Symbolic Semantic Rules for Producing Compact STGLAs from Value Passing Process Descriptions

MARCO BERNARDO
University of Urbino - Italy

Value passing process algebras with infinite data domains need to be equipped with symbolic semantic models in order for their analysis to be possible. This means that appropriate symbolic models and the related verification algorithms must be developed, together with suitable semantic rules mapping the value passing process descriptions to such symbolic models. In this paper, we first introduce the model of the symbolic transition graphs with lookahead assignment (STGLA), a variant of the symbolic transition graphs with assignment (STGA) of Lin that can undergo to the strong, weak and observational bisimulation equivalence checking algorithms of Li and Chen. We then define a set of symbolic semantic rules that map a useful fragment of value passing CCS to finite STGLA without making any assumption on the variable names. We demonstrate that the symbolic semantic rules are correct with respect to both the usual concrete semantic rules and the novel issue of the assignment application order. Finally, we prove that, for the considered fragment of value passing CCS, the STGLA produced by the symbolic semantic rules are optimal with respect to a certain compactness criterion, thus improving on the symbolic models and the semantic rules previously proposed in the literature.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*automata*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*operational semantics*; *process models*

General Terms: Languages, Theory, Verification

Additional Key Words and Phrases: Process algebras, value passing, symbolic semantics

1. INTRODUCTION

Process description languages, such as e.g. CCS [Milner 1989] and the extensions to mobility, real time, performance evaluation, and security, are a useful tool for the design of concurrent and distributed systems because of their compositional nature and their well established semantic theory. The study of the properties of a process description is usually conducted on a transition graph, formally derived from the description itself via structured operational semantic rules, by applying well known analysis techniques [Cleaveland et al. 1993]. Therefore, algorithms for

Author address: Marco Bernardo, Università di Urbino, Istituto di Scienze e Tecnologie dell'Informazione, Piazza della Repubblica 13, 61029 Urbino, Italy (bernardo@sti.uniurb.it).

the generation and analysis of transition graphs underlying process terms have received much attention, especially because of their impact on the implementation of process algebra based software tools.

A special case of process algebra is the one in which the interprocess communication mechanism not only provides a synchronization facility but also supports value passing across the processes. For a value passing process algebra with infinite data domains, the problem is that the traditional operational semantic rules yield infinite transition graphs, thus preventing value passing process descriptions from being analyzed.

Objectives. The purpose of this paper is to identify a useful fragment of value passing CCS with possibly infinite data domains for which suitable semantic rules can be defined in such a way that:

(1) the process terms in the fragment are transparently mapped to finite transition graphs,

(2) the generated transition graphs are equipped with verification algorithms, and

(3) the generated transition graphs are intuitive and compact.

Variants of Symbolic Transition Graphs. In the literature it has been recognized that symbolic semantic models and symbolic semantic rules must be developed to deal with infinite data domains. To start with, in [Hennessy and Lin 1995] symbolic transition graphs were advocated to enable a finite representation of a large class of process descriptions, so that conventional analysis techniques apply. The model of symbolic transition graphs is recalled below.

Let us preliminarily introduce some syntactic categories that we shall use in the sequel:

—Let Val be a set of values ranged over by v.

—Let Var be a set of variables ranged over by x, y, z.

—Let Exp be a set of expressions over $\mathit{Val} \cup \mathit{Var}$ ranged over by e, including the usual arithmetical, logical, and relational operators.

—Let BExp be a set of boolean expressions in Exp ranged over by β.

—Let $\mathit{Sub} \subseteq (\mathit{Var} \times \mathit{Exp})^*$ be a set of well typed, unique, ordered substitutions ranged over by σ, with $\triangleright$ denoting the non-commutative concatenation operator on Sub. A substitution is denoted by $\underline{x} := \underline{e}$ to specify the well typed, ordered assignment of the vector of n expressions $\underline{e}$ to the vector of n distinct variables $\underline{x}$.

—Let $\mathit{Act} = \mathit{Act}_{\mathrm{I}} \cup \mathit{Act}_{\mathrm{O}} \cup \{\tau\}$ be a set of actions ranged over by α. Denoted by $\mathit{Name}_{\mathrm{IO}}$ a set of action names not containing τ and ranged over by a, an action α can be an input action in $\mathit{Act}_{\mathrm{I}} = \{a?(\underline{x}) \mid a \in \mathit{Name}_{\mathrm{IO}} \wedge \underline{x} \subseteq \mathit{Var}\}$, an output action in $\mathit{Act}_{\mathrm{O}} = \{a!(\underline{e}) \mid a \in \mathit{Name}_{\mathrm{IO}} \wedge \underline{e} \subseteq \mathit{Exp}\}$, or the invisible action τ. We define the sets of free and bound variables occurring in an action as usual: denoted by $\mathit{fv}(\underline{e})$ the set of variables freely occurring in $\underline{e}$, we let $\mathit{fv}(a!(\underline{e})) = \mathit{fv}(\underline{e})$, $\mathit{bv}(a?(\underline{x})) = \underline{x}$, and $\mathit{fv}(\alpha) = \mathit{bv}(\alpha) = \emptyset$ in all the other cases. We pose $\mathit{name}(a?(\underline{x})) = \mathit{name}(a!(\underline{e})) = a$ and $\mathit{name}(\tau) = \tau$.

Definition 1.1. A symbolic transition graph [Hennessy and Lin 1995] (STG) is a tuple

$$(S, \longrightarrow, s_0; \mathit{Var}, \mathit{BExp} \times \mathit{Act})$$

where:

—S is a set of states each labeled with the set $\mathit{fv}(s) \subseteq \mathit{Var}$ of its free variables;
—$\longrightarrow \subseteq S \times (\mathit{BExp} \times \mathit{Act}) \times S$ is the transition relation;
—$s_0 \in S$ is the initial state.

The meaning of transition $s_1 \xrightarrow{\beta,\alpha} s_2$, where $\mathit{fv}(\alpha) \cup \mathit{fv}(\beta) \subseteq \mathit{fv}(s_1)$ and $\mathit{fv}(s_2) \subseteq \mathit{fv}(s_1) \cup \mathit{bv}(\alpha)$, is that s_2 can be reached from s_1 by performing α whenever β is satisfied at s_1. ■

As can be seen, the model is called symbolic because the variables occurring in input actions do not get instantiated and the expressions occurring in output actions do not get evaluated. This keeps the model finite for many process descriptions in the case of infinite value domains.

As pointed out in [Lin 1996], still many intuitively simple processes cannot be represented as finite STGs. As an example, consider the following process outputting all the integer numbers following a given one:

$$A(x) \stackrel{\Delta}{=} a!(x).A(x+1)$$

The STG for process $A(x)$ has countably many states representing $A(x+n)$ with outgoing transition

$$A(x+n) \xrightarrow{\mathit{true},a!(x+n)} A(x+n+1)$$

for all $n \in \mathbb{N}$. As a solution to this problem, in [Lin 1996] it was proposed to associate an assignment with each transition to be applied before executing the related action.

Definition 1.2. A symbolic transition graph with assignment [Lin 1996] (STGA) is a tuple

$$(S, \longrightarrow, s_0; \mathit{Var}, \mathit{BExp} \times \mathit{Sub} \times \mathit{Act})$$

where $\longrightarrow \subseteq S \times (\mathit{BExp} \times \mathit{Sub} \times \mathit{Act}) \times S$. The meaning of transition $s_1 \xrightarrow{\beta,\sigma,\alpha} s_2$ with $\sigma \equiv \underline{x} := \underline{e}$, where $\mathit{fv}(\alpha) \subseteq \underline{x}$ and $\mathit{fv}(\beta) \cup \mathit{fv}(\underline{e}) \subseteq \mathit{fv}(s_1)$ and $\mathit{fv}(s_2) \subseteq \underline{x} \cup \mathit{bv}(\alpha)$, differs from the meaning of $s_1 \xrightarrow{\beta,\alpha} s_2$ because in addition the free variables $\underline{x}$ at s_2 are assigned the values of $\underline{e}$ evaluated at s_1 before executing α. ■

In a STGA transition, σ is used to update the free variables both in α (because $\mathit{fv}(\alpha) \subseteq \underline{x}$ with σ applied before executing α) and in s_2. As an example, the STGA for process $A(x)$ is shown in Fig. 1(a), where each state is labeled with the set of its free variables. STGA are equipped with bisimulation equivalence checking algorithms developed in [Lin 1996; 1998].

As later observed in [Li and Chen 1999], a symbolic model for process $A(x)$ like the one in Fig. 1(a) is not intuitive, as one would expect a more compact symbolic model like the one in Fig. 1(b). To achieve that, in [Li and Chen 1999] the following variant of STGA was defined, where the assignments are applied after (rather than before) executing the related actions.

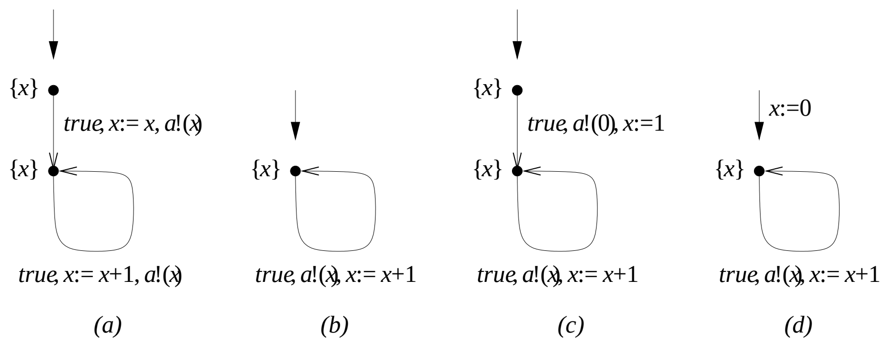

Fig. 1. Symbolic transition graphs with assignment for $A(x) \stackrel{\Delta}{=} a!(x).A(x+1)$ and $A(0)$

Definition 1.3. The variant of symbolic transition graph with assignment of [Li and Chen 1999] (STGA' for short) is a tuple

$$(S, \longrightarrow, s_0; \mathit{Var}, \mathit{BExp} \times \mathit{Act} \times \mathit{Sub})$$

where $\longrightarrow \subseteq S \times (\mathit{BExp} \times \mathit{Act} \times \mathit{Sub}) \times S$. The meaning of transition $s_1 \xrightarrow{\beta,\alpha,\sigma} s_2$ with $\sigma \equiv \underline{x} := \underline{e}$, where $\mathit{fv}(\alpha) \cup \mathit{fv}(\beta) \cup \mathit{fv}(\underline{e}) \subseteq \mathit{fv}(s_1)$ and $\mathit{fv}(s_2) \subseteq \mathit{fv}(s_1) \cup \underline{x} \cup \mathit{bv}(\alpha)$, [1] differs from the meaning of $s_1 \xrightarrow{\beta,\sigma,\alpha} s_2$ because σ is evaluated after (rather than before) executing α. ■

In a STGA' transition, σ no longer applies to the free variables in α but only to the free variables in s_2. STGA' are equipped with strong, weak and observational bisimulation equivalence checking algorithms developed in [Li and Chen 1999].

In [Li and Chen 1999] it is also observed that some simple processes still have not an intuitive representation with STGA'. As an example, for process $A(0)$ the STGA' is shown in Fig. 1(c), while one would expect a more compact symbolic model like the one in Fig. 1(d). To achieve that, in this paper we advocate the use of symbolic transition graphs with lookahead assignment, which were introduced in a preliminary work [Bernardo 1997] prior to [Li and Chen 1999] and are characterized by the presence of an initial assignment.

Definition 1.4. A symbolic transition graph with lookahead assignment (STGLA for short) is a tuple

$$(S, \longrightarrow, s_0, \sigma_0; \mathit{Var}, \mathit{BExp} \times \mathit{Act} \times \mathit{Sub})$$

where $(S, \longrightarrow, s_0; \mathit{Var}, \mathit{BExp} \times \mathit{Act} \times \mathit{Sub})$ is a STGA' and $\sigma_0 \equiv \underline{x}_0 := \underline{v}_0 \in \mathit{Sub}$ is the initial assignment with $\underline{v}_0$ vector of values and $\underline{x}_0 \subseteq \mathit{fv}(s_0)$. ■

The STGLA for process $A(0)$ is depicted in Fig. 1(d). The symbolic model of STGLA, which inherits by definition the bisimulation equivalence checking algorithms for STGA', combines the idea of applying the assignments after executing the actions with the idea of having an initial assignment for the initial state. From

[1]In [Li and Chen 1999] the additional constraint $\mathit{bv}(\alpha) \cap (\underline{x} \cup \mathit{fv}(\underline{e})) = \emptyset$ was imposed just to simplify the STGA' symbolic rules, especially in the case of the parallel composition operator. Here we omit this constraint for reasons related to the well formedness of the assignments generated by our symbolic rules, as we shall see in Sect. 2.3.3.

the point of view of a value passing process description, this can be seen as an attempt to keep symbolic the initial term of the description as well as every derivative term before generating its outgoing transitions, hence the terminology lookahead assignment. By keeping a term symbolic we just mean that the variables occurring in the term are not instantiated.

Structure of the Paper. This paper, which is a revised and extended version of the main ideas contained in [Bernardo 1997], is organized as follows. In Sect. 2 we define a set of symbolic semantic rules that associate a finite STGLA with every term of a useful fragment of value passing CCS with possibly infinite data domains, without making any assumption on the variable names – unlike the rules in [Lin 1996; Li and Chen 1999] – thanks to the transparent introduction of the dot notation and localities [Boudol and Castellani 1988]. Transparency is an important issue when implementing the symbolic semantic rules in a software tool, as the variables must be correctly handled in the presence of the freedom for the user to choose the same variable names across different process terms. This achieves objective (1). In addition, the symbolic semantic rules allow us to apply the verification algorithms for STGLA inherited from [Li and Chen 1999] to the terms of the considered fragment of value passing CCS, thereby realizing objective (2). In Sect. 3 we prove that the symbolic semantic rules are correct w.r.t. both the usual concrete semantic rules and the assignment application order. As far as the latter issue is concerned, we point out that the assignment application order is important in the simulation based analysis of value passing process descriptions, because simulation requires the transitions to be actually executed, hence the related assignments to be actually applied. In Sect. 4 we address the achievement of objective (3) by showing that the STGLA generated by the symbolic semantic rules for the terms of the considered fragment of value passing CCS are optimal w.r.t. a certain compactness criterion, thereby improving on [Hennessy and Lin 1995; Lin 1996; Li and Chen 1999], and we report on two examples that illustrate the advantages gained thanks to the compactness of the generated STGLA. Finally, in Sect. 5 we draw some conclusions.

2. STGLA SEMANTICS FOR A FRAGMENT OF VALUE PASSING CCS

2.1 Syntax

In this section we define the syntax of a fragment of value passing CCS. The classical LTS based concrete semantic rules and the novel STGLA based symbolic semantic rules for the considered language will be presented in the next two sections.

Definition 2.1. The set $\mathcal{G}$ of process terms of the fragment of value passing CCS is generated by

$$\begin{array}{rcl} E & ::= & Q \mid E \backslash L \mid E[\varphi] \mid E \,|\, E \mid A(\underline{e}) \\ Q & ::= & \sum_{i \in I} \alpha_i . E_i \mid \mathrm{if}\,(\beta)\,\mathrm{then}\,Q\,\mathrm{else}\,Q \end{array}$$

where:

—I is a finite set of indices, with the summation being denoted by $\underline{0}$ whenever $I = \emptyset$.

—$L \subseteq \mathit{Name}_{\mathrm{IO}}$.

—φ belongs to a set *Rel* of action relabeling functions that preserve the action observability, structure, and parameters: $\varphi(\tau) = \tau$ and $\varphi(a?(\underline{x})) = a'?(\underline{x})$ and $\varphi(a!(\underline{e})) = a'!(\underline{e})$ with $a' \in \mathit{Name}_{\mathrm{IO}}$.

—A belongs to a set *Const* of constants and is equipped with a defining equation $A(\underline{x}; \underline{y}) \stackrel{\Delta}{=} E$, with $\underline{x}$ a possibly empty vector of formal parameters and $\underline{y}$ a possibly empty vector of local variables, such that:
  —the formal parameters are correctly used, i.e. no formal parameter in $\underline{x}$ occurs in an input action of E;
  —the local variables are input guarded, i.e. no local variable $y \in \underline{y}$ occurs in E outside the scope of an input action prefix operator of the form $a?(\underline{y'}).\_$ with $y \in \underline{y'} \subseteq \underline{y}$;
  —the body does not contain occurrences of undeclared variables;
  —the body is guarded, i.e. all the constant invocations occurring in E are (or can be rewritten according to their defining equations in order to appear) within the scope of an action prefix operator $\alpha.\_$;
  —the body is not recursive over static operators, i.e. no recursive constant invocation is allowed in E across the operators $\_\backslash L$, $\_[\varphi]$, and $\_\,|\,\_$. ■

In the syntax above, the guarded alternative composition operator "$\sum_{i \in I} \alpha_i.\_$" expresses a nondeterministic choice among $|I|$ terms whose outermost operator is an action prefix operator: term $\sum_{i \in I} \alpha_i.E_i$ executes one action among $\alpha_{i_1}, \dots, \alpha_{i_{|I|}}$, say α_{i_j}, and then behaves as term E_{i_j}. The conditional operator "if (β) then _ else _" expresses a choice between two terms that is driven by the boolean expression β: term if (β) then Q_1 else Q_2 behaves as either term Q_1 or term Q_2 depending on whether β evaluates to *true* or *false*. The restriction operator "$\_\backslash L$" prevents the actions in L from occurring: term $E \backslash L$ behaves as term E except that each action $a \in L$ executable by E can no longer be executed. The relabeling operator "$\_[\varphi]$" changes the actions according to φ: term $E[\varphi]$ behaves as term E except that each action a executed by E is turned into $\varphi(a)$. The parallel composition operator "$\_\,|\,\_$" expresses the concurrent execution of two terms: term $E_1 \,|\, E_2$ asynchronously executes actions of E_1 or E_2 and synchronously executes pairs composed of an input (output) action of E_1 and an output (input) action of E_2 having the same name, giving rise to a τ action and the value passing from E_2 (E_1) to E_1 (E_2). The constant invocation $A(\underline{e})$ behaves as term E in the defining equation $A(\underline{x}; \underline{y}) \stackrel{\Delta}{=} E$, with the formal parameters $\underline{x}$ replaced by the actual parameters $\underline{e}$.

Besides the verbosity of the conditional operator and the explicit indication of the local variables in the constant defining equations, there are two major differences between the syntax in Def. 2.1 and the syntax of value passing CCS. The first difference is that the action prefix operator and the choice operator have been combined into a single operator called guarded alternative composition. This operator rules out CCS terms of the form $\sum_{i \in I} E_i$ with at least one summand not starting with an action prefix. In practice, this does not reduce the modeling power. In addition, it makes it possible the generation of the lookahead assignments in the symbolic semantic rules and guarantees the obtainment of compact STGLA, as we shall see in Sect. 2.3.4.

The second difference is that the conditional operator requires its two operand terms to be either guarded alternative compositions or conditionals themselves.

From the modeling viewpoint, this rules out useful regular value passing CCS terms like e.g.

$$A(x;y) \stackrel{\Delta}{=} c?(y).\,\text{if}\,(y<0)\,\text{then}\,A(x-y)\,\text{else}\,d!(x).A(x+y)$$

However, terms like the previous one may somehow be restored by inserting suitable τ actions:

$$A'(x;y) \stackrel{\Delta}{=} c?(y).\,\text{if}\,(y<0)\,\text{then}\,\tau.A'(x-y)\,\text{else}\,d!(x).A'(x+y)$$

Should this result in τ loops with assignments, which cannot be dealt with by the weak bisimulation equivalence checking algorithm of [Li and Chen 1999], one may try to resort to the algorithm in [Lin 1998]. Furthermore, as we shall see in Sect. 2.3.4, the guardedness constraint on the conditional operator makes it possible the generation of the lookahead assignments in the symbolic semantic rules and guarantees the obtainment of compact STGLA, with possible restoring τ actions not increasing the state space.

2.2 Concrete Semantics

The usual concrete semantics for $\mathcal{G}$ maps terms to LTSs. In this process, the occurrences of the free variables arising in the derivative term of an input action prefix or a constant invocation must be instantiated via syntactical substitutions. To formalize this, we define below the result of the application of a syntactical substitution – called an evaluation – to the free variables occurring in a term.

Definition 2.2. Let $F \in \mathcal{G}$. The set $fv(F)$ of free variables occurring in F is defined by structural induction as follows:

$$\begin{array}{rcl}
fv(\sum_{i\in I}\alpha_i.E_i) & = & \bigcup_{i\in I} fv(\alpha_i.E_i)\\
fv(\text{if}\,(\beta)\,\text{then}\,Q_1\,\text{else}\,Q_2) & = & fv(\beta)\cup fv(Q_1)\cup fv(Q_2)\\
fv(E\backslash L) & = & fv(E)\\
fv(E[\varphi]) & = & fv(E)\\
fv(E_1 \mid E_2) & = & fv(E_1)\cup fv(E_2)\\
fv(A(\underline{e})) & = & fv(\underline{e})
\end{array}$$

where:

$$\begin{array}{rcl}
fv(a?(\underline{x}).E) & = & fv(E) - \underline{x}\\
fv(a!(\underline{e}).E) & = & fv(E)\cup fv(\underline{e})\\
fv(\tau.E) & = & fv(E)
\end{array}$$

■

Definition 2.3. Let $Eval = \{\rho : Var \longrightarrow Var \cup Val \mid \rho(x)\in Var \Longrightarrow \rho(x) = x\}$ be a set of evaluations, which are naturally lifted from Var to $Expr$. Given $F\in\mathcal{G}$ and $\rho \in Eval$, term $F\rho$ is defined by structural induction as follows:

$$\begin{array}{rcl}
(\sum_{i\in I}\alpha_i.E_i)\rho & = & \sum_{i\in I}((\alpha_i.E_i)\rho)\\
(\text{if}\,(\beta)\,\text{then}\,Q_1\,\text{else}\,Q_2)\rho & = & \text{if}\,(\rho(\beta))\,\text{then}\,Q_1\rho\,\text{else}\,Q_2\rho\\
(E\backslash L)\rho & = & E\rho\backslash L\\
(E[\varphi])\rho & = & E\rho[\varphi]\\
(E_1 \mid E_2)\rho & = & E_1\rho \mid E_2\rho\\
A(\underline{e})\rho & = & A(\rho(\underline{e}))
\end{array}$$

where:

$$\begin{array}{rcl}
(a?(\underline{x}).E)\rho & = & a?(\underline{x}).E\rho' \text{ with } \rho'(x) = x \text{ and } \rho'(z) = \rho(z) \text{ for } z \neq x\\
(a!(\underline{e}).E)\rho & = & a!(\rho(\underline{e})).E\rho\\
(\tau.E)\rho & = & \tau.E\rho
\end{array}$$

We define the domain of ρ as $dom(\rho) = \{x\in Var \mid \rho(x)\in Val\}$. ■

Following [Hennessy and Lin 1995], the concrete semantics for $\mathcal{G}$ can be defined in two different ways depending on when the instantiation of the free variables arising in the derivative term of an input action prefix takes place. In the early case, the instantiation takes place soon through the application of a substitution in the semantic rule for the input action prefix operator. In the late case, instead, the instantiation takes place only through the application of a substitution in the semantic rule for the synchronization. In this paper we restrict ourselves to the concrete early semantics, which we simply call concrete semantics for the sake of conciseness. The concrete semantics for $\mathcal{G}$ is shown in Table I. Every state of the concrete semantics is a pair composed of a term and an evaluation that closes the term, i.e. an evaluation for the free variables occurring in the term that causes every expression occurring in an output action or a constant invocation of the term to be turned into a value. As a consequence, the concrete semantics of a specific term is defined w.r.t. an initial evaluation that closes the term.

Definition 2.4. The operational concrete semantics of $E \in \mathcal{G}$ w.r.t. $\rho_0 \in \mathit{Eval}$ such that $\mathit{fv}(E) \subseteq \mathit{dom}(\rho_0)$ is the LTS

$$[\![E]\!]_{\mathrm{c},\rho_0} = (S_{\mathrm{c},\rho_0,E}, \mathit{Act}_{\mathrm{c}}, \longrightarrow_{\mathrm{c},\rho_0,E}, \langle E, \rho_0\rangle)$$

where:

—$S_{\mathrm{c},\rho_0,E}$ is the least subset of $\mathcal{G}$ such that:
  —$\langle E, \rho_0\rangle \in S_{\mathrm{c},\rho_0,E}$;
  —if $\langle E_1, \rho_1\rangle \in S_{\mathrm{c},\rho_0,E}$ and $\langle E_1, \rho_1\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E_2, \rho_2\rangle$, then $\langle E_2, \rho_2\rangle \in S_{\mathrm{c},\rho_0,E}$.

—$\mathit{Act}_{\mathrm{c}} = \mathit{Act}_{\mathrm{I,c}} \cup \mathit{Act}_{\mathrm{O,c}} \cup \{\tau\}$ with

$$\begin{array}{rcl} \mathit{Act}_{\mathrm{I,c}} & = & \{a?(\underline{v}) \mid a \in \mathit{Name}_{\mathrm{IO}} \wedge \underline{v} \subseteq \mathit{Val}\} \\ \mathit{Act}_{\mathrm{O,c}} & = & \{a!(\underline{v}) \mid a \in \mathit{Name}_{\mathrm{IO}} \wedge \underline{v} \subseteq \mathit{Val}\} \end{array}$$

—$\longrightarrow_{\mathrm{c},\rho_0,E}$ is the restriction of $\longrightarrow_{\mathrm{c}}$ to $S_{\mathrm{c},\rho_0,E} \times \mathit{Act}_{\mathrm{c}} \times S_{\mathrm{c},\rho_0,E}$. ■

It is well known that, in the case of infinite data domains, the concrete semantics may give rise to infinite LTSs because:

—the first rule for the guarded alternative composition operator (input action prefix) may result in an infinite branching as there is a syntactical substitution for every possible vector of values, [2] and

—a constant may be invoked infinitely many times with different actual parameters, with each invocation resulting in a different state according to the syntactical substitution in the rule for the constant invocation operator.

2.3 Symbolic Semantics

In this section we define a symbolic semantics that maps $\mathcal{G}$ to STGLA. The idea underlying such a semantics is to overcome the problems with infinite data domains by suitably modifying those rules of the operational concrete semantics that contain syntactical substitutions. Following the guidelines in [Hennessy and Lin 1995; Lin 1996; Li and Chen 1999], we keep symbolic the input actions labeling the

[2] This can be avoided by resorting to the concrete late semantics.

$$\langle \sum_{i\in I} \alpha_i.E_i, \rho\rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E_j, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \quad \alpha_j = a?(\underline{x}), \underline{v} \subseteq \mathit{Val}$$

$$\langle \sum_{i\in I} \alpha_i.E_i, \rho\rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E_j, \rho\rangle \quad \alpha_j = a!(\underline{e}), \rho(\underline{e}) = \underline{v}$$

$$\langle \sum_{i\in I} \alpha_i.E_i, \rho\rangle \xrightarrow{\tau}_{\mathrm{c}} \langle E_j, \rho\rangle \quad \alpha_j = \tau$$

$$\frac{\langle Q_1, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle}{\langle \text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle} \quad \rho(\beta)$$

$$\frac{\langle Q_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle}{\langle \text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle} \quad \neg\rho(\beta)$$

$$\frac{\langle E, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle}{\langle E\backslash L, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E'\backslash L, \rho'\rangle} \quad \mathit{name}(\alpha) \notin L$$

$$\frac{\langle E, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle}{\langle E[\varphi], \rho\rangle \xrightarrow{\varphi(\alpha)}_{\mathrm{c}} \langle E'[\varphi], \rho'\rangle}$$

$$\frac{\langle E_1, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E_1', \rho'\rangle}{\langle E_1 \mid E_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E_1' \mid E_2, \rho'\rangle}$$

$$\frac{\langle E_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E_2', \rho'\rangle}{\langle E_1 \mid E_2, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E_1 \mid E_2', \rho'\rangle}$$

$$\frac{\langle E_1, \rho\rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E_1', \rho_1'\rangle \quad \langle E_2, \rho\rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E_2', \rho_2'\rangle}{\langle E_1 \mid E_2, \rho\rangle \xrightarrow{\tau}_{\mathrm{c}} \langle E_1' \mid E_2', \rho_1' \cup \rho_2'\rangle}$$

$$\frac{\langle E_1, \rho\rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E_1', \rho_1'\rangle \quad \langle E_2, \rho\rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E_2', \rho_2'\rangle}{\langle E_1 \mid E_2, \rho\rangle \xrightarrow{\tau}_{\mathrm{c}} \langle E_1' \mid E_2', \rho_1' \cup \rho_2'\rangle}$$

$$\frac{\langle E, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle}{\langle A(\underline{e}), \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle} \quad A(\underline{x};\underline{y}) \stackrel{\Delta}{=} E, \rho(\underline{e}) = \underline{v}$$

Table I. Operational concrete semantics

transitions and we encode through assignments the syntactical substitutions due to constant invocations. This is transparently accomplished by combining the use of the dot notation and localities for the variables with a term unfolding procedure for generating the assignments.

2.3.1 *Dot Notation and Localities for Variables.* Before presenting the symbolic rules, some comments on the treatment of variables are in order. When moving from the concrete to the symbolic semantics, the variables are no longer instantiated within the terms. Instead, they are given suitable values by means of assignments labeling the transitions. In order for such assignments to be correct, proper names should be given to the variables so that clashes are avoided. For this purpose, the symbolic rules that appeared in the literature [Lin 1996; Li and Chen 1999] assume disjoint name spaces for processes composed in parallel. Instead, the symbolic rules we shall introduce do not make any assumption on the variable names because of some technicalities – dot notation and localities – that are introduced in a way that is completely transparent to the user of the process description language. This not only gives the freedom of choosing the same name for variables declared in different constants, but is necessary for easily automating in a software tool the correct management of the variables declared in several occurrences of the same constant composed in parallel.

To differentiate the variables with the same name that are declared in different constants, we use the dot notation. A term of $\mathcal{G}$ is preprocessed in such a way that the name of every variable is prefixed by the name of the constant in which the variable is declared, with a dot separating the two names. If we consider e.g.

$$\begin{aligned} A(x;\,) &\stackrel{\Delta}{=} a!(x).(A_1(x+1) \mid A_2(x+2)) \\ A_1(z;\,) &\stackrel{\Delta}{=} a_1!(z).A_1(z+1) \\ A_2(z;\,) &\stackrel{\Delta}{=} a_2!(z).A_2(z+1) \end{aligned}$$

where parameter z of A_1 must be initialized to $x+1$ while parameter z of A_2 must be initialized to $x+2$, we observe that no clash between the two parameters arises because the former is renamed $A_1.z$ while the latter is renamed $A_2.z$.

To differentiate the variables that are declared in different occurrences of the same constant composed in parallel, we further prefix the dotted variable names with localities. In the spirit of [Boudol and Castellani 1988], localities express positions w.r.t. occurrences of the parallel composition operator within a term.

Definition 2.5. The set of localities is defined by $Loc = \{\swarrow, \searrow\}^*$ and ranged over by ψ. We extend Var in order to include for all ψ and x the variable ψx. We extend Exp in order to include for all ψ and e the expression ψe, which is obtained from e by replacing each variable x occurring in it with ψx. We extend Sub in order to include for all ψ and σ the substitution $\psi\sigma$, which is obtained from σ by replacing each variable x occurring in it with ψx. When a locality ψ is applied to a set, it is intended that ψ must be applied to each element in the set. ■

If we consider e.g.

$$\begin{aligned} A(x;\,) &\stackrel{\Delta}{=} a!(x).(A'(x+1) \mid A'(x+2)) \\ A'(z;\,) &\stackrel{\Delta}{=} a'!(z).A'(z+1) \end{aligned}$$

where parameter z of the left occurrence of A' must be initialized to $x+1$ while parameter z of the right occurrence of A' must be initialized to $x+2$, we observe that no clash between the two parameters arises because the former is renamed $\swarrow A'.z$ while the latter is renamed $\searrow A'.z$. Although localities would be needed in principle for every binary operator, their introduction is problematic in the case of the traditional choice operator and conditional operator. In our framework the problem has been avoided by considering a fragment of value passing CCS that allows only guarded summands and branches, so that terms like $A'(x+1)+A'(x+2)$ and if (β) then $A'(x+1)$ else $A'(x+2)$ are never encountered. We shall return on this issue in Sect. 2.3.4.

In conclusion, the name of every variable is prefixed by a string in the concatenation of *Loc* and *Const*. While the constant name based dot notation is introduced statically via a simple term preprocessing, as we shall see in Sect. 2.3.3 the localities are introduced dynamically during the application of the symbolic rules.

2.3.2 *Assignment Generation through Term Unfolding.* To cope with infinite data domains, the syntactical substitutions occurring in the rules of the concrete semantics must be encoded through assignments instead of being applied to terms. Once decided that the input actions labeling the transitions must stay symbolic, we have to deal with the syntactical substitutions due to the concrete rule for the constant invocation. According to objective (3) stated in the introduction, the syntactical substitution occurring in the concrete rule for the constant invocation must absolutely be avoided. The ideal situation is to have for each constant A defined by $A(\underline{x};\underline{y}) \stackrel{\Delta}{=} E$ one single state $A(\underline{x};\underline{y})$ instead of a possibly infinite set of states $\{A(\underline{e}_j)\}_j$. This can easily be achieved by defining the symbolic rules in such a way that they look ahead in the derivative term of each generated transition to expand every constant invocation just before the constant is invoked, so that there will be no need for a symbolic rule for the constant invocation operator. This is carried out through the following function to be applied both to the initial term of a value passing specification and to the derivative terms of the symbolic transitions generated for the value passing specification.

Definition 2.6. Let $\mathcal{G}_{\mathrm{u}}$ be the set of terms of $\mathcal{G}$ not having constants invocations outside the scope of an action prefix operator. We define function $\mathit{unfold} : \mathcal{G} \longrightarrow \mathcal{G}_{\mathrm{u}}$ by structural induction as follows:

$$\begin{array}{rcll}
\mathit{unfold}(\sum_{i\in I}\alpha_i.E_i) & = & \sum_{i\in I}\alpha_i.E_i & \\
\mathit{unfold}(\text{if }(\beta)\text{ then }Q_1\text{ else }Q_2) & = & \text{if }(\beta)\text{ then }Q_1\text{ else }Q_2 & \\
\mathit{unfold}(E\backslash L) & = & \mathit{unfold}(E)\backslash L & \\
\mathit{unfold}(E[\varphi]) & = & \mathit{unfold}(E)[\varphi] & \\
\mathit{unfold}(E_1 \mid E_2) & = & \mathit{unfold}(E_1) \mid \mathit{unfold}(E_2) & \\
\mathit{unfold}(A(\underline{e})) & = & \mathit{unfold}(E) & \text{if } A(\underline{x};\underline{y}) \stackrel{\Delta}{=} E \qquad \blacksquare
\end{array}$$

It is worth noting that $A(\underline{e})$, where $A(\underline{x};\underline{y}) \stackrel{\Delta}{=} E$, is replaced neither by $A(\underline{x};\underline{y})$ nor by E, but by $\mathit{unfold}(E)$. This is necessary to deal with sequences of constant invocations like e.g.

$$\begin{array}{rcl}
A(x;) & \stackrel{\Delta}{=} & A'(x+3) \\
A'(z;) & \stackrel{\Delta}{=} & a'!(z).A'(z+1)
\end{array}$$

We also observe that the unfolding process terminates because we consider only guarded terms, which can be rewritten according to constant defining equations in such a way that no constant invocation occurs outside the scope of an action prefix operator.

The unfolding process must be accompanied by a process that records the assignments generated along the way. Observed that the assignments generated in the case of a synchronization will be handled by the corresponding symbolic rule, there are two sources of assignments at unfolding time. First, we have to keep track of the assignments of actual parameters to formal parameters arising from the constant invocations. Since such assignments are built just before the constant invocation takes place, this procedure is consistent with the fact that in STGLA (and STGA') the assignments are evaluated after executing the actions, whereas it cannot be applied to STGA as it may cause errors. Second, when unfolding a parallel composition, it is necessary to generate suitable technical assignments for the formal parameters and the local variables of the constant whose defining equation contains the parallel composition being unfolded. Consider for instance the following process

$$\begin{array}{rcl} A(x;y) & \stackrel{\Delta}{=} & (a!(x).A'(x+1) \mid a?(y).A'(y+2))\backslash\{a\} \\ A'(z;) & \stackrel{\Delta}{=} & a'!(z).A'(z+1) \end{array}$$

Synchronizing $a!(A.x)$ with $a?(A.y)$ and unfolding $A'(A.x{+}1) \mid A'(A.y{+}2)$ produce the assignment $\searrow A.y := \swarrow A.x \triangleright \swarrow A'.z := \swarrow A.x{+}1 \triangleright \searrow A'.z := \searrow A.y{+}2$. However, the original formal parameter is $A.x$. In order for the assignment above to be correctly evaluated, a link has to be established between $A.x$ and $\swarrow A.x$. This is achieved by means of the insertion of technical assignment $\swarrow A.x := A.x$ at the beginning of the previous assignment.

To record the assignments generated during the unfolding process, we need to build and navigate the locality tree of the term being unfolded. Intuitively, the locality tree of a term specifies for each locality of the term the constant in which the variables occurring in that locality have been declared. In other words, the locality tree of a term is a binary tree that keeps track of the positions in the syntactical structure of the term in which parallel composition operators have been encountered, together with the names of the constants in which those operators occur (or $\bot$ if they do not occur within the scope of any constant defining equation). The set of locality trees is represented by the set $\mathit{CLoc} = \{\xi \mid \xi : \mathit{Loc} \nrightarrow \mathit{Const} \cup \{\bot\}\}$ of partial locality functions associating a constant (or $\bot$) with each locality. We formalize below the assignment recording.

Definition 2.7. We define function $\mathit{record} : \mathcal{G} \times \mathit{CLoc} \times \mathit{Loc} \longrightarrow \mathit{CLoc} \times \mathit{Sub}$ by structural induction as follows:

$$\begin{array}{rcl} \mathit{record}(\sum_{i \in I} \alpha_i.E_i, \xi, \psi) & = & (\xi, \varepsilon) \\ \mathit{record}(\text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \xi, \psi) & = & (\xi, \varepsilon) \\ \mathit{record}(E\backslash L, \xi, \psi) & = & \mathit{record}(E, \xi, \psi) \\ \mathit{record}(E[\varphi], \xi, \psi) & = & \mathit{record}(E, \xi, \psi) \end{array}$$

$$
\begin{array}{rcl}
\mathit{record}(E_1 \mid E_2, \xi, \psi) & = & (\xi_1 \cup \xi_2, \sigma_{\mathrm{tech},1,2} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2) \\
 & & \text{where :} \\
 & & \quad (\xi_1, \sigma_1) = \mathit{record}(E_1, \xi\{\swarrow \psi \mapsto \xi(\psi)\}, \swarrow\psi) \\
 & & \quad (\xi_2, \sigma_2) = \mathit{record}(E_2, \xi\{\searrow \psi \mapsto \xi(\psi)\}, \searrow\psi) \\
 & & \quad \sigma_{\mathrm{tech},1,2} = \begin{cases} \swarrow\psi\underline{xy} := \searrow\psi\underline{xy} := \psi\underline{xy} \\ \quad \text{if } \xi(\psi) = A \text{ and} \\ \quad \xi(\swarrow\psi), \xi(\searrow\psi) \text{ undefined} \\ \quad \text{with } A(\underline{x}; \underline{y}) \stackrel{\Delta}{=} E \\ \varepsilon \\ \quad \text{otherwise} \end{cases} \\
\mathit{record}(A(\underline{e}), \xi, \psi) & = & (\xi', \underline{x} := \underline{e} \triangleright \sigma') \\
 & & \text{where :} \\
 & & \quad A(\underline{x}; \underline{y}) \stackrel{\Delta}{=} E \\
 & & \quad (\xi', \sigma') = \mathit{record}(E, \xi\{\psi \mapsto A\}, \psi)
\end{array}
$$

If $\mathit{record}(E, \xi, \psi) = (\xi', \sigma')$, ξ' is denoted by $\mathit{loctree}(E, \xi, \psi)$ while σ' is denoted by $\mathit{assign}(E, \xi, \psi)$. ■

Let us explain the use of parameters ξ and ψ of function *record* during the application of the symbolic rules to a given term E. Parameter ξ represents the locality tree of E. Parameter ψ, instead, is used to navigate the locality tree described by ξ, i.e. it denotes the current locality (which is a leaf of ξ). The initial locality tree of E is $\{(\varepsilon, \mathit{const}(E))\}$, where ε is the empty locality while $\mathit{const}(E)$ is A if E is a constant invocation $A(\underline{e})$, $\perp$ otherwise. The locality tree of E is then updated by the symbolic rules whenever the unfolding of a derivative term of E takes place, with the navigator initially set to ε before each application of the rules. An update of the locality tree can refer to either the constant labeling a leaf, in the case when a further constant invocation is encountered, or the structure of the tree, in the case when a further occurrence of the parallel composition operator is encountered. If an invocation of constant A is encountered in the locality tree ξ at locality ψ, the constant labeling leaf ψ is changed from $\xi(\psi)$ to A. If a parallel composition is encountered in the locality tree ξ at locality ψ, then the two new leaves $\swarrow\psi$ and $\searrow\psi$ are added as children of ψ and labeled with $\xi(\psi)$, i.e. with the same constant as their parent. Moreover, as can be seen from the definition of $\sigma_{\mathrm{tech},1,2}$, two technical assignments must be generated if the subterm under examination is within the scope of a constant defining equation ($\xi(\psi) = A$) and the parallel composition operator under consideration has not been encountered before ($\xi(\swarrow\psi), \xi(\searrow\psi)$ undefined). We recall from Def. 2.1 that a term cannot contain occurrences of undeclared variables. As a consequence, no variable can occur outside the scope of a constant defining equation, hence no technical assignment must be generated when encountering for the first time a parallel composition operator outside the scope of a constant defining equation ($\xi(\psi) = \perp$).

2.3.3 *Symbolic Rules.* The symbolic rules producing STGLA for terms of $\mathcal{G}_{\mathrm{u}}$ are shown in Table II. Note that each state is a triple $\langle E, \xi, \psi\rangle$ and that the three auxiliary functions *unfold*, *loctree*, and *assign* are used only in the axiom for the guarded alternative composition operator, while nontrivial boolean guards can be introduced only by the two inference rules for the conditional operator. Note also that the localities prefixing the names of the variables occurring in the assignments are recovered by the four inference rules for the parallel composition operator. In addition, the two inference rules for the synchronization can also generate new assignments due to value passing. Finally, note that there is no inference rule for the constant invocation operator because function *unfold* is applied just before a constant is invoked.

Definition 2.8. The operational symbolic semantics of $E \in \mathcal{G}$ is the STGLA

$$\begin{aligned}[\![E]\!]_{\mathrm{s}} = \; & (S_{\mathrm{s},E}, \longrightarrow_{\mathrm{s},E}, \\ & \langle \mathit{unfold}(E), \mathit{loctree}(E, \{(\varepsilon, \mathit{const}(E))\}, \varepsilon), \varepsilon\rangle, \\ & \mathit{assign}(E, \{(\varepsilon, \mathit{const}(E))\}, \varepsilon); \\ & \mathit{Var}, \mathit{BExp} \times \mathit{Act} \times \mathit{Sub})\end{aligned}$$

where:

—$S_{\mathrm{s},E}$ is the least subset of $\mathcal{G}_{\mathrm{u}} \times \mathit{CLoc} \times \mathit{Loc}$ such that:
  —$\langle \mathit{unfold}(E), \mathit{loctree}(E, \{(\varepsilon, \mathit{const}(E))\}, \varepsilon), \varepsilon\rangle \in S_{\mathrm{s},E}$;
  —if $\langle E_1, \xi_1, \psi\rangle \in S_{\mathrm{s},E}$ and $\langle E_1, \xi_1, \psi\rangle \xrightarrow{\beta,\alpha,\sigma}_{\mathrm{s}} \langle E_2, \xi_2, \psi\rangle$, then $\langle E_2, \xi_2, \psi\rangle \in S_{\mathrm{s},E}$.
—$\longrightarrow_{\mathrm{s},E}$ is the restriction of $\longrightarrow_{\mathrm{s}}$ to $S_{\mathrm{s},E} \times (\mathit{BExp} \times \mathit{Act} \times \mathit{Sub}) \times S_{\mathrm{s},E}$. ■

We observe that, for all $E \in \mathcal{G}$, $[\![E]\!]_{\mathrm{s}}$ is indeed a STGLA because the assignments labeling the transitions are well formed. Unlike [Li and Chen 1999], where in the definition of STGA' the additional constraint $\mathit{bv}(\alpha) \cap (\underline{x} \cup \mathit{fv}(\underline{e})) = \emptyset$ was imposed for each transition $s_1 \xrightarrow{\beta,\alpha,\underline{x}:=\underline{e}} s_2$ to simplify the symbolic rules especially in the case of the parallel composition operator, here no additional constraint is needed:

—In the case of a synchronization, constraint $\mathit{bv}(\alpha) \cap \underline{x} = \emptyset$ ensures that all the variables occurring in the left hand side of an assignment are distinct. In general, this may not necessarily be the case. To see why, consider a subterm of the defining equation of a constant A having the form $a?(\underline{x}).A(\underline{e})$, with some elements of $\underline{x}$ being formal parameters of A, that can synchronize with an output action of the form $a!(\underline{e}')$. The assignment labeling the synchronization transition may contain a first assignment to the formal parameters in $\underline{x}$ from the corresponding elements of $\underline{e}'$, followed by a second assignment to the same formal parameters arising from the unfolding of $A(\underline{e})$. In [Li and Chen 1999] this problem is solved at the semantic level by avoiding the unfolding of a constant invocation following an input action. In our framework, instead, the problem does not arise because, according to Def. 2.1, the formal parameters of a constant cannot occur in an input action, so in particular they cannot occur in an input action preceding a constant invocation. This syntax restriction, besides being reasonable from a modeling viewpoint, permits at the semantic level a uniform treatment of the unfolding of the constant invocations following input actions and the constant invocations following output actions and τ.

—In the case of a synchronization, constraint $\mathit{bv}(\alpha) \cap \mathit{fv}(\underline{e}) = \emptyset$ ensures the correct applicability of an assignment. In general, this may not necessarily be the case. To see why, consider the same situation as above, with $\underline{x}$ now being a vector of local vaiables of A. The assignment labeling the synchronization transition may contain a first assignment to the local variables in $\underline{x}$ from the expressions in $\underline{e}'$, followed by a second assignment arising from the unfolding of $A(\underline{e})$ in which some of the previously assigned local variables occur in the right hand side. In [Li and Chen 1999] this is a problem because there a substitution is defined as a function from *Var* to *Exp*, hence all the elements of a vector of assignments must be applied simultaneously. In the overall assignment of the synchronization transition above, this means that the value of $\underline{x}$ used in the second element of the assignment is still the old one instead of the new one coming from the first element of the assignment. In our framework, instead, the problem does not arise because a substitution is defined as a sequence of pairs in $\mathit{Var} \times \mathit{Expr}$, so the elements of an assignment are applied sequentially from left to right rather that simultaneously.

The following theorem shows that objectives (1) and (2) stated in the introduction have been achieved.

Theorem 2.9. *Let $E \in \mathcal{G}$. Then $[\![E]\!]_{\mathrm{s}}$ is a finite STGLA.*

Proof. From Def. 2.1 we recall that E is guarded and does not contain any recursive constant invocation across the static operators. The result then follows by observing that the symbolic rule for the guarded alternative composition operator avoids infinite branching – as input actions are kept symbolic – as well as infinitely many variants of the invocation of the same constant – because of the application of function *unfold* to the derivative of every action prefix operator. □

2.3.4 *More on Localities and Syntax Restrictions.* We recall from Sect. 2.3.1 that, while the dot notation is related to the definition of a constant – which is unique – the localities are related to the invocations of a constant – which can be multiple. Therefore, unlike the dot notation, the localities cannot be introduced through a static preprocessing of the constant defining equations, but during the application of the symbolic rules. As can be seen from Table II, the technique used to accomplish this is to introduce the localities in the assignments but not in the terms, with the four symbolic rules for the parallel composition operator taking care of recovering the localities prefixing the variables in the assignments. This technique has the advantage of resulting in a general symbolic semantics, which is parametrized w.r.t. the position of any locality tree in which a term is plugged. Another pleasant property of this technique is that function *unfold* is closed under the application of the symbolic rules. In other words, applying *unfold* to the derivative term of a transition in a top down fashion – instead of applying it in the symbolic rules – is the same as applying *unfold* to the derivative terms of the actions resulting in the transition and combining such unfolded derivative terms in a bottom up fashion – as is done in the symbolic rules.

Theorem 2.10. *Let $E \in \mathcal{G}_{\mathrm{u}}$, $\xi \in \mathit{CLoc}$, and $\psi \in \mathit{Loc}$. Whenever $\langle E, \xi, \psi \rangle \xrightarrow{\beta,\alpha,\sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle$ for some $\beta \in \mathit{BExp}$, $\alpha \in \mathit{Act}$, $\sigma \in \mathit{Sub}$, $E' \in \mathcal{G}_{\mathrm{u}}$, and $\xi' \in \mathit{CLoc}$, then $\mathit{unfold}(E') = E'$.*

Proof. We proceed by induction on the length l of the derivation of the symbolic transition in the hypothesis:

$$\langle \sum_{i \in I} \alpha_i . E_i, \xi, \psi \rangle \xrightarrow{true, \alpha_j, assign(E_j, \xi, \psi)}_{\mathrm{s}} \langle \mathit{unfold}(E_j), \mathit{loctree}(E_j, \xi, \psi), \psi \rangle$$

$$\frac{\langle Q_1, \xi, \psi \rangle \xrightarrow{\beta', \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}{\langle \text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \xi, \psi \rangle \xrightarrow{\beta \wedge \beta', \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}$$

$$\frac{\langle Q_2, \xi, \psi \rangle \xrightarrow{\beta', \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}{\langle \text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \xi, \psi \rangle \xrightarrow{\neg\beta \wedge \beta', \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}$$

$$\frac{\langle E, \xi, \psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}{\langle E \backslash L, \xi, \psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle E' \backslash L, \xi', \psi \rangle} \quad \mathit{name}(\alpha) \notin L$$

$$\frac{\langle E, \xi, \psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi \rangle}{\langle E[\varphi], \xi, \psi \rangle \xrightarrow{\beta, \varphi(\alpha), \sigma}_{\mathrm{s}} \langle E'[\varphi], \xi', \psi \rangle}$$

$$\frac{\langle E_1, \xi, \swarrow\psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle E'_1, \xi', \swarrow\psi \rangle}{\langle E_1 \mid E_2, \xi, \psi \rangle \xrightarrow{\swarrow\beta, \alpha, \swarrow\sigma}_{\mathrm{s}} \langle E'_1 \mid E_2, \xi', \psi \rangle}$$

$$\frac{\langle E_2, \xi, \searrow\psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle E'_2, \xi', \searrow\psi \rangle}{\langle E_1 \mid E_2, \xi, \psi \rangle \xrightarrow{\searrow\beta, \alpha, \searrow\sigma}_{\mathrm{s}} \langle E_1 \mid E'_2, \xi', \psi \rangle}$$

$$\frac{\langle E_1, \xi, \swarrow\psi \rangle \xrightarrow{\beta_1, a!(\underline{e}), \sigma_1}_{\mathrm{s}} \langle E'_1, \xi'_1, \swarrow\psi \rangle \quad \langle E_2, \xi, \searrow\psi \rangle \xrightarrow{\beta_2, a?(\underline{x}), \sigma_2}_{\mathrm{s}} \langle E'_2, \xi'_2, \searrow\psi \rangle}{\langle E_1 \mid E_2, \xi, \psi \rangle \xrightarrow{\swarrow\beta_1 \wedge \searrow\beta_2, \tau, \searrow\underline{x} := \swarrow\underline{e} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2}_{\mathrm{s}} \langle E'_1 \mid E'_2, \xi'_1 \cup \xi'_2, \psi \rangle}$$

$$\frac{\langle E_1, \xi, \swarrow\psi \rangle \xrightarrow{\beta_1, a?(\underline{x}), \sigma_1}_{\mathrm{s}} \langle E'_1, \xi'_1, \swarrow\psi \rangle \quad \langle E_2, \xi, \searrow\psi \rangle \xrightarrow{\beta_2, a!(\underline{e}), \sigma_2}_{\mathrm{s}} \langle E'_2, \xi'_2, \searrow\psi \rangle}{\langle E_1 \mid E_2, \xi, \psi \rangle \xrightarrow{\swarrow\beta_1 \wedge \searrow\beta_2, \tau, \swarrow\underline{x} := \searrow\underline{e} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2}_{\mathrm{s}} \langle E'_1 \mid E'_2, \xi'_1 \cup \xi'_2, \psi \rangle}$$

Table II. Operational symbolic semantics

—If $l = 1$, then the symbolic rule for the guarded alternative composition operator has been applied, hence trivially $\mathit{unfold}(E') = E'$ because E' is the result of an unfolding.

—Let $l > 1$. There are several cases based on the syntactical structure of E, which cannot be a constant invocation because it is the result of an unfolding:

—The case $E \equiv \sum_{i \in I} \alpha_i . E_i$ is not possible because $l > 1$.

—Let $E \equiv$ if (β) then Q_1 else Q_2. If the first symbolic rule for the conditional operator has been applied (the application of the second one can be treated similarly), then $\langle Q_1, \xi, \psi \rangle \xrightarrow{\beta', \alpha, \sigma}_{\rm s} \langle E', \xi', \psi \rangle$ whose derivation length is $l-1$. By the induction hypothesis it follows that $\mathit{unfold}(E') = E'$.

—Let $E \equiv F \backslash L$. Then the symbolic rule for the restriction operator has been applied, from which it follows that $\langle F, \xi, \psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\rm s} \langle F', \xi', \psi \rangle$ whose derivation length is $l-1$. By the induction hypothesis it follows that $\mathit{unfold}(F') = F'$, hence $\mathit{unfold}(E') = \mathit{unfold}(F' \backslash L) = \mathit{unfold}(F') \backslash L = F' \backslash L = E'$.

—The case $E \equiv F[\varphi]$ is similar to the previous one.

—Let $E \equiv E_1 \mid E_2$. There are four cases based on which symbolic rule for the parallel composition operator has been applied:

  —If the first symbolic rule has been applied (the application of the second one can be treated similarly), then $\langle E_1, \xi, \swarrow\psi \rangle \xrightarrow{\beta', \alpha, \sigma}_{\rm s} \langle E'_1, \xi', \swarrow\psi \rangle$ whose derivation length is $l-1$. By the induction hypothesis it follows that $\mathit{unfold}(E'_1) = E'_1$, hence $\mathit{unfold}(E') = \mathit{unfold}(E'_1 \mid E_2) = \mathit{unfold}(E'_1) \mid \mathit{unfold}(E_2) = E'_1 \mid E_2 = E'$, where we have exploited the fact that E is the result of an unfolding, hence so is E_2.

  —If the third symbolic rule has been applied (the application of the fourth one can be treated similarly), then $\langle E_1, \xi, \swarrow\psi \rangle \xrightarrow{\beta_1, a!(\underline{e}), \sigma_1}_{\rm s} \langle E'_1, \xi'_1, \swarrow\psi \rangle$ and $\langle E_2, \xi, \searrow\psi \rangle \xrightarrow{\beta_2, a?(\underline{x}), \sigma_2}_{\rm s} \langle E'_2, \xi'_2, \searrow\psi \rangle$ whose derivation lengths are at most $l-1$. By the induction hypothesis it follows that $\mathit{unfold}(E'_1) = E'_1$ and $\mathit{unfold}(E'_2) = E'_2$, hence $\mathit{unfold}(E') = \mathit{unfold}(E'_1 \mid E'_2) = \mathit{unfold}(E'_1) \mid \mathit{unfold}(E'_2) = E'_1 \mid E'_2 = E'$.

$\square$

Of course, a drawback of this general symbolic semantics is the complicated structure of the states, which are triples. A simpler but less general symbolic semantics can be obtained by employing a different technique, which introduces the localities in the terms in such a way that the assignments already contain the variables prefixed by the right localities. The observation is that the application of function *unfold* is needed only in the derivative term of the axiom for the guarded alternative composition operator, with the overall derivative term of the term under consideration being built in a bottom up way. Thus, function *unfold* itself may be used to introduce the localities in the derivative terms by simply modifying its clause for the parallel composition operator as follows

$$\mathit{unfold}(E_1 \mid E_2) = \mathit{unfold}(E_1\swarrow) \mid \mathit{unfold}(E_2\searrow)$$

where $E\swarrow$ (resp. $E\searrow$) is the term obtained from E by appending $\swarrow$ (resp. $\searrow$) at the end of the locality of each variable occurring in E. Since the terms would contain the variables prefixed by the right localities, the states would be represented by terms, with function *record* collapsing into function *assign* defined over terms and always generating technical assignments when encountering parallel composition operators within the scope of a constant defining equation. Additionally, the four symbolic rules for the parallel composition operator would not have to recover the right localities in the labels of the generated transitions. For all $E \in \mathcal{G}$, the STGLA $[\![E]\!]_{\rm s'}$ obtained with this alternative technique would be isomorphic to $[\![E]\!]_{\rm s}$. However it would not be the case that, whenever $E_1 \xrightarrow{\beta,\alpha,\sigma}_{\mathrm{s}'} E_2$, then $\mathit{unfold}(E_2) = E_2$. The reason is that function *unfold* would add localities to E_2 whereas they are already there. Therefore, the closure of function *unfold* under the application of the symbolic rules would be lost.

In principle, appropriate localities should be introduced to deal with all the binary operators, not only the parallel composition operator. In our case this is not necessary because of the syntax restrictions imposed in Def. 2.1 on the traditional choice operator and conditional operator. Let us see what happens if we drop such syntax restrictions.

The first technique considered above – symbolic rules introducing the localities only in the assignments – does not work for dynamic operators [3] like the choice operator and the conditional operator. Suppose that we use $\rfloor$ and $\lfloor$ as localities for the choice operator, to be taken into account during the application of functions *unfold* and *record* and of the symbolic rules. In addition, suppose that functions *unfold* and *record* are modified to deal with occurrences of the choice operator and the conditional operator in which there are operand terms that are not guarded by an action prefix (such operands must be unfolded). Now consider as an example

$$\begin{aligned} A(x;) &\triangleq a!(x).(A'(x+1) + A'(x+2)) \\ A'(z;) &\triangleq a'!(z).A'(z+1) \end{aligned}$$

After executing $a!(A.x)$ we get $\mathit{unfold}(A'(A.x{+}1){+}A'(A.x{+}2)) = a'!(A'.z).A'(A'.z+1) + a'!(A'.z).A'(A'.z+1)$ as derivative term with assignment $\rfloor A'.z := A.x + 1 \triangleright \lfloor A'.z := A.x + 2$. At this point, if the action of the left summand is executed, the new derivative term is $\mathit{unfold}(A'(A'.z{+}1)) = a'!(A'.z).A'(A'.z{+}1)$ with assignment $A'.z := A'.z + 1$ rather than $A'.z := \rfloor A'.z + 1$. Unfortunately, the connection with $\rfloor A'.z$ has been lost because the related choice operator has disappeared after the action execution. In order to make the first technique work, we should transparently transform the dynamic binary operators into static binary operators. For instance, in the case of the choice operator we may introduce a place holder *id* together with additional terms of the form $E + \mathit{id}$ and $\mathit{id} + E$ that would appear in the derivative term of the conclusion of the inference rules for the choice operator. Similarly, in the case of the conditional operator, we may introduce additional terms of the form $\text{if}\,(\mathit{id})\,\text{then}\,E\,\text{else}\,\mathit{id}$ and $\text{if}\,(\mathit{id})\,\text{then}\,\mathit{id}\,\text{else}\,E$. However, by so doing, objective (3) stated in the introduction would not be achieved. For instance, a state containing a term of the form $E + \mathit{id}$ would be different from a state containing a term of the form $\mathit{id} + E$.

The second technique considered above – symbolic rules introducing the localities in the terms in such a way that the assignments already contain the variables prefixed by the right localities – works in the case of the dynamic binary operators. However, its application would again hamper the achievement of objective (3), as one should expect due to the fact that the two techniques give rise to isomorphic STGLA. For instance, if we consider again the term $A(x;)$ defined in the last example, we see that after the execution of $a!(A.x)$ we move to $a'!(\rfloor A'.z).A'(\rfloor A'.z + 1) + a'!(\lfloor A'.z).A'(\lfloor A'.z + 1)$, from which two different states

[3]Operators no longer occurring in the derivative term of the conclusion of the related inference rules.

– $a'!(\rfloor A'.z).A'(\rfloor A'.z+1)$ and $a'!(\lfloor A'.z).A'(\lfloor A'.z+1)$ – can be reached instead of a single one – $a'!(A'.z).A'(A'.z+1)$ – as would be more intuitive.

A third possibility to deal with the dynamic binary operators in the absence of the syntax restrictions imposed in Def. 2.1 is – differently from the two previous techniques – not to unfold the operands in the occurrences of the choice operator and the conditional operator that are not guarded by an action prefix. However, this has two shortcomings. First, the resulting semantic model would be a hybrid between a STGA and a STGLA, with the transitions labeled with assignments before and after the actions. In fact, in this scenario the constant invocations occurring in an unguarded operand of a choice operator or a conditional operator have to be handled through an additional symbolic rule that expands them into the corresponding defining terms before inferring transitions, thus generating assignments that must be applied before the execution of the actions labeling the inferred transitions themselves. Second, once again the achievement of objective (3) would not be possible, because e.g. choice operators with operands not guarded by an action prefix like $A_1(\underline{e}_1) + A_2(\underline{e}_2)$ and $A_1(\underline{e}'_1) + A_2(\underline{e}'_2)$ would result in two different states.

3. CORRECTNESS OF THE SYMBOLIC SEMANTIC RULES

3.1 Consistency with the Concrete Semantic Rules

The proof of correctness of the STGLA based symbolic semantics proceeds in two steps. First, we establish that, for every term, its symbolic semantics and its concrete semantics are able to mimic each other. The reason is that, in principle, every concrete state can be related to the symbolic state obtainable from it by applying function *unfold*, because every application of function *unfold* to a symbolic derivative state corresponds to the application of the concrete rule for the constant invocation to a concrete source state. More precisely, in the consistency theorem below, we ideally consider a binary relation where each pair is composed of a set of concrete states and a symbolic state that share the same term E, and we prove that every pair in the relation satisfies a bisimulation like property. The reason why a set of concrete states, rather than a single concrete state, is related to a symbolic state is that, in the case of conditional branching, the symbolic state keeps both ways while a concrete state exposes only one way, which is determined by its evaluation (see the concrete and the symbolic rules for the conditional operator). To fully mimic the behavior of a symbolic state whose term is obtained by unfolding term E, we thus need to take into account all the concrete states whose term is E.

THEOREM 3.1. *For all $E \in \mathcal{G}$ the following holds:*

—*For all $\rho \in Eval$ such that $fv(E) \subseteq dom(\rho)$, whenever $\langle E, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho'\rangle$ for some $\alpha \in Act_{\mathrm{c}}$, $E' \in \mathcal{G}$, and $\rho' \in Eval$ with $fv(E') \subseteq dom(\rho')$, then, for all $\xi \in CLoc$ and $\psi \in Loc$, there exist $\beta' \in BExp$, $\alpha' \in Act$, $\sigma' \in Sub$, $\xi' \in CLoc$, $\underline{x'} \subseteq Var$, and $\underline{v'} \subseteq Val$ such that $\langle unfold(E), \xi, \psi\rangle \xrightarrow{\beta',\alpha',\sigma'}_{\mathrm{s}} \langle unfold(E'), \xi', \psi\rangle$ with:*

—$\alpha' = \begin{cases} a?(\underline{x}) & \text{if } \alpha = a?(\underline{v}) \\ a!(\underline{e}) & \text{if } \alpha = a!(\underline{v}) \wedge \rho\{\underline{x'} \mapsto \underline{v'}\}(\underline{e}) = \underline{v} \\ \alpha & \text{if } \alpha = \tau \end{cases}$ *;*

—$\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta') = true$;
—$\underline{x'}$ *and* $\underline{v'}$ *empty if and only if no application of the concrete rule for the constant invocation operator takes place during the derivation of* $\langle E, \rho \rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho' \rangle$.

—*For all* $\xi \in CLoc$ *and* $\psi \in Loc$, *whenever* $\langle unfold(E), \xi, \psi \rangle \xrightarrow{\beta,\alpha,\sigma}_{\mathrm{s}} \langle unfold(E'), \xi', \psi \rangle$ *for some* $\beta \in BExp$ *satisfiable,* $\alpha \in Act$, $\sigma \in Sub$, $E' \in \mathcal{G}$, *and* $\xi' \in CLoc$, *then there exist* $\rho \in Eval$ *with* $fv(E) \subseteq dom(\rho)$, $\alpha' \in Act_{\mathrm{c}}$, $\rho' \in Eval$ *with* $fv(E') \subseteq dom(\rho')$, $\underline{x'} \subseteq Var$, *and* $\underline{v'} \subseteq Val$ *such that* $\langle E, \rho \rangle \xrightarrow{\alpha'}_{\mathrm{c}} \langle E', \rho' \rangle$ *with:*
—$\alpha' = \begin{cases} a?(\underline{v}) & \text{if } \alpha = a?(\underline{x}) \\ a!(\underline{v}) & \text{if } \alpha = a!(\underline{e}) \wedge \rho\{\underline{x'} \mapsto \underline{v'}\}(\underline{e}) = \underline{v} \\ \alpha & \text{if } \alpha = \tau \end{cases}$;
—$\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta) = true$;
—$\underline{x'}$ *and* $\underline{v'}$ *empty if and only if no subterm of* $unfold(E)$ *resulting from the unfolding of a constant invocation contributes to the derivation of* $\langle unfold(E), \xi, \psi \rangle \xrightarrow{\beta,\alpha,\sigma}_{\mathrm{s}} \langle unfold(E'), \xi', \psi \rangle$.

Proof. We prove the first part of the theorem by proceeding by induction on the length l of the derivation of the concrete transition $\langle E, \rho \rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho' \rangle$ in the hypothesis. Let $\xi \in CLoc$ and $\psi \in Loc$ be arbitrarily chosen:

—If $l = 1$, then $E \equiv \sum_{i \in I} \alpha_i . E_i$ and one of the three concrete rules for the guarded alternative composition operator has been applied. There are three cases based on the syntactical structure of α:
—If $\alpha = a?(\underline{v})$ corresponds to $\alpha_j = a?(\underline{x})$ in E for some $j \in I$, then $E' \equiv E_j$ and $\rho' = \rho\{\underline{x} \mapsto \underline{v}\}$.
—If $\alpha = a!(\underline{v})$ corresponds to $\alpha_j = a!(\underline{e})$ in E for some $j \in I$, then $E' \equiv E_j$, $\rho' = \rho$, and $\rho(\underline{e}) = \underline{v}$.
—If $\alpha = \tau = \alpha_j$ for some $j \in I$, then $E' \equiv E_j$ and $\rho' = \rho$.
Since $unfold(E) = \sum_{i \in I} \alpha_i . E_i$ and the concrete rule for the constant invocation operator has not been applied, the result follows by applying the symbolic rule for the guarded alternative composition operator by selecting $\alpha' = \alpha_j$ for the same $j \in I$ as above, with $\underline{x'}$ and $\underline{v'}$ empty, $\beta' = true$ hence $\rho(\beta') = true$, $\sigma' = assign(E_j, \xi, \psi)$, and $\xi' = loctree(E_j, \xi, \psi)$.
—Let $l > 1$. There are several cases based on the syntactical structure of E:
—The case $E \equiv \sum_{i \in I} \alpha_i . E_i$ is not possible because $l > 1$.
—Let $E \equiv$ if (β) then Q_1 else Q_2. If the first concrete rule for the conditional operator has been applied (the application of the second one can be treated similarly), then $\langle Q_1, \rho \rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E', \rho' \rangle$, with $\rho(\beta) = true$, whose derivation length is $l - 1$. Due to the syntax restrictions on Q_1, the previous transition does not require the application of the concrete rule for the constant invocation operator, so we take $\underline{x'}$ and $\underline{v'}$ empty. By the induction hypothesis it follows in particular [4] that there exist $\beta', \alpha', \sigma', \xi'$ such that $\langle unfold(Q_1), \xi, \psi \rangle \xrightarrow{\beta',\alpha',\sigma'}_{\mathrm{s}} \langle unfold(E'), \xi', \psi \rangle$ with α' related to α as stated in

[4]By saying "in particular" we focus on a particular locality tree and a particular locality w.r.t. the universal quantification in the induction hypothesis.

the theorem and $\rho(\beta') = \mathit{true}$. Observed that $\mathit{unfold}(Q_1) = Q_1$, by applying the first simbolic rule for the conditional operator we get $\langle \text{if } (\beta) \text{ then } Q_1 \text{ else } Q_2, \xi, \psi\rangle \xrightarrow{\beta \wedge \beta', \alpha', \sigma'}_{\mathrm{s}} \langle \mathit{unfold}(E'), \xi', \psi\rangle$. The result follows from $\mathit{unfold}(E) = \text{if } (\beta)$ then Q_1 else Q_2 and $\rho(\beta \wedge \beta') = \rho(\beta) \wedge \rho(\beta') = \mathit{true}$.

—Let $E \equiv F \backslash L$. Then the concrete rule for the restriction operator has been applied, from which it follows that $\langle F, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle F', \rho'\rangle$, with $E' \equiv F' \backslash L$ and $\mathit{name}(\alpha) \notin L$, whose derivation length is $l - 1$. By the induction hypothesis it follows in particular that there exist $\beta', \alpha', \sigma', \xi', \underline{x'}, \underline{v'}$ such that $\langle \mathit{unfold}(F), \xi, \psi\rangle \xrightarrow{\beta', \alpha', \sigma'}_{\mathrm{s}} \langle \mathit{unfold}(F'), \xi', \psi\rangle$ with α' related to α as stated in the theorem and $\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta') = \mathit{true}$. By applying the symbolic rule for the restriction operator we get $\langle \mathit{unfold}(F)\backslash L, \xi, \psi\rangle \xrightarrow{\beta', \alpha', \sigma'}_{\mathrm{s}} \langle \mathit{unfold}(F')\backslash L, \xi', \psi\rangle$. The result follows from $\mathit{unfold}(F)\backslash L = \mathit{unfold}(F\backslash L)$ and $\mathit{unfold}(F')\backslash L = \mathit{unfold}(F'\backslash L)$.

—The case $E \equiv F[\varphi]$ is similar to the previous one.

—Let $E \equiv E_1 \mid E_2$. There are four cases based on which concrete rule for the parallel composition operator has been applied:

—If the first concrete rule has been applied (the application of the second one can be treated similarly), then $\langle E_1, \rho\rangle \xrightarrow{\alpha}_{\mathrm{c}} \langle E'_1, \rho'\rangle$, with $E' \equiv E'_1 \mid E_2$, whose derivation length is $l - 1$. By the induction hypothesis it follows in particular that there exist $\beta', \alpha', \sigma', \xi', \underline{x'}, \underline{v'}$ such that $\langle \mathit{unfold}(E_1), \xi, \swarrow\psi\rangle \xrightarrow{\beta', \alpha', \sigma'}_{\mathrm{s}} \langle \mathit{unfold}(E'_1), \xi', \swarrow\psi\rangle$ with α' related to α as stated in the theorem and $\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta') = \mathit{true}$. By applying the first symbolic rule for the parallel composition operator we get $\langle \mathit{unfold}(E_1) \mid \mathit{unfold}(E_2), \xi, \psi\rangle \xrightarrow{\swarrow\beta', \alpha', \swarrow\sigma'}_{\mathrm{s}} \langle \mathit{unfold}(E'_1) \mid \mathit{unfold}(E_2), \xi', \psi\rangle$. The result follows from $\mathit{unfold}(E_1) \mid \mathit{unfold}(E_2) = \mathit{unfold}(E_1 \mid E_2)$ and $\mathit{unfold}(E'_1) \mid \mathit{unfold}(E_2) = \mathit{unfold}(E'_1 \mid E_2)$ as well as from the fact that $\rho\{\swarrow\underline{x'} \mapsto \underline{v'}\}(\swarrow\beta') = \mathit{true}$, because every variable of E_1 occurring in β' is prefixed by $\swarrow$ when considered in the scope of $E_1 \mid E_2$, hence the truth value of $\swarrow\beta'$ is the same as the truth value of β'.

—If the third concrete rule has been applied (the application of the fourth one can be treated similarly), then $\alpha = \tau$ and $\langle E_1, \rho\rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E'_1, \rho'_1\rangle$ and $\langle E_2, \rho\rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E'_2, \rho'_2\rangle$, with $E' \equiv E'_1 \mid E'_2$ and $\rho' = \rho'_1 \cup \rho'_2$, whose derivation lengths are at most $l - 1$. By the induction hypothesis it follows in particular that there exist $\beta'_1, \beta'_2, \underline{e}, \underline{x}, \sigma'_1, \sigma'_2, \xi'_1, \xi'_2, \underline{x'}_1, \underline{x'}_2, \underline{v'}_1, \underline{v'}_2$ such that $\langle \mathit{unfold}(E_1), \xi, \swarrow\psi\rangle \xrightarrow{\beta'_1, a!(\underline{e}), \sigma'_1}_{\mathrm{s}} \langle \mathit{unfold}(E'_1), \xi'_1, \swarrow\psi\rangle$ and $\langle \mathit{unfold}(E_2), \xi, \searrow\psi\rangle \xrightarrow{\beta'_2, a?(\underline{x}), \sigma'_2}_{\mathrm{s}} \langle \mathit{unfold}(E'_2), \xi'_2, \searrow\psi\rangle$ with $\rho\{\underline{x'}_1 \mapsto \underline{v'}_1\}(\underline{e}) = \underline{v}$ and $\rho\{\underline{x'}_1 \mapsto \underline{v'}_1\}(\beta'_1) = \mathit{true} = \rho\{\underline{x'}_2 \mapsto \underline{v'}_2\}(\beta'_2)$. By applying the third symbolic rule for the parallel composition operator we get $\langle \mathit{unfold}(E_1) \mid \mathit{unfold}(E_2), \xi, \psi\rangle \xrightarrow{\swarrow\beta'_1 \wedge \searrow\beta'_2, \tau, \searrow\underline{x} := \swarrow\underline{e} \triangleright \swarrow\sigma'_1 \triangleright \searrow\sigma'_2}_{\mathrm{s}} \langle \mathit{unfold}(E'_1) \mid \mathit{unfold}(E'_2), \xi'_1 \cup \xi'_2, \psi\rangle$. The result follows from $\mathit{unfold}(E_1) \mid \mathit{unfold}(E_2) = \mathit{unfold}(E_1 \mid E_2)$ and $\mathit{unfold}(E'_1)$

$\mid \mathit{unfold}(E'_2) = \mathit{unfold}(E'_1 \mid E'_2)$ as well as from the fact that $\rho\{\swarrow\underline{x'}_1, \searrow\underline{x'}_2 \mapsto \underline{v'}_1, \underline{v'}_2\}(\swarrow\beta'_1 \wedge \searrow\beta'_2) = \mathit{true}$, because every variable of E_1 (resp. E_2) occurring in β'_1 (resp. β'_2) is prefixed by $\swarrow$ (resp. $\searrow$) when considered in the scope of $E_1 \mid E_2$, hence the truth value of $\swarrow\beta'_1 \wedge \searrow\beta'_2$ is the same as the truth value of $\beta'_1 \wedge \beta'_2$.

—Let $E \equiv A(\underline{e})$ with $A(\underline{x}; \underline{y}) \stackrel{\Delta}{=} F$. Then the concrete rule for the constant invocation operator has been applied, from which it follows that $\langle F, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha}_{\rm c} \langle E', \rho'\rangle$, with $\rho(\underline{e}) = \underline{v}$, whose derivation length is $l - 1$. By the induction hypothesis it follows in particular that there exist $\beta', \alpha', \sigma', \xi', \underline{x'}, \underline{v'}$ such that $\langle \mathit{unfold}(F), \xi, \psi\rangle \xrightarrow{\beta', \alpha', \sigma'}_{\rm s} \langle \mathit{unfold}(E'), \xi', \psi\rangle$ with α' related to α via $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}$ as stated in the theorem and $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}(\beta') = \mathit{true}$. The result follows from $\mathit{unfold}(A(\underline{e})) = \mathit{unfold}(F)$.

We prove the second part of the theorem by proceeding by induction on the length l of the derivation of the symbolic transition $\langle \mathit{unfold}(E), \xi, \psi\rangle \xrightarrow{\beta, \alpha, \sigma}_{\rm s} \langle \mathit{unfold}(E'), \xi', \psi\rangle$ in the hypothesis:

—If $l = 1$, then $\mathit{unfold}(E) = \sum_{i \in I} \alpha_i . E_i \equiv E$ and the symbolic rule for the guarded alternative composition operator has been applied with $\beta = \mathit{true}$. If $\alpha = \alpha_j$ for some $j \in I$, then $\mathit{unfold}(E') = \mathit{unfold}(E_j)$. Since none of the subterms of $\mathit{unfold}(E)$ results from the unfolding of a constant invocation, we take $\underline{x'}$ and $\underline{v'}$ empty. Given an arbitrary $\rho \in \mathit{Eval}$ such that $\mathit{fv}(E) \subseteq \mathit{dom}(\rho)$, $\rho(\beta)$ trivially holds and there are three cases based on the syntactical structure of α:

—If $\alpha = a?(\underline{x})$, the result follows by applying the first concrete rule for the guarded alternative composition operator with $\alpha' = a?(\underline{v})$ and $\rho' = \rho\{\underline{x} \mapsto \underline{v}\}$ for some $\underline{v} \subseteq \mathit{Val}$.

—If $\alpha = a!(\underline{e})$, the result follows by applying the second concrete rule for the guarded alternative composition operator with $\alpha' = a!(\rho(\underline{e}))$ and $\rho' = \rho$.

—If $\alpha = \tau$, then the result follows by applying the third concrete rule for the guarded alternative composition operator with $\alpha' = \tau$ and $\rho' = \rho$.

—Let $l > 1$. There are several cases based on the syntactical structure of E:

—The case $E \equiv \sum_{i \in I} \alpha_i . E_i$ is not possible because $l > 1$.

—Let $E \equiv$ if $(\hat{\beta})$ then Q_1 else Q_2, hence $\mathit{unfold}(E) = E$. If the first symbolic rule for the conditional operator has been applied (the application of the second one can be treated in similarly), then $\langle Q_1, \xi, \psi\rangle \xrightarrow{\beta', \alpha, \sigma}_{\rm s} \langle E', \xi', \psi\rangle$, with $\beta = \hat{\beta} \wedge \beta'$, whose derivation length is $l - 1$. Due to the syntax restrictions on Q_1, $\mathit{unfold}(Q_1) = Q_1$ and no subterm of $\mathit{unfold}(Q_1)$ results from the unfolding of a constant invocation, so we take $\underline{x'}$ and $\underline{v'}$ empty. In addition, $\mathit{unfold}(E') = E'$ because function unfold is closed under the application of the symbolic rules. By the induction hypothesis it follows that there exist $\bar{\rho}, \alpha', \bar{\rho}'$ such that $\langle Q_1, \bar{\rho}\rangle \xrightarrow{\alpha'}_{\rm c} \langle E', \bar{\rho}'\rangle$ with α' related to α as stated in the theorem and $\bar{\rho}(\beta') = \mathit{true}$. Since β is satisfiable, from $\bar{\rho}$ it is possible to obtain an evaluation ρ such that $\langle Q_1, \rho\rangle \xrightarrow{\alpha'}_{\rm c} \langle E', \rho'\rangle$ and $\rho(\beta) = \mathit{true}$. This implies that $\rho(\hat{\beta}) = \mathit{true}$.

The result then follows by applying the first concrete rule for the conditional operator.

—Let $E \equiv F \backslash L$, hence $\mathit{unfold}(E) = \mathit{unfold}(F) \backslash L$. Then the symbolic rule for the restriction operator has been applied, from which it follows that $\langle \mathit{unfold}(F), \xi, \psi \rangle \xrightarrow{\beta', \alpha', \sigma'}_{\mathrm{s}} \langle F', \xi', \psi \rangle$, with $E' \equiv F' \backslash L$ and $\mathit{name}(\alpha) \notin L$, whose derivation length is $l - 1$. $\mathit{unfold}(F') = F'$ because function *unfold* is closed under the application of the symbolic rules. By the induction hypothesis it follows that there exist $\rho, \alpha', \rho', \underline{x'}, \underline{v'}$ such that $\langle F, \rho \rangle \xrightarrow{\alpha'}_{\mathrm{c}} \langle F', \rho' \rangle$ with α' related to α as stated in the theorem and $\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta) = \mathit{true}$. From $\mathit{name}(\alpha) \notin L$ it follows that $\mathit{name}(\alpha') \notin L$. The result then follows by applying the concrete rule for the restriction operator.

—The case $E \equiv F[\varphi]$ is similar to the previous one.

—Let $E \equiv E_1 \mid E_2$, hence $\mathit{unfold}(E) = \mathit{unfold}(E_1) \mid \mathit{unfold}(E_2)$. There are four cases based on which symbolic rule for the parallel composition operator has been applied:

—If the first symbolic rule has been applied (the application of the second one can be treated similarly), then $\langle \mathit{unfold}(E_1), \xi, \swarrow\psi \rangle \xrightarrow{\beta', \alpha', \sigma'}_{\mathrm{s}} \langle E'_1, \xi', \swarrow\psi \rangle$, with $E' \equiv E'_1 \mid E_2$, $\beta = \swarrow\beta'$, and $\sigma = \swarrow\sigma'$, whose derivation length is $l-1$. $\mathit{unfold}(E'_1) = E'_1$ because function *unfold* is closed under the application of the symbolic rules. By the induction hypothesis it follows that there exist $\bar{\rho}, \alpha', \bar{\rho}', \underline{x'}, \underline{v'}$ such that $\langle E_1, \bar{\rho} \rangle \xrightarrow{\alpha'}_{\mathrm{c}} \langle E'_1, \bar{\rho}' \rangle$ with α' related to α as stated in the theorem and $\bar{\rho}\{\underline{x'} \mapsto \underline{v'}\}(\beta') = \mathit{true}$. If we take $\rho = \bar{\rho}\{\swarrow\underline{x'} \mapsto \underline{v'}\}$ and $\rho' = \bar{\rho}'\{\swarrow\underline{x'} \mapsto \underline{v'}\}$, then $\langle E_1, \rho \rangle \xrightarrow{\alpha'}_{\mathrm{c}} \langle E'_1, \rho' \rangle$ and $\rho\{\underline{x'} \mapsto \underline{v'}\}(\beta) = \mathit{true}$. The result then follows by applying the first concrete rule for the parallel composition operator.

—If the third symbolic rule has been applied (the application of the fourth one can be treated similarly), then $\alpha = \tau$ and $\langle \mathit{unfold}(E_1), \xi, \swarrow\psi \rangle \xrightarrow{\beta'_1, a!(\underline{e}), \sigma'_1}_{\mathrm{s}} \langle E'_1, \xi'_1, \swarrow\psi \rangle$ and $\langle \mathit{unfold}(E_2), \xi, \searrow\psi \rangle \xrightarrow{\beta'_2, a?(\underline{x}), \sigma'_2}_{\mathrm{s}} \langle E'_2, \xi'_2, \searrow\psi \rangle$, with $E' \equiv E'_1 \mid E'_2$, $\beta = \swarrow\beta_1 \wedge \searrow\beta_2$, and $\sigma = \searrow\underline{x} := \swarrow\underline{e} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2$, whose derivation lengths are at most $l-1$. $\mathit{unfold}(E'_1) = E'_1$ and $\mathit{unfold}(E'_2) = E'_2$ because function *unfold* is closed under the application of the symbolic rules. By the induction hypothesis it follows that there exist $\bar{\rho}, \bar{\rho}'_1, \bar{\rho}'_2, \underline{x'}_1, \underline{x'}_2, \underline{v'}_1, \underline{v'}_2$ such that $\langle E_1, \bar{\rho} \rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E'_1, \bar{\rho}'_1 \rangle$ and $\langle E_2, \bar{\rho} \rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E'_2, \bar{\rho}'_2 \rangle$, with $\bar{\rho}(\underline{e}) = \underline{v}$ and $\bar{\rho}\{\underline{x'}_1 \mapsto \underline{v'}_1\}(\beta'_1) = \mathit{true} = \bar{\rho}\{\underline{x'}_2 \mapsto \underline{v'}_2\}(\beta'_2)$. If we take $\rho = \bar{\rho}\{\swarrow\underline{x'}_1, \searrow\underline{x'}_2 \mapsto \underline{v'}_1, \underline{v'}_2\}$, $\rho'_1 = \bar{\rho}'_1\{\swarrow\underline{x'}_1 \mapsto \underline{v'}_1\}$, and $\rho'_2 = \bar{\rho}'_2\{\searrow\underline{x'}_2 \mapsto \underline{v'}_2\}$, then $\langle E_1, \rho \rangle \xrightarrow{a!(\underline{v})}_{\mathrm{c}} \langle E'_1, \rho'_1 \rangle$ and $\langle E_2, \rho \rangle \xrightarrow{a?(\underline{v})}_{\mathrm{c}} \langle E'_2, \rho'_2 \rangle$. The result then follows by applying the third concrete rule for the parallel composition operator.

—Let $E \equiv A(\underline{e})$ with $A(\underline{x}; \underline{y}) \triangleq F$. Then the symbolic transition in the hypothesis becomes $\langle \mathit{unfold}(F), \xi, \psi \rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle \mathit{unfold}(E'), \xi', \psi \rangle$. We now proceed by induction on the length n of the derivation of $\mathit{unfold}(F)$ through the application of function *unfold* to F:

—If $n = 1$, then $\mathit{unfold}(F) = F$, hence F is not a constant invocation. Therefore we can exploit the previous fragment of the second part of the proof to deduce that there exist $\rho, \alpha', \rho', \underline{x'}, \underline{v'}$ such that $\langle F, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle E', \rho'\rangle$ with $\rho(\underline{e}) = \underline{v}$, α' related to α via $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}$ as stated in the theorem, and $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}(\beta) = \mathit{true}$. The result then follows by applying the concrete rule for the constant invocation operator.

—Let $n > 1$. There are several cases based on the syntactical structure of F:

—The case in which the outermost operator of F is a guarded alternative composition operator or a conditional operator is not possible because $n > 1$.

—If $F \equiv G\backslash L$, then $\mathit{unfold}(F) = \mathit{unfold}(G)\backslash L$. Since the symbolic rule for the restriction operator has been applied to derive the symbolic transition in the hypothesis, we get $\langle \mathit{unfold}(G), \xi, \psi\rangle \xrightarrow{\beta,\alpha,\sigma}_{\rm s} \langle \mathit{unfold}(G'), \xi', \psi\rangle$, with $E' \equiv G'\backslash L$ and $\mathit{name}(\alpha) \notin L$. By the induction hypothesis it follows that there exist $\rho, \alpha', \rho', \underline{x'}, \underline{v'}$ such that $\langle G, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle G', \rho'\rangle$ with $\rho(\underline{e}) = \underline{v}$, α' related to α via $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}$ as stated in the theorem, and $\rho\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}(\beta') = \mathit{true}$. Since $\mathit{name}(\alpha') \notin L$, by applying the concrete rule for the restriction operator we get $\langle G\backslash L, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle G'\backslash L, \rho'\rangle$. The result then follows by applying the concrete rule for the constant invocation operator.

—The case $F \equiv G[\varphi]$ is similar to the previous one.

—If $F \equiv F_1 \mid F_2$, then $\mathit{unfold}(F) = \mathit{unfold}(F_1) \mid \mathit{unfold}(F_2)$. There are four cases based on which symbolic rule for the parallel composition operator has been applied to derive the symbolic transition in the hypothesis:
If the first symbolic rule has been applied (the application of the second one can be treated similarly), then $\langle \mathit{unfold}(F_1), \xi, \swarrow\psi\rangle \xrightarrow{\beta',\alpha,\sigma'}_{\rm s} \langle \mathit{unfold}(F_1'), \xi', \swarrow\psi\rangle$, with $E' \equiv F_1' \mid F_2$, $\beta = \swarrow\beta'$, and $\sigma = \swarrow\sigma'$. By the induction hypothesis it follows that there exist $\bar{\rho}, \alpha', \bar{\rho}', \underline{x'}, \underline{v'}$ such that $\langle F_1, \bar{\rho}\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle F_1', \bar{\rho}'\rangle$, with $\rho(\underline{e}) = \underline{v}$, α' related to α via $\bar{\rho}\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}$ as stated in the theorem, and $\bar{\rho}\{\underline{x}, \underline{x'} \mapsto \underline{v}, \underline{v'}\}(\beta') = \mathit{true}$. If we take $\rho = \bar{\rho}\{\swarrow\underline{x'} \mapsto \underline{v'}\}$ and $\rho' = \bar{\rho}'\{\swarrow\underline{x'} \mapsto \underline{v'}\}$, then $\langle F_1, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle F_1', \rho'\rangle$ and $\rho\{\underline{x} \mapsto \underline{v}\}(\beta) = \mathit{true}$. By applying the first concrete rule for the parallel composition operator we get $\langle F_1 \mid F_2, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle F_1' \mid F_2, \rho'\rangle$. The result then follows by applying the concrete rule for the constant invocation operator.
If the third symbolic rule has been applied (the application of the fourth one can be treated similarly), then $\langle \mathit{unfold}(F_1), \xi, \swarrow\psi\rangle \xrightarrow{\beta_1', a!(\underline{e}_1), \sigma_1}_{\rm s} \langle \mathit{unfold}(F_1'), \xi', \swarrow\psi\rangle$ and $\langle \mathit{unfold}(F_2), \xi, \searrow\psi\rangle \xrightarrow{\beta_2', a!(\underline{x}_2), \sigma_2}_{\rm s} \langle \mathit{unfold}(F_2'), \xi', \searrow\psi\rangle$, with $E' \equiv F_1' \mid F_2'$, $\beta = \swarrow\beta_1' \wedge \searrow\beta_2'$, $\alpha = \tau$, and $\sigma = \searrow\underline{x}_2 := \swarrow\underline{e}_1 \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2$. By the induction hypothesis it follows that there ex-

ist $\bar{\rho}, \bar{\rho}'_1, \bar{\rho}'_2, \underline{x'}_1, \underline{x'}_2, \underline{v'}_1, \underline{v'}_2$ such that $\langle F_1, \bar{\rho}\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{a!(\underline{v''})}_{\rm c} \langle F'_1, \bar{\rho}'_1\rangle$ and $\langle F_2, \bar{\rho}\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{a?(\underline{v''})}_{\rm c} \langle F'_2, \bar{\rho}'_2\rangle$, with $\bar{\rho}(\underline{e}) = \underline{v}$ and $\bar{\rho}\{\underline{x}, \underline{x'}_1 \mapsto \underline{v}, \underline{v'}_1\}(\beta'_1) = \mathit{true} = \bar{\rho}\{\underline{x}, \underline{x'}_2 \mapsto \underline{v}, \underline{v'}_2\}(\beta'_2)$. If we take $\rho = \bar{\rho}\{\swarrow\underline{x'}_1, \searrow\underline{x'}_2 \mapsto \underline{v'}_1, \underline{v'}_2\}$, $\rho'_1 = \bar{\rho}'_1\{\swarrow\underline{x'}_1 \mapsto \underline{v'}_1\}$, and $\rho'_2 = \bar{\rho}'_2\{\searrow\underline{x'}_2 \mapsto \underline{v'}_2\}$, then $\langle F_1, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{a!(\underline{v''})}_{\rm c} \langle F'_1, \rho'_1\rangle$, $\langle F_2, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{a?(\underline{v''})}_{\rm c} \langle F'_2, \rho'_2\rangle$, and $\rho\{\underline{x} \mapsto \underline{v}\}(\beta) = \mathit{true}$. By applying the third concrete rule for the parallel composition operator we get $\langle F_1 \mid F_2, \rho\{\underline{x} \mapsto \underline{v}\}\rangle \xrightarrow{\tau}_{\rm c} \langle F'_1 \mid F'_2, \rho'_1 \cup \rho'_2\rangle$. The result then follows by applying the concrete rule for the constant invocation operator.

—Let $F \equiv A''(\underline{e''})$ with $A''(\underline{x''}; \underline{y''}) \equiv G$. Then $\mathit{unfold}(F) = \mathit{unfold}(G)$. By the induction hypothesis it follows that there exist $\rho, \alpha', \rho', \underline{x'}, \underline{v'}$ such that $\langle G, \rho\{\underline{x}, \underline{x''} \mapsto \underline{v}, \underline{v''}\}\rangle \xrightarrow{\alpha'}_{\rm c} \langle E', \rho'\rangle$, with $\rho(\underline{e}) = \underline{v}$, $\rho\{\underline{x} \mapsto \underline{v}\}(\underline{e''}) = \underline{v''}$, α' related to α via $\rho\{\underline{x}, \underline{x''}, \underline{x'} \mapsto \underline{v}, \underline{v''}, \underline{v'}\}$ as stated in the theorem, and $\rho\{\underline{x}, \underline{x''}, \underline{x'} \mapsto \underline{v}, \underline{v''}, \underline{v'}\}(\beta) = \mathit{true}$. The result then follows by applying the concrete rule for the constant invocation operator twice.

□

3.2 Assignment Application Order

Recalled that *Sub* is defined as a set of sequences of elements of *Var* × *Expr* rather than a set of functions from *Var* to *Expr*, the second step in the proof of correctness of the STGLA based symbolic semantics is about establishing that the order in which the assignments occur in the symbolic transitions is correct. This is important e.g. in the case of simulation based analysis of a value passing process description, because simulation requires the symbolic transitions to be actually executed, hence the related assignments to be actually applied.

Since the symbolic rules are always applied to terms obtained from the application of function *unfold*, the assignments stemming from synchronizations must be applied before the assignments generated by the unfolding of the corresponding derivative terms. As can be seen from the two symbolic rules for the synchronization, this is the order in which such assignments are generated. As an example, consider

$$\begin{aligned} A(x; y) &\triangleq (a!(x).A'(x+1) \mid a?(y).A'(y+2))\backslash\{a\} \\ A'(z) &\triangleq a'!(z).A'(z+1) \end{aligned}$$

The symbolic transition originated by the synchronization of $a!(\swarrow A.x)$ with $a?(\searrow A.y)$ is labeled with assignment $\searrow A.y := \swarrow A.x \triangleright \swarrow A'.z := \swarrow A.x + 1 \triangleright \searrow A'.z := \searrow A.y + 2$. Assignment $\searrow A.y := \swarrow A.x$ must be evaluated before the other two, otherwise $\searrow A'.z$ would get a wrong value.

The assignments generated during the unfolding process must be applied in the same order as they are produced by function *assign*. In particular, in the clause for the parallel composition operator, the technical assignments must be applied before the other assignments. Similarly, in the clause for the constant invocation operator, the parameter setting assignments must be applied before the other assignments. As an example, consider

$$\begin{aligned}
A(x;\,) &\triangleq a!(x).(A_1(x+1) \mid A_2(x+2)) \\
A_1(y;\,) &\triangleq a_1!(y).A_1(y+3) \\
A_2(z;\,) &\triangleq A_3(z+4) \\
A_3(z;\,) &\triangleq a_3!(z).A_3(z+5)
\end{aligned}$$

The symbolic transition originated by the execution of $a!(A.x)$ is labeled with assignment $\swarrow A.x := \searrow A.x := A.x \triangleright \swarrow A_1.y := \swarrow A.x + 1 \triangleright \searrow A_2.z := \searrow A.x + 2 \triangleright \searrow A_3.z := \searrow A_2.z + 4$, which has to be applied exactly in this order to avoid errors.

Finally, let us denote by *prefix*(x) the string in the concatenation of *Loc* and *Const* that precedes the actual name of variable x. We observe that for the parameter setting assignments that refer to variables with the same prefix, i.e. declared in the same constant invoked in the same locality, it is not possible to fix an order that allows them to be treated individually. To see why, consider

$$A(x, y) \triangleq a!(x, y).A(y + 1, x + 1)$$

The symbolic transition originated by the execution of $a!(A.x, A.y)$ is labeled with assignment $A.x := A.y{+}1 \triangleright A.y := A.x{+}1$. If $A.x$ is immediately set after evaluating $A.y + 1$, then $A.y$ subsequently gets a wrong value (the value of $A.x$ before the action execution incremented by 2); a similar argument applies if we start from the second element of the assignment. This kind of assignments – which can easily be recognized during the application of function *assign* – must be simultaneously treated in two phases. In the first phase, all of their right hand sides must be evaluated. Then, in the second phase, all of their left hand sides must be set.

We can thus conclude that the assignment generation order imposed by function *assign* and the symbolic rules is correct w.r.t. the assignment application order in the sense below.

THEOREM 3.2. *Let $E \in \mathcal{G}_{\rm u}$, $\xi \in$ CLoc, and $\psi \in$ Loc. Whenever $\langle E, \xi, \psi\rangle \xrightarrow{\beta,\alpha,\sigma}_{\rm s} \langle E', \xi', \psi\rangle$ for some $\beta \in$ BExp, $\alpha \in$ Act, $\sigma \in$ Sub, $E' \in \mathcal{G}_{\rm u}$, and $\xi' \in$ CLoc, then for all $x \in$ Var set by $x := e$ in σ, the assignment $x := e$ is unique in σ and there is no assignment $x' := e'$ preceding $x := e$ in σ with prefix$(x') \neq$ prefix(x) such that x occurs in e'.*

PROOF. We proceed by induction on the length l of the derivation of the symbolic transition in the hypothesis:

—If $l = 1$, then $E \equiv \sum_{i\in I} \alpha_i.E_i$ and the symbolic rule for the guarded alternative composition operator has been applied, with $\beta = true$, $\alpha = \alpha_j$, $\sigma = assign(E_j, \xi, \psi)$, and $E' = unfold(E_j)$ for some $j \in I$. We now proceed by induction on the length n of the derivation of σ through the application of function *assign* to E_j:

—If $n = 1$, then the outermost operator of E_j is either a guarded alternative composition operator or a conditional operator. In both cases, $\sigma = \varepsilon$ and the result trivially follows.

—Let $n > 1$. There are several cases based on the syntactical structure of E_j:

—The case in which the outermost operator of E_j is a guarded alternative composition operator or a conditional operator is not possible because $n > 1$.

—If $E_j \equiv F\backslash L$, then $\mathit{assign}(E_j, \xi, \psi) = \mathit{assign}(F, \xi, \psi)$. By the induction hypothesis it follows that $\mathit{assign}(F, \xi, \psi)$ satisfies the property of interest, hence so does $\mathit{assign}(E_j, \xi, \psi)$.

—The case $E_j \equiv F[\varphi]$ is similar to the previous one.

—If $E_j \equiv E_1 \mid E_2$, then $\mathit{assign}(E_j, \xi, \psi) = \sigma_{\mathrm{tech},1,2} \triangleright \swarrow\mathit{assign}(E_1, \xi\{\swarrow\psi \mapsto \xi(\psi)\}, \swarrow\psi) \triangleright \searrow\mathit{assign}(E_2, \xi\{\searrow\psi \mapsto \xi(\psi)\}, \searrow\psi)$. By the induction hypothesis, both $\mathit{assign}(E_1, \xi\{\swarrow\psi \mapsto \xi(\psi)\}, \swarrow\psi)$ and $\mathit{assign}(E_2, \xi\{\searrow\psi \mapsto \xi(\psi)\}, \searrow\psi)$ satisfy the property of interest, hence so does $\swarrow\mathit{assign}(E_1, \xi\{\swarrow\psi \mapsto \xi(\psi)\}, \swarrow\psi) \triangleright \searrow\mathit{assign}(E_2, \xi\{\searrow\psi \mapsto \xi(\psi)\}, \searrow\psi)$ because of the two different initial localities. The property is still preserved when including $\sigma_{\mathrm{tech},1,2}$ at the beginning because, in the case that $\sigma_{\mathrm{tech},1,2} \neq \varepsilon$, the left hand side of a technical assignment cannot be reassigned in an assignment of $\swarrow\mathit{assign}(E_1, \xi\{\swarrow\psi \mapsto \xi(\psi)\}, \swarrow\psi) \triangleright \searrow\mathit{assign}(E_2, \xi\{\searrow\psi \mapsto \xi(\psi)\}, \searrow\psi)$, due to the syntax restrictions in Def. 2.1 discussed in Sect. 2.3.3, and the right hand side of a technical assignment cannot occur in the left hand side of an assignment of $\swarrow\mathit{assign}(E_1, \xi\{\swarrow\psi \mapsto \xi(\psi)\}, \swarrow\psi) \triangleright \searrow\mathit{assign}(E_2, \xi\{\searrow\psi \mapsto \xi(\psi)\}, \searrow\psi)$, as it has a shorter locality string.

—If $E_j \equiv A(\underline{e})$ with $A(\underline{x}; \underline{y}) \triangleq F$, then $\mathit{assign}(E_j, \xi, \psi) = \underline{x} := \underline{e} \triangleright \mathit{assign}(F, \xi\{\psi \mapsto A\}, \psi)$. By the induction hypothesis it follows that $\mathit{assign}(F, \xi\{\psi \mapsto A\}, \psi)$ satisfies the property of interest, hence so does $\mathit{assign}(E_j, \xi, \psi)$ because $\underline{x}$ cannot be reassigned in an assignment of $\mathit{assign}(F, \xi\{\psi \mapsto A\}, \psi)$ and the possible variables of $\underline{e}$ cannot occur in the left hand side of an assignment of $\mathit{assign}(F, \xi\{\psi \mapsto A\}, \psi)$, due to the syntax restrictions in Def. 2.1 discussed in Sect. 2.3.3.

—Let $l > 1$. There are several cases based on the syntactical structure of E:

—The case $E \equiv \sum_{i \in I} \alpha_i.E_i$ is not possible because $l > 1$.

—Let $E \equiv$ if (β') then Q_1 else Q_2. If the first symbolic rule for the conditional operator has been applied (the application of the second one can be treated similarly), then $\langle Q_1, \xi, \psi\rangle \xrightarrow{\beta'', \alpha, \sigma}_{\mathrm{s}} \langle E', \xi', \psi\rangle$ whose derivation length is $l-1$. By the induction hypothesis it follows that σ satisfies the property of interest.

—Let $E \equiv F\backslash L$. Then the symbolic rule for the restriction operator has been applied, from which it follows that $\langle F, \xi, \psi\rangle \xrightarrow{\beta, \alpha, \sigma}_{\mathrm{s}} \langle F', \xi', \psi\rangle$ whose derivation length is $l - 1$. By the induction hypothesis it follows that σ satisfies the property of interest.

—The case $E \equiv F[\varphi]$ is similar to the previous one.

—Let $E \equiv E_1 \mid E_2$. There are four cases based on which symbolic rule for the parallel composition operator has been applied:

—If the first symbolic rule has been applied (the application of the second one can be treated similarly), then $\langle E_1, \xi, \swarrow\psi\rangle \xrightarrow{\beta', \alpha, \sigma'}_{\mathrm{s}} \langle E'_1, \xi'_1, \swarrow\psi\rangle$ whose derivation length is $l - 1$. By the induction hypothesis it follows that σ' satisfies the property of interest, hence so does $\sigma = \swarrow\sigma'$.

—If the third symbolic rule has been applied (the application of the fourth one can be treated similarly), then $\langle E_1, \xi, \swarrow\psi\rangle \xrightarrow{\beta_1, a!(\underline{e}), \sigma_1}_{\mathrm{s}} \langle E'_1, \xi'_1, \swarrow\psi\rangle$ and

$\langle E_2, \xi, \searrow\psi\rangle \xrightarrow{\beta_2, a?(\underline{x}), \sigma_2}_{\rm s} \langle E'_2, \xi'_2, \searrow\psi\rangle$ whose derivation lengths are at most $l-1$. By the induction hypothesis it follows that both σ_1 and σ_2 satisfy the property of interest, hence so does $\swarrow\sigma_1 \triangleright \searrow\sigma_2$ because of the two different initial localities. The property is preserved by $\sigma = \searrow\underline{x} := \swarrow\underline{e} \triangleright \swarrow\sigma_1 \triangleright \searrow\sigma_2$ because $\searrow\underline{x}$ cannot be reassigned in an assignment of σ_2 and the possible variables of $\swarrow\underline{e}$ cannot occur in the left hand side of an assignment of σ_1, due to the syntax restrictions in Def. 2.1 discussed in Sect. 2.3.3.

□

4. COMPACTNESS OF THE GENERATED STGLA

In this section we investigate the compactness of the STGLA produced by the symbolic rules, i.e. the achievement of objective (3) stated in the introduction. Since we generate symbolic models out of value passing process descriptions, it is useful for verification purposes to benefit as much as possible from the inherent parametricity of value passing terms. Such a parametricity is especially evident when we consider the constant defining equations, as each of them is provided only once for a given set of formal parameters, whereas there can be several constant invocations for different sets of actual parameters.

A compactness criterion should thus require that the constant invocations be somehow avoided as much as possible within the symbolic states. This can be achieved in two different ways:

—The first technique consists of rewriting every constant invocation occurring in a value passing specification into the right hand side of the corresponding constant defining equation, by somehow keeping track of the relationship between actual parameters and formal parameters. Since this process may not terminate, we restrict ourselves to value passing process algebras that – like $\mathcal{G}$ – require their terms to be guarded and we consequently rewrite only those constant invocations occurring outside the scope of an action prefix operator. This has the advantage of rewriting every constant invocation just before the invocation takes place, so there is no need to store information about the relationship between actual parameters and formal parameters far in advance. We call r-compactness, where 'r' stands for rewriting, the compactness criterion according to which no constant invocation occurs in a symbolic state outside the scope of an action prefix operator.

—The second technique consists of syntactically substituting formal parameters for actual parameters within every constant invocation occurring in a value passing specification, by somehow keeping track of the relationship between actual parameters and formal parameters. This abstraction step can be made independently of the guardedness of the terms of the considered value passing process algebra. We call a-compactness, where 'a' stands for abstraction, the compactness criterion according to which no constant invocation occurs in a symbolic state with actual parameters.

The two compactness criteria above are essentially incomparable. As an example, if we consider

$$A(x;) \stackrel{\Delta}{=} a!(x).b!(x).A(1) + c!(x).b!(x).A(2)$$

the underlying symbolic model has three states in the case the r-compactness based technique is applied, while it has only two states in the case the a-compactness based technique is applied, as in this case $A(1)$ and $A(2)$ are immediately replaced with $A(x)$ thus causing $b!(x).A(1)$ and $b!(x).A(2)$ to collapse into $b!(x).A(x)$. On the other hand, if we consider

$$\begin{aligned} B(x;\,) &\stackrel{\Delta}{=} B_1(x+1) \mid B_2(x-1) \\ B_1(x;\,) &\stackrel{\Delta}{=} b_1!(x).B_1(x+1) \\ B_2(x;\,) &\stackrel{\Delta}{=} b_2!(x).B_2(x-1) \end{aligned}$$

a single symbolic state $b_1!(x).B_1(x{+}1) \mid b_2!(x).B_2(x{-}1)$ is generated in the case the r-compactness based technique is applied, while two different symbolic states $B(x)$ and $B_1(x) \mid B_2(x)$ are generated in the case the a-compactness based technique is applied.

In this section we consider r-compactness for two reasons. The first reason is that a-compactness is more difficult to implement, as the fact that assignments may be computed far in advance forces to associate with them some information about the instant in which they become effective. This time related information is not necessary in the case of the r-compactness, as in this case the assignments are computed just before they are needed. The second reason is that r-compactness is more convenient in the case of terms with no value passing because it avoids the generation of unnecessary states due to constant aliasing, as shown by the last example above after removing the constant parameters, whereas this is not the case for a-compactness.

The important property of the STGLA based symbolic rules is that they are optimal w.r.t. r-compactness. This property cannot be achieved when using STGs or STGA or when applying the STGA' based symbolic rules proposed in [Li and Chen 1999] for the considered fragment of value passing CCS.

Theorem 4.1. *For all $E \in \mathcal{G}$, every state of $[\![E]\!]_{\rm s}$ is r-compact.*

Proof. It follows from the guardedness of E, the use of function *unfold* in the symbolic rule for the guarded alternative composition operator, and the definition of function *unfold* itself. $\square$

We now illustrate by means of two examples the advantages gained thanks to the r-compactness of the generated STGLA.

Example 4.2. The first example is about a comparison with the concrete semantics. Here the observation is that, if we consider a value passing specification E given by the parallel composition of n constant invocations having $m_1, \dots, m_n$ independent parameters respectively, where formal parameter $j = 1, \dots, m_i$ of constant $i = 1, \dots, n$ can assume $q_{i,j}$ different values, then $[\![E]\!]_{\rm s}$ can be up to $\prod_{1 \le i \le n, 1 \le j \le m_i} q_{i,j}$ times smaller than $[\![E]\!]_{\rm c}$. From an application viewpoint, this means that it may be convenient to model with value passing terms those systems that would not necessarily need it.

Let us consider the case of the alternating bit protocol [Bartlett et al. 1969]. It is a data link level communication protocol that establishes a means whereby two stations, one acting as a sender and the other acting as a receiver, connected by a full duplex, FIFO communication channel that may lose messages, can cope with message loss. The name of the protocol stems from the fact that each message is augmented with an additional bit. Since consecutive messages that are not lost are tagged with additional bits that are pairwise complementary, it is easy to distinguish between an original message and its possible duplicates. Initially, if the data link level of the sender obtains a message from the upper level, it augments the message with an additional bit set to 0, sends the tagged message to the receiver, and starts a timer. If an acknowledgement tagged with 0 is received before the timeout expires, then the subsequent message obtained from the upper level will be sent with an additional bit set to 1, otherwise the current tagged message is sent again. On the other side, the data link level of the receiver waits for a message tagged with 0. If it receives such a tagged message for the first time, then it passes the message to the upper level, sends an acknowledgement tagged with 0 back to the sender, and waits for a message tagged with 1, whereas if it receives a duplicate tagged message (due to message loss, acknowledgement loss, or propagation taking an exceedingly long time), then it sends an acknowledgement tagged with the same additional bit back to the sender. Abstracting from the contents of the messages that are actually exchanged, the protocol can be modeled as follows in the considered fragment of value passing CCS:

$$
\begin{array}{l}
\mathit{ABP}(\mathit{tagging\_bit};) \triangleq \\
\qquad (\mathit{Sender}(\mathit{tagging\_bit}) \mid \mathit{Channel}() \mid \mathit{Receiver}(\mathit{tagging\_bit})) \\
\qquad\qquad \setminus \{\mathit{transmit\_msg}, \mathit{deliver\_msg}, \mathit{transmit\_ack}, \mathit{deliver\_ack}\} \\
\mathit{Sender}(\mathit{sent\_bit};) \triangleq \\
\qquad \mathit{generate\_msg}?().\mathit{transmit\_msg}!(\mathit{sent\_bit}).\mathit{Sender}'(\mathit{sent\_bit}) \\
\mathit{Sender}'(\mathit{sent\_bit}; \mathit{received\_bit}) \triangleq \\
\qquad \mathit{deliver\_ack}?(\mathit{received\_bit}).\mathit{Sender}''(\mathit{sent\_bit}, \mathit{received\_bit}) + \\
\qquad \mathit{timeout}!().\mathit{Sender}'''(\mathit{sent\_bit}) \\
\mathit{Sender}''(\mathit{sent\_bit}, \mathit{received\_bit};) \triangleq \\
\qquad \text{if } (\mathit{received\_bit} = \mathit{sent\_bit}) \text{ then} \\
\qquad\quad \tau.\mathit{Sender}(\neg \mathit{sent\_bit}) \\
\qquad \text{else} \\
\qquad\quad \tau.\mathit{Sender}'(\mathit{sent\_bit}) \\
\mathit{Sender}'''(\mathit{sent\_bit}; \mathit{received\_bit}) \triangleq \\
\qquad \mathit{transmit\_msg}!(\mathit{sent\_bit}).\mathit{Sender}'(\mathit{sent\_bit}) + \\
\qquad \mathit{deliver\_ack}?(\mathit{received\_bit}).\mathit{Sender}''(\mathit{sent\_bit}, \mathit{received\_bit}) \\
\mathit{Channel}(;) \triangleq \mathit{Line}_m() \mid \mathit{Line}_a() \\
\mathit{Line}_m(; \mathit{tagging\_bit}) \triangleq \\
\qquad \mathit{transmit\_msg}?(\mathit{tagging\_bit}).\mathit{propagate\_msg}!(). \\
\qquad\quad (\tau.\mathit{deliver\_msg}!(\mathit{tagging\_bit}).\mathit{Line}_m + \\
\qquad\quad\; \tau.\mathit{Line}_m) \\
\mathit{Line}_a(; \mathit{tagging\_bit}) \triangleq \\
\qquad \mathit{transmit\_ack}?(\mathit{tagging\_bit}).\mathit{propagate\_ack}!(). \\
\qquad\quad (\tau.\mathit{deliver\_ack}!(\mathit{tagging\_bit}).\mathit{Line}_a + \\
\qquad\quad\; \tau.\mathit{Line}_a)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Receiver}(\mathit{expected\_bit}; \mathit{received\_bit}) \stackrel{\Delta}{=} \\
\qquad \mathit{deliver\_msg}?(\mathit{received\_bit}). \\
\qquad\quad \text{if}\ (\mathit{received\_bit} = \mathit{expected\_bit})\ \text{then} \\
\qquad\qquad \mathit{consume\_msg}!().\mathit{transmit\_ack}!(\mathit{received\_bit}).\mathit{Receiver}(\neg \mathit{expected\_bit}) \\
\qquad\quad \text{else} \\
\qquad\qquad \mathit{transmit\_ack}!(\mathit{received\_bit}).\mathit{Receiver}(\mathit{expected\_bit})
\end{array}
$$

It is easy to see that $[\![ABP(\mathit{false})]\!]_{\mathrm{c}}$ has twice as many states as $[\![ABP(\mathit{false})]\!]_{\mathrm{s}}$. This means that a pure process description of the alternating bit protocol, in which we must provide the behavior of each component both when the tagging bit is 0 and when the tagging bit is 1, would have twice as many states as $[\![ABP(\mathit{false})]\!]_{\mathrm{s}}$. Although in *ABP* there are two constant invocations – $\mathit{Sender}(\mathit{tagging\_bit})$ and $\mathit{Receiver}(\mathit{tagging\_bit})$ – each having one parameter that can take two different values – *false* and *true* – the actual reduction factor is 2 instead of the maximum reduction factor 4 because the two parameters are not independent of each other. We finally observe that, while the τ actions in the definition of Line_m and Line_a model the uncontrollable choice between the arrival and the loss of a message or an acknowledgement, in the definition of Sender'' two τ actions have been introduced to comply with the syntax restrictions of Def. 2.1. Such actions do not result in the generation of additional states nor τ loops. Thanks to the initial assignment mechanism, the resulting STGLA has some fewer states than the STGA' that would be generated with the rules in [Li and Chen 1999]. This can be seen e.g. by looking at the sender component. The STGA' for $\mathit{Sender}(\mathit{false})$ has one more state than the STGLA for the same term, as it has both state $\mathit{Sender}(\mathit{false})$ and state $\mathit{generate\_msg}!().\mathit{transmit\_msg}!(\mathit{sent\_bit}).\mathit{Sender}'(\mathit{sent\_bit})$ with the first one being the initial state and having no incoming transitions. ■

Example 4.3. The second example is about a comparison with the STGA' based symbolic rules of [Li and Chen 1999]. The advantage with using the STGLA based symbolic rules is not only given by the presence of the initial assignment, but comes more in general from the r-compactness. For instance, suppose that we want to model a history dependent computing device (HDCD) that works as follows. HDCD has n buttons, a numeric keyboard, and a small screen. Button i is associated with a mathematical function f_i having three real arguments and returning one real number. HDCD is started with a user selected number (history dependent number), then waits for the user to introduce two real numbers from the keyboard and press one of the buttons. HDCD then applies the function corresponding to the pressed button to the two input numbers and the history dependent number, then it applies to the previously obtained result a further mathematical function f – which is independent of the pressed button – and prints the outcome – which becomes the new history dependent number – on the screen. This procedure can subsequently be repeated an unbounded number of times. HDCD can be modeled as follows in the considered fragment of value passing CCS:

$$
\begin{array}{rcl}
\mathit{HDCD}(\mathit{hd\_num}; \mathit{num}_1, \mathit{num}_2) & \stackrel{\Delta}{=} & \sum_{i=1}^{n} \mathit{press}_i?(\mathit{num}_1, \mathit{num}_2). \\
& & \qquad \mathit{HDCD}'(f_i(\mathit{num}_1, \mathit{num}_2, \mathit{hd\_num})) \\
\mathit{HDCD}'(\mathit{num};) & \stackrel{\Delta}{=} & \mathit{print}!(f(\mathit{num})).\mathit{HDCD}(f(\mathit{num}))
\end{array}
$$

For all real numbers r, the STGLA for $HDCD(r)$ has only two states, which are $unfold(HDCD(r))$ and $print!(f(num)).HDCD(f(num))$, with n input transitions from the first state to the second one and one output transition from the second state to the first one. Instead, the STGA' for $HDCD(r)$ generated according to the rules in [Li and Chen 1999] has $n + 2$ states corresponding to $HDCD(r)$, $HDCD'(f_i(num'_1, num'_2, hd\_num))$ for $1 \leq i \leq n$, and $unfold(HDCD(f(num))$. ■

5. CONCLUSION

In this paper we have introduced a symbolic model called STGLA for value passing processes with infinite data domains, which is a variant of those of [Hennessy and Lin 1995; Lin 1996; Li and Chen 1999] and inherits from [Li and Chen 1999] the algorithms for strong, weak and observational bisimulation equivalence checking. We have then defined a set of operational symbolic semantic rules that transparently map the terms of a fragment of value passing CCS into finite and r-compact STGLA that are consistent with the operational concrete semantics and yield a correct assignment application order, thus improving on [Hennessy and Lin 1995; Lin 1996; Li and Chen 1999]. Finally, we have provided in Sect. 2.3.4 a discussion about the STGLA based treatment of the whole value passing CCS, which may be a source of further research in this area.

To conclude, we would like to mention that a slight variant of the model of STGLA and of the symbolic rules described in this paper has been implemented in the stochastic process algebra based software tool TwoTowers [Bernardo 2002], which has been successfully employed to evaluate via simulation the performance of complex data dependent systems like adaptive mechanisms for Internet telephony [Aldini et al. 2001] and dynamic server selection mechanisms for replicated Web services [Bernardo 2001].

ACKNOWLEDGMENTS

The author is grateful to the anonymous referees for their comments on earlier versions of this paper.

REFERENCES

Aldini, A., Bernardo, M., Gorrieri, R., and Roccetti, M. 2001. Comparing the qos of internet audio mechanisms via formal methods. *ACM Trans. on Modeling and Computer Simulation 11*, 1–42.

Bartlett, K. A., Scantlebury, R. A., and Wilkinson, P. T. 1969. A note on reliable full-duplex transmission over half-duplex links. *Comm. of the ACM 12*, 260–261.

Bernardo, M. 1997. Enriching empa with value passing: a symbolic approach based on lookahead. In *Proc. of the 5th Int. Workshop on Process Algebra and Performance Modelling (PAPM 1997)*. 35–49.

Bernardo, M. 2001. A simulation analysis of dynamic server selection algorithms for replicated web services. In *Proc. of the 9th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*. IEEE-CS Press, 371–378.

Bernardo, M. 2002. *TwoTowers 2.0 User Manual*. www.sti.uniurb.it/bernardo/twotowers/.

Boudol, G. and Castellani, I. 1988. A non-interleaving semantics for ccs based on proved transitions. *Fundamenta Informaticae XI*, 433–452.

Cleaveland, W. R., Parrow, J., and Steffen, B. 1993. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. on Programming Languages and Systems 15*, 36–72.

Hennessy, M. C. B. and Lin, H. 1995. Symbolic bisimulations. *Theoretical Computer Science 138*, 353–389.

Li, Z. and Chen, H. 1999. Computing strong/weak bisimulation equivalences and observation congruence for value-passing processes. In *Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1999).* LNCS, vol. 1579. Springer, 300–314.

Lin, H. 1996. Symbolic transition graph with assignment. In *Proc. of the 7th Int. Conf. on Concurrency Theory (CONCUR 1996).* LNCS, vol. 1119. Springer, 50–65.

Lin, H. 1998. On-the-fly instantiation of value-passing processes. In *Proc. of the IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV 1998).* Kluwer, 215–230.

Milner, R. 1989. *Communication and Concurrency.* Prentice Hall.

Received November 2001; revised October 2002; accepted January 2003