Technical Report DAIMI FN-19, Aarhus University, Department of Computer Science, Aarhus, 1981.

[24] A. Rabinovich, B. Trakhtenbrot, *"Behaviour Structures and Nets"*, in Fundamenta Informaticae 11:357-404, 1988.

[25] W. Vogler, *"Bisimulation and action refinement"*, in Theoretical Computer Science 114:173-200, 1993