On Computing Reachability Sets of Process Rewrite Systems

Ahmed Bouajjani and Tayssir Touili

LIAFA, University of Paris 7, 2 place Jussieu, 75251 Paris cedex 5, France.
email: abou@liafa.jussieu.fr, touili@liafa.jussieu.fr

Abstract. We consider the problem of symbolic reachability analysis of a class of term rewrite systems called Process Rewrite Systems (PRS). A PRS can be seen as the union of two mutually interdependent sets of term rewrite rules: a prefix rewrite system (or, equivalently, a pushdown system), and a multiset rewrite system (or, equivalently, a Petri net). These systems are natural models for multi-threaded programs with dynamic creation of concurrent processes and recursive procedure calls.
We propose a generic framework based on tree automata allowing to combine (finite-state automata based) procedures for the reachability analysis of pushdown systems with (linear arithmetics/semilinear sets based) procedures for the analysis of Petri nets in order to analyze PRS models. We provide a construction which is parametrized by such procedures and we show that it can be instantiated to (1) derive procedures for constructing the (exact) reachability sets of significant classes of PRS, (2) derive various approximate algorithms, or exact semi-algorithms, for the reachability analysis of PRS obtained by using existing symbolic reachability analysis techniques for Petri nets and counter automata.

1 Introduction

Software verification is one of the main challenges in computer-aided verification. Among the difficulties to face when dealing with this problem, we can mention the fact that (1) programs manipulate data ranging over infinite domains, and the fact that (2) programs may have complex control structures. Concerning the first point, abstraction techniques (such as predicate abstraction) can be used to obtain abstract programs on finite data domains (see, e.g., [5]). As for the second point, standard model-checking algorithms can be used if the control structure of the program is finite. However, programs may have unbounded control structures due to, e.g., (unbounded depth) recursive procedure calls, and dynamic creation of concurrent processes (threads). Therefore, we need to extend the capabilities of automatic verification to deal with infinite-state models which capture the behaviors of such (abstract) programs.

Recently, model-checking techniques for infinite-state systems have been successfully used in this context. Pushdown systems have been proposed as a natural model for sequential programs with procedure calls [23, 26], whereas Petri nets have been used to reason about multi-threaded programs without procedure calls (in this case, each thread is a finite-state system, but there may be an arbitrary number of them running at the same time) [4, 19]. In both cases, symbolic reachability analysis techniques are used to verify properties, basically safety properties, on these models.

In this paper, our aim is to define reachability analysis techniques for models which subsume pushdown systems and Petri nets, allowing to deal with multi-threaded programs with procedures calls. We consider models based on term rewrite systems called *Process Rewrite Systems* (PRS). These models can be seen as combinations of prefix and multiset rewrite systems.

The construction of the reachability sets of PRS is a hard problem, in particular because they subsume Petri nets which are known to be not semilinear in general (i.e., their reachability sets cannot be defined in Presburger arithmetics) [30]. However, there exist well-known classes of semilinear Petri nets (see, e.g., [25]), and moreover, several algorithms or semi-algorithms have been developed (and implemented) recently for computing exact or upperapproximate reachability sets of counter automata, using various representation structures for linear arithmetical constraints (polyhedra, automata, etc), and fixpoint acceleration techniques [17, 8, 14, 39, 2, 18, 28, 7].

Then, our approach is to design a framework where all existing semilinear sets-based symbolic analysis procedures for Petri nets (or counter automata) can be integrated with existing automata-based symbolic analysis algorithms for pushdown systems in order to derive procedures for the reachability analysis of PRS models.

Our main contribution is a *generic* procedure which constructs a tree automata-based representation of the reachability set of a given PRS by invoking procedures for the analysis of prefix and multiset rewrite systems. While the procedure for analyzing prefix rewrite systems can be considered as fixed (e.g., the one of [15]), the construction is parameterized by a procedure for the analysis of multiset rewrite systems using semilinear sets (or Presburger arithmetics) for the representation of sets of configurations (markings). This procedure can be:

- an *exact algorithm*, but applicable to some particular subclass of multiset rewrite systems (which is known to be semilinearity preserving), or to some particular subclass of semilinear sets (which is known to be closed under multiset rewriting),
- an *approximate algorithm*, or an *exact semi-algorithm* (for which termination is not guaranteed), but applicable in general to any multiset rewrite system and to any semilinear set of configurations.

The construction we propose allows to derive, for every class of multiset rewrite systems $\mathcal{C}$ for which we have an algorithm (resp. semi-algorithm) for exact (resp. upperapproximate) reachability analysis, an algorithm (resp. semi-algorithm) for exact (resp. upperapproximate) reachability analysis for the class of PRS obtained by combining $\mathcal{C}$ systems with prefix rewrite systems. We show that our construction can be instantiated in such a manner to derive:

- an algorithm for computing the exact forward and backward reachability sets for the synchronization-free PRS (equivalent to the so-called PAD systems) which subsume pushdown systems and synchronization-free Petri nets. As a corollary, we obtain an algorithm for *global model checking* of PAD vs the EF fragment of the temporal logic CTL, i.e., an algorithm for computing the set of all configurations of a PAD system satisfying some given formula in the logic EF. These results extend all the existing ones concerning symbolic reachability analysis for subclasses of PRS, and solves for the first time the problem of global model checking of EF for PAD systems.

– exact/approximate procedures for the analysis of PRS based on various analysis procedures and invariant generation techniques for Petri nets and counter automata.

In order to characterize the reachability sets of PRS, we use a class of tree automata allowing to define (nonregular) sets of trees with unbounded width, which are closed under commutation of the children of some of their nodes (those corresponding to the parallel operator). This class of automata enjoys all closure and decision properties which are necessary in symbolic reachability analysis.

Related work: Mayr has proved in [34] that the reachability problem between two fixed terms is decidable using a reduction to the reachability problem of Petri nets. The problem we consider here is the constructibility problem of the (potentially infinite) set of terms which are reachable from a given possibly infinite set of terms.

Symbolic reachability analysis based on (word) automata techniques has been used for model-checking pushdown systems in, e.g., [9, 29]. In [33], this approach is used for the analysis of PA processes, i.e., combination of context-free prefix and multiset rewrite rules (left-hand-sides of the rules are reduced to single symbols). In that work, the authors use finite bounded-width tree automata as symbolic representation structures.

Our construction allows to handle (in particular) the class of PAD systems (or synchronization-free PRS) which is strictly larger than both pushdown systems and PA. The class PAD allows for instance to take into account return values of procedures whereas PA cannot. In the case of a pushdown system, our construction will behave like the algorithms for this class of systems (it will compute the set of all reachable configurations). In the case of PA systems our construction will also compute the precise set of all reachable configurations, whereas the construction in [33] computes only a set of representatives of terms w.r.t. associativity and associativity-commutativity of the sequential and parallel composition. Indeed, the set of all reachable configurations of PA processes is not regular in general in the sense that it cannot be represented by a bounded-width tree automaton. In [12], we have extended the approach of [33] for constructing representatives of the reachability sets to the class of PAD. The construction we give in this paper is more general in the sense that it allows to compute the whole set of all backward/forward reachable configurations, and allows also to solve the problem of global model checking for PAD against the EF fragment of CTL. The decidability problem of PAD vs EF has been shown to be decidable by Mayr [34]. However, his proof is rather complex and concerns only the model checking problem for a single process term against a formula, whereas our approach allows to compute for the first time the whole satisfiability set of a PAD formula.

The application of symbolic reachability analysis of PRS to program analysis has been advocated in [23] for sequential recursive programs using pushdown systems. This approach is extended in [27] to parallel programs using PA systems. In [21] a discussion about the modeling power of PRS is given. In [4, 19], the model of Petri nets is proposed for the verification of multi-threaded programs without procedure calls (threads are finite-state communicating systems).

In [10, 11] we define a different framework for the analysis of concurrent programs with procedure calls based on models which are either communicating pushdown systems [10] or synchronized PA systems (PA with synchronization à-la CCS) [11] The verification problem for these model is undecidable, and therefore we propose an analysis approach based on computing (either finite or commutative) abstractions of the path languages. While the considered models in [10, 11] are more general than those we consider in this paper, their analysis is (by necessity) approximate.

In [35], another approach for dealing with concurrent programs with procedures is proposed based on combining the so-called summarization technique (control location reachability in pushdown systems) with a partial order-like approach. Basically, the authors show that under some restrictions on the occurrences of synchronizations, the program has a representative (modulo some action commutations) where the summarization technique can be applied. When these conditions are not satisfied, the technique may not terminate. It is not completely clear how our models are related to the class of programs for which the algorithm of [35] terminates. However, in principle our symbolic techniques are more general than summarization-based techniques since they allow to construct the whole infinite set of reachable configurations. Moreover, the approach we present in this paper allows to deal with dynamic creation of concurrent processes.

Finally, the tree automata we use in this paper are extensions of the hedge automata [13] recognizing sets of trees with unbounded width (i.e., tree languages closed under associativity). Our automata, called commutative hedge automata, are equivalent to other kinds of automata defined recently in the literature for instance in [16, 32, 37]. However, as far as we know, our use of commutative hedge automata in the context of the analysis of process rewrite systems is original.

2 Models

We introduce hereafter Process Rewrite Systems (PRS for short) [34]. Our presentation of PRS does not follow the standard one given in [34]. In fact, we adopt the view that PRS are sets of (mutually dependent) multiset and prefix rewrite rules. It can be proved that every (standard) PRS can be transformed into an equivalent one that has the form we consider here [34].

2.1 Process terms

Let $Const = \{X, Y, \ldots\}$ be a set of process constants, and let T_p be the set of process terms t defined by the following syntax:

$$t ::= 0 \mid X \mid t \odot t \mid t || t$$

where, intuitively, 0 corresponds to the idle process, and "$\odot$" (resp. "$||$") represents the sequential composition (resp. parallel composition) operator. We use ω to denote in a generic way $\odot$ or $||$. We denote by $\overline{\omega}$ the operator $\odot$ (resp. $||$) if $\omega = ||$ (resp. $\omega = \odot$).

Process terms are considered modulo the following algebraic properties: associativity of "$\odot$", commutativity and associativity of "$||$", and neutrality of 0 w.r.t. both "$\odot$" and "$||$". Let $\simeq$ be the equivalence relation on T_p induced by these properties.

Process terms in *canonical form* are terms t defined by:

$$\begin{aligned} t &::= 0 \mid s \mid p \\ s &::= X \mid p_1 \odot p_2 \cdots \odot p_n,\ n \geq 2 \\ p &::= X \mid s_1 || s_2 \cdots || s_n,\ n \geq 2 \end{aligned}$$

It can easily be seen that every term has an $\simeq$-equivalent term in canonical form. From now on, we work only with terms in canonical form.

A *seq-term* (resp. *paral-term*) is either 0, a constant X, or a term of the form $p_1 \odot \cdots \odot p_n$, called $\odot$*-rooted term* (resp. $s_1 || \cdots || s_n$, called $||$*-rooted term*), for $n \geq 2$. A *flat seq-terms* (resp. *flat paral-terms*) is a term of the form $X_1 \odot \cdots \odot X_n$ (resp. $X_1 || \cdots || X_n$) for $n \geq 0$ (the case $n = 0$ corresponds to the term 0, and the case $n = 1$ corresponds to a process constants X).

2.2 Process Rewrite Systems

A PRS is a set of rewrite rules of the forms:

$$X_1 \odot \cdots \odot X_n \hookrightarrow Y_1 \odot \cdots \odot Y_m \tag{1}$$

$$X_1 || \cdots || X_n \hookrightarrow Y_1 || \cdots || Y_m \tag{2}$$

for $n, m \geq 0$. Rules of the form (1) (resp. (2)) are called $\odot$*-rules* (resp. $||$*-rules*).

A PRS R induces a transition relation $\rightarrow_R$ over T_p defined as the smallest relation between process terms such that:

1. if $t_1 \hookrightarrow t_2$ is a rule in R, then $t_1 \rightarrow_R t_2$,
2. if $t_1 = t \omega t_2$, and $t \rightarrow_R t'$, then $t_1 \rightarrow_R t' \omega t_2$,
3. if $t_1 \simeq t_1'$, $t_1' \rightarrow_R t_2'$, and $t_2' \simeq t_2$, then $t_1 \rightarrow_R t_2$,

Let $Post_R(t) = \{t' \in T_p \mid t \rightarrow_R t'\}$, and $Pre_R(t) = \{t' \in T_p \mid t' \rightarrow_R t\}$. As usual, $Post_R^*(t)$ and $Pre_R^*(t)$ denote respectively the reflexive-transitive closures of $Post_R(t)$ and $Pre_R(t)$. We omit the subscript R when it is understood from the context. Also, we write sometimes $R(t)$ instead of $Post_R(t)$, and similarly $R^*(t)$ instead of $Post_R^*(t)$. These definitions and notations can be extended to sets of terms in the obvious manner. Given a system R, we denote by R^{-1} the system obtained by swapping the left-hand-sides and right-hand-sides of the rules of R. Notice that for every set of process terms $\mathcal{L}$, $Pre_R^*(\mathcal{L}) = Post_{R^{-1}}^*(\mathcal{L})$.

2.3 Subclasses of PRS and Program modeling

PRS is a natural formal model for multithreaded programs with procedure calls (see, e.g., [23, 27, 21]). It subsumes several well-known classes of (infinite-state) models. We mention hereafter the ones which are relevant to this paper and mention their relevance in program modeling.

– **Prefix rewrite systems** are sets of $\odot$-rules. They are equivalent to **pushdown systems** (**PDS**). Prefix rewrite systems are models for sequential programs with procedure (recursive) calls ranging over finite data domains (see, e.g., [23, 36]): values of global variables correspond to control states, local variables and program control points are modeled as stack symbols which are stored at each recursive call.
– **BPA processes** (or context-free processes) are prefix rewrite systems where all the left-hand-sides of the rules are process constants. They are equivalent to pushdown systems with a single control state. Therefore, they do not allow to take into account global variables.
– **Multiset rewrite systems** are sets of $\|$-rules. These systems are equivalent to **Petri nets** (**PN**). They are natural models of multithreaded programs (with arbitrary number of parallel finite-state systems) (see, e.g., [18]).
– **BPP processes** (commutative context-free processes) are multiset rewrite systems where all left-hand-sides of the rules are process constants. They are equivalent to synchronization-free Petri nets.
– **PA processes** are PRSs where all the left-hand-sides of the rules are process constants (i.e., they are the nesting of BPA and BPP systems). PA processes are abstract models for programs with procedure calls and dynamic creation of (asynchronous) parallel processes [27].
– **PAD processes** are PRSs which are the nesting of BPP and prefix rewrite systems (i.e., PRS such that all the left-hand-sides of their rules are seq-terms). PAD systems subsume pushdown systems and PA processes. Contrary to PA processes, they allow to take into account return values of procedure calls.

Let $\mathcal{C}$ be a class of multiset rewrite systems. We denote by **PRS[$\mathcal{C}$]** the class of PRSs that are unions of prefix rewrite systems and multiset rewrite systems in $\mathcal{C}$. For instance, PRS[BPP] is precisely the class PAD.

Given a class $\mathcal{C}$ of PRS, we denote by co$\mathcal{C}$ the *dual* class of $\mathcal{C}$ which consists of all systems R such that R^{-1} is in $\mathcal{C}$. Clearly, PRS = coPRS, and the same holds for prefix and multiset rewrite systems.

3 Tree Automata-based symbolic representations

We use a class of tree automata, called *commutative hedge automata*, for the representation and the manipulation of infinite sets of PRS process terms. The automata we consider extend (bottom-up) hedge automata recognizing sets of arbitrary-width trees [13]. They recognize sets of terms modulo associativity of $\odot$ and associativity-commutativity of $\|$. Our automata are very close to the ones defined in [16], and can be seen as particular cases of those introduced in [37, 32]. Let us give briefly the intuition behind the definition of these automata: Consider first a "classical" bottom-up tree automaton over fixed-width trees, say binary trees. To accept a tree in a bottom-up manner, the automaton has to find a labelling of the nodes of this tree by control states which (1) puts a final state at the root, and (2) is compatible with rules of the form: either (*i*) $a \to q$ allowing to label a leaf a with state q, or (*ii*) $f(q_1, q_2) \to q$ allowing to label a tree $f(t_1, t_2)$ with q provided that its subtrees t_1 and t_2 are labeled by q_1 and q_2. Now, assume that f represents an associative operator. Then, we consider that a node corresponding to such an operator can have an arbitrary number of sons. Therefore, we use labelling rules of the form $f(L) \to q$, where L is a regular language over the alphabet of control states, which allow to label a tree $f(t_1,\ldots,t_n)$ with q provided that each sub-tree t_i is labelled by q_i and the word $q_1 \cdots q_n$ is in the language L. Assume furthermore that f is associative *and* commutative. In such a case, the ordering between sons is not relevant. Therefore, we use rules of the form $f(\varphi) \to q$, where φ is an arithmetical constraint, allowing to label a tree $f(t_1,\ldots,t_n)$ with q provided that each subtree t_i is labelled by q_i and the number of occurrences of each control state in the word $q_1 \cdots q_n$ satisfies the constraint expressed by φ.

In the sequel, we give the definition of the general class of commutative hedge automata, and then we describe the particular automata which are used for the representation of sets of process terms.

3.1 Preliminaries

Presburger arithmetics is the first order logic of integers with addition and linear ordering. Given a formula φ, we denote by $FV(\varphi)$ the set of its free variables. Let $FV(\varphi) = \{x_1,\ldots,x_n\}$. Then, a vector $u = (u_1,\ldots,u_n) \in \mathbb{Z}^n$ satisfies φ, written $u \models \varphi$, if $\varphi(u) = \varphi[x_i \leftarrow u_i]$ is true. Each formula φ defines a set of integer vectors $[\![\varphi]\!] = \{u \in \mathbb{Z}^n \mid u \models \varphi\}$. Presburger formulas define *semilinear sets* of integer vectors, i.e., finite union of sets of the form $\{x \in \mathbb{Z}^n \mid \exists k_1,\ldots,k_m \in \mathbb{Z}, x = v_0 + k_1 v_1 \cdots + k_m v_m\}$, where $v_i \in \mathbb{Z}^n$, for $1 \leq i \leq m$.

Given a word w over an alphabet $\Sigma = \{a_1,\ldots,a_n\}$, the *Parikh image* of w, denoted $Parikh(w)$, is the vector $(|w|_{a_1},\ldots,|w|_{a_n})$. This definition can be generalized to sets of words (languages) over Σ in the obvious manner.

As usual, a set of words is *regular* if it is definable by a finite-state automaton. The notion of regularity can be transfered straightforwardly to sets of flat seq-terms. Similarly, the notion of semilinearity can be transfered to sets of flat paral-term by associating with a term $X_1 \| \cdots \| X_n$ the vector $Parikh(X_1 \cdots X_n)$.

In the sequel, we will represent by γ a *constraint* which is either a regular language or a Presburger formula. We say that a word $w = a_1 a_2 \ldots a_n$ *satisfies* the constraints γ if $w \in \gamma$ (resp. $Parikh(w) \models \gamma$) when γ is a language (resp. a formula).

3.2 Commutative Hedge Automata

Let $\Sigma = \Sigma' \cup \Sigma_A$ be a finite alphabet, where Σ' is a ranked alphabet, and Σ_A is a finite set of associative operators. We assume that Σ' and Σ_A are disjoint. For $k \geq 0$, let Σ_k denote the set of elements of Σ' of rank k.

Σ-Terms: Let $\mathcal{X}$ be a fixed denumerable set of variables $\{x_1, x_2, \ldots\}$. The set $T_\Sigma[\mathcal{X}]$ of Σ-terms over $\mathcal{X}$ is the smallest set such that:

- $\Sigma_0 \cup \mathcal{X} \subseteq T_\Sigma[\mathcal{X}]$,
- for $k \geq 1$, if $f \in \Sigma_k$ and $t_1,\ldots,t_k \in T_\Sigma[\mathcal{X}]$, then $f(t_1,\ldots,t_k) \in T_\Sigma[\mathcal{X}]$,
- if $f \in \Sigma_A$, $t_1,\ldots,t_n \in T_\Sigma[\mathcal{X}]$ for some $n \geq 1$, and $root(t_i) \neq f$ for every $1 \leq i \leq n$, then $f(t_1,\ldots,t_n) \in T_\Sigma[\mathcal{X}]$, where $root(\sigma) = \sigma$ if $\sigma \in \Sigma_0 \cup \mathcal{X}$, and $root\big(g(u_1,\ldots,u_m)\big) = g$.

Terms without variables are called *ground terms*. Let T_Σ be the set of ground terms over Σ. A term t in $T_\Sigma[X]$ is *linear* if each variable occurs at most once in t. A *context* C is a linear term of $T_\Sigma[X]$. Let $t_1, \ldots, t_n$ be terms of T_Σ, then $C[t_1, \ldots, t_n]$ denotes the term obtained by replacing in the context C the occurrence of the variable x_i by the term t_i, for each $1 \leq i \leq n$.

Definition of CH-automata Let us consider that $\Sigma_A = \Sigma'_A \cup \Sigma'_{AC}$ where Σ'_{AC} is a set of associative and commutative operators. We assume that Σ'_A and Σ'_{AC} are disjoint. Then, a CH-automaton is a tuple $\mathcal{A} = (Q, \Sigma, F, \Delta)$ where:

- Q is a union of disjoint finite sets of states $Q' \cup \bigcup_{f \in \Sigma_A} Q_f$,
- $F \subseteq Q$ is a set of final states,
- Δ is a set of rules of the form:
  1. $a \to q$, where $q \in Q', a \in \Sigma_0$,
  2. $f(q_1, \ldots, q_k) \to q$, where $f \in \Sigma_k, q \in Q'$, and $q_i \in Q$,
  3. $q \to q'$, where $(q, q') \in Q' \times Q' \cup \bigcup_{f \in \Sigma_A} Q_f \times Q_f$,
  4. $f(L) \to q$, where $f \in \Sigma'_A$, $L \subseteq (Q \setminus Q_f)^*$, and $q \in Q_f$,
  5. $f(\varphi) \to q$, where $f \in \Sigma'_{AC}$, $q \in Q_f$, and φ is a Presburger formula such that $FV(\varphi) = \{x_q \mid q \in Q \setminus Q_f\}$.

We define a *move relation* $\to_\Delta$ between ground terms in $T_{\Sigma \cup Q}$ as follows: for every two terms t and t', we have $t \to_\Delta t'$ iff there exist a context C and a rule $r \in \Delta$ such that $t = C[s]$, $t' = C[s']$, and:

- $r = a \to q$, with $s = a$ and $s' = q$, or
- $r = q \to q'$, with $s = q$ and $s' = q'$, or
- $r = f(q_1, \ldots, q_k) \to q$, with $s = f(q_1, \ldots, q_k)$ and $s' = q$, or
- $r = f(L) \to q$, with $f \in \Sigma'_A$, $s = f(q_1, \ldots, q_n)$, $q_1 \cdots q_n \in L$, and $s' = q$, or
- $r = f(\varphi) \to q$, with $f \in \Sigma'_{AC}$, $s = f(q_1, \ldots, q_n)$, $Parikh(q_1 \cdots q_n) \models \varphi$, and $s' = q$.

Let $\xrightarrow{*}_\Delta$ denote the reflexive-transitive closure of $\to_\Delta$. A ground term $t \in T_\Sigma$ is accepted by a state q if $t \xrightarrow{*}_\Delta q$. Let $L_q = \{t \mid t \xrightarrow{*}_\Delta q\}$. A ground term t is accepted by the automaton $\mathcal{A}$ if there is some state q in F such that $t \xrightarrow{*}_\Delta q$. The CH-language of $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of all ground terms accepted by $\mathcal{A}$.

By adapting the constructions for hedge automata [13], it is possible to prove the following fact (see [16, 32, 37, 38]):

Theorem 1. *The class of CH-automata is effectively closed under boolean operations. Moreover, the emptiness problem of CH-automata is decidable.*

3.3 CH-automata for PRS process terms

We consider PRS process terms as trees and use CH-automata to represent sets of such trees. Indeed, the set T_p of PRS process terms can be seen as the set of Σ-terms T_Σ where $\Sigma_0 = \{0\} \cup Const$, $\Sigma'_A = \{\odot\}$, and $\Sigma'_{AC} = \{\|\}$.

Sets of process terms are recognized by CH-automata $\mathcal{A} = (Q, \Sigma, F, \Delta)$ such that (1) Q is the disjoint union $Q = Q' \cup Q_\odot \cup Q_\|$ where Q' is itself the disjoint union $Q' =$

$Q_0 \cup Q_-$, and (2) the rules in Δ are of the form: (a) $X \to q$, where $q \in Q_-, X \in Const$, (b) $0 \to q$, where $q \in Q_0$, (c) $q \to q'$, where $(q,q') \in (Q_0)^2 \cup (Q_-)^2 \cup (Q_\odot)^2 \cup (Q_\|)^2$, (d) $\odot(L) \to q$, where $L \subseteq \big(Q \setminus (Q_\odot \cup Q_0)\big)^*$ and $q \in Q_\odot$, and (e) $\|(\varphi) \to q$, where $q \in Q_\|$, and φ is a Presburger formula such that $FV(\varphi) = \{x_q \mid q \in Q \setminus (Q_\| \cup Q_0)\}$. In other words, the states in $Q_\odot$ (resp. $Q_\|$) recognize trees whose root is $\odot$ (resp. $\|$). The states in Q_- recognize constants in *Const*, and the states in Q_0 recognize 0.

4 A generic construction of PRS reachability sets

We provide a construction of the reachability analysis of PRS which is *parameterized* by two algorithms $\Theta_\odot$ and $\Theta_\|$ such that (1) $\Theta_\odot$ is an algorithm for the reachability analysis of prefix rewrite systems based on regular languages, and (2) $\Theta_\|$ is an algorithm for the reachability analysis of multiset rewrite systems based on semilinear sets.

4.1 Preliminaries

A class of multiset rewrite systems (Petri nets) $\mathcal{C}$ is *effectively semilinear* if we have for it an algorithm $\Theta_\|$ which constructs, for every given system $M \in \mathcal{C}$ and every semilinear set of paral-terms S (markings), a set $\Theta_\|(M,S)$ which is semilinear and equal to $M^*(S)$.

Let us fix for the rest of this section an effectively semilinear class $\mathcal{C}$ of multiset rewrite systems and let $\Theta_\|$ be the algorithm we have for its symbolic reachability analysis. For a reason which will be clear in the next subsection, we assume that $\mathcal{C}$ is *1-rules closed*, i.e., for every system $M \in \mathcal{C}$, and every X, Y in *Const*, the system $M \cup \{X \to Y\}$ is also in the class $\mathcal{C}$.

Let us fix also an algorithm $\Theta_\odot$ for computing regular reachability sets of prefix rewrite systems (let us assume for instance that it corresponds to one of the algorithms in [15, 9, 22]).

Then, let us consider a PRS $R = R_\odot \cup R_\|$, where $R_\odot$ is a prefix rewrite system, and $R_\|$ is a multiset rewrite system in $\mathcal{C}$, and let $\mathcal{A} = (Q, \Sigma, F, \Delta)$ be a CH-automaton recognizing a set of initial configurations (process terms) $\mathcal{L}$. The rest of this section is devoted to the construction of a CH-automaton $\mathcal{A}^*[R, \Theta_\odot, \Theta_\|] = (\widetilde{Q}, \Sigma, \widetilde{F}, \widetilde{\Delta})$ which recognizes $R^*(\mathcal{L})$, where $\widetilde{Q}$ is the set of states, $\widetilde{F}$ is the set of final states, and $\widetilde{\Delta}$ the set of rules.

4.2 The set of states:

The set of states $\widetilde{Q}$ includes the set of states Q of $\mathcal{A}$ and contains new states q_X, which are assumed to accept precisely the singletons $\{X\}$ (i.e., $L_{q_X} = \{X\}$), for each $X \in Const$. Let Q_R be the set of states $\{q_X \mid X \in Const\}$. In addition, the set $\widetilde{Q}$ contains states which recognize the *successors* by R of terms in L_q for each $q \in Q \cup Q_R$ (these states will have this property at the end of the iterative construction of automaton described below). In order to ensure (during the construction) that the recognized trees are always in canonical form, we need to partition the sets of recognized trees according to their types (given by their root). We associate with each $q \in Q \cup Q_R$ different states $(q,-), (q,0), (q,\odot)$, and $(q,\|)$ recognizing successors of terms in L_q which are respectively constants in *Const*, null (equal to 0), $\odot$-rooted terms, and $\|$-rooted terms.

Let $Q = Q_0 \cup Q_- \cup Q_\odot \cup Q_\|$. We consider that the set $\widetilde{Q}$ is equal to the union of the following sets: (1) $\widetilde{Q}_0 = Q_0 \cup \{(q,0) \ : \ q \in Q \cup Q_R\}$, (2) $\widetilde{Q}_- = Q_- \cup Q_R \cup \{(q,-) \ :$

$q \in Q \cup Q_R\}$, and (3) $\widetilde{Q}_\omega = Q_\omega \cup \{(q,\omega) \ : \ q \in Q \cup Q_R\}$, for $\omega \in \{\odot, \|\}$. Moreover, we consider that $\widetilde{F} = \{q, (q,-), (q,0), (q,\odot), (q,\|) \ : \ q \in F\}$.

4.3 Nested Prefix/Multiset Rewriting

The construction of the automaton $\mathcal{A}^*[R, \Theta_\odot, \Theta_\|]$ is based on closure rules which add new transitions to those originally in the automaton $\mathcal{A}$ in order to recognize terms obtained by applying the transitive closure of R. The added transitions are defined by computing new constraints reflecting the effect of applying sequences of rewriting steps using the systems $R_\odot$ and $R_\|$. Intuitively, given a rule $\omega(\gamma) \rightarrow q$, we would like to add a rule $\omega(\gamma') \rightarrow q$ where γ' is obtained, roughly speaking, by applying Θ_ω to γ. However, many problems appear since $\odot$ and $\|$-rooted terms are nested, and the applications of $R_\odot$ and $R_\|$ may interact at different levels of the term. The main issue is to deal with these interactions in such a manner that only a finite number of transitions needs to be added to the automaton $\mathcal{A}$. The crucial idea is to use, instead of the system R, an extended system R' (called its transitive normal form) where nested prefix and multiset rewriting have been taken into account. The computation of this system is a preliminary step of our construction. Then, the construction itself deals with the problems which come from closing the language of the given CH-automaton under the application of the system S.

Transitive Normal Form: The *transitive normal form* of R is the union of the multiset and prefix rewrite systems $R'_\|$ and $R'_\odot$ defined, for $\omega \in \{\odot, \|\}$, by $R'_\omega \ = \ R_\omega \cup \{X \rightarrow Y \ : \ Y \in R^*(X)\}$.

Lemma 1. *We have $R^* = (R')^*$. Moreover, for every flat seq-term (resp. paral-term) t, ${R'_\odot}^*(t)$ (resp. ${R'_\|}^*(t)$) is the set of all the flat seq-terms (resp. paral-terms) in $R^*(t)$.*

The systems R'_ω, for $\omega \in \{\odot, \|\}$, can be computed iteratively as the limits of the ascending chains of sets of rules $(R_i^\omega)_{i \geq 0}$ defined by $R_0^\omega = R_\omega$, and $R_{i+1}^\omega = R_i^\omega \cup \{X \rightarrow Y \ : \ Y \in \Theta_{\overline{\omega}}(R_i^{\overline{\omega}}, X)\}$, for $i \geq 0$. Notice that these chains are finite since there is a finite number of pairs (X,Y).

Rewrite system over the alphabet of states: Rules in CH-automata (of the forms $\omega(\gamma) \rightarrow q$) involve constraints on sequences of *states*, whereas the systems $R'_\odot$ and $R'_\|$ are defined over the alphabet of process constants. Therefore, we define the systems $R''_\odot = \alpha(R'_\odot)$ and $R''_\| = \alpha(R'_\|)$ where α is the substitution such that $\alpha(X) = q_X$, for every $X \in Const$ (extended in the standard way to terms, rules, and sets of rules).

Successor Closure: The system S used in the construction is the union of the systems S_ω, for $\omega \in \{\odot, \|\}$, defined by $S_\omega = R''_\omega \cup \{q \rightarrow (q,-), (q,\overline{\omega}) \ : \ q \in Q \cup Q_R\}$.

The role of the additional rules is, roughly speaking, to close the set of accepted trees under the "succession relation": If a rule $\|(\varphi) \rightarrow q$ is added to the automaton, whenever it allows to recognize a tree $\|(t_1, t_2, \cdots, t_n)$ at state q, it should also recognize any tree of the form $\|(t'_1, t'_2, \cdots, t'_n)$ where each t'_i is a (constant or $\odot$-rooted) successor of t_i. For the case of the operator $\odot$, this closure concerns only the left-most tree due to prefix rewriting.

4.4 The set of transition rules

The set $\widetilde{\Delta}$ is inductively defined as the smallest set of transition rules which (1) contains Δ, (2) contains the set of rules $X \to q_X$ for every $X \in Const$, and (3) is such that:

(β_1) **Initialization rules:**
For every state $q \in Q \cup Q_R$, (a) if $q \in Q_0$, then $0 \to (q,0) \in \widetilde{\Delta}$, (b) if $q \in Q_- \cup Q_R$, then $q \to (q,-) \in \widetilde{\Delta}$, and (c) if $q \in Q_\omega$, then $q \to (q,\omega) \in \widetilde{\Delta}$.

These rules express that $L_q \subseteq L_{(q,0)}$ if $q \in Q_0$, $L_q \subseteq L_{(q,-)}$ if $q \in Q_- \cup Q_R$, and $L_q \subseteq L_{(q,\omega)}$ if $q \in Q_\omega$.

(β_2) **Simulation of the ε-rules:**
If $q \to q' \in \Delta$, then $\{(q,-) \to (q',-), (q,0) \to (q',0), (q,\omega) \to (q',\omega)\} \subseteq \widetilde{\Delta}$.

This rule expresses that if initially $L_q \subseteq L_{q'}$, then any successor of L_q is also a successor of $L_{q'}$.

(β_3) **Term flattening rules:**
(a) If $\omega(\gamma) \to (q,\omega) \in \widetilde{\Delta}$, $(q',-) \in \gamma$, and $(q',\overline{\omega}) \in \gamma$, then $(q',0) \to (q,0) \in \widetilde{\Delta}$, $(q',-) \to (q,-) \in \widetilde{\Delta}$, and $(q',\overline{\omega}) \to (q,\overline{\omega}) \in \widetilde{\Delta}$,
(b) If $\omega(\gamma) \to (q,\omega) \in \widetilde{\Delta}$ and $0 \in \gamma$, then $0 \to (q,0) \in \widetilde{\Delta}$.

These rules express that if $\omega(t)$ is a successor of L_q, then t and all its successors are also successors of L_q.

(β_4) **Closure rules: successors of process constants and 0:**
(a) If $X \xrightarrow{*}_{\widetilde{\Delta}} (q,-)$, then $\omega\big(\Theta_\omega(S_\omega, q_X)\big) \to (q,\omega) \in \widetilde{\Delta}$,
(b) If $0 \xrightarrow{*}_{\widetilde{\Delta}} (q,0)$, then $\omega\big(\Theta_\omega(S_\omega, 0)\big) \to (q,\omega) \in \widetilde{\Delta}$.

The rule (a) says that if X is a successor of some term in L_q, then all its ω-successors obtained by applying the system R'_ω are also successors of L_q. The rule (b) says the same thing for successors of 0.

(β_5) **Closure rule: successors of ω-rooted terms:**
If $\omega(\gamma) \to p \in \Delta$, then $\omega\big(\Theta(S'_\omega, \sigma(\gamma))\big) \to (p,\omega) \in \widetilde{\Delta}$, where:
$-$ σ is the substitution such that $\forall q \in Q \cup Q_R$, $\sigma(q) = \{q\} \cup \{q_X \ : \ X \xrightarrow{*}_{\Delta} q\}$,
$-$ $S'_\omega = S_\omega \cup \{q \to q_X \ : \ X \xrightarrow{*}_{\widetilde{\Delta}} (q,-), q \in Q \cup Q_R\} \cup \{q \to 0 \ : \ 0 \xrightarrow{*}_{\widetilde{\Delta}} (q,0)\}$

This rule concerns the case of ω-rooted terms where rewritings have not occurred so far at their root (they have occurred starting from the level of the child of the root). The rule says the following: Let $t = \omega(t_1, \ldots, t_n)$ be a term initially accepted at p due to the fact that each of the t_i's can be labelled with a q_i with $q_1 \cdots q_n \models \gamma$. Let t' be a successor of t, and assume that it is equal to t where some of the t_i's have been rewritten to 0*, and some others have been transformed into constants X. Then, any successor of t', obtained by applying the rewrite system R'_ω to the subsequence of all constants appearing among its first-level children, must also be a successor of t.*

The set of rules $\widetilde{\Delta}$ can be constructed iteratively as the limit of an increasing sequence $\widetilde{\Delta}_1 \subseteq \widetilde{\Delta}_2 \subseteq \ldots$ of set of rules, obtained by applying iteratively the closure and flattening β-rules. It can be seen that this iterative procedure terminates. Indeed, the flattening rules (β_3) create rewrite rules of the form $(q,\sim) \to (q',\sim)$. Since there is a finite number of pairs (q,q'), (β_3) can only be applied a finite number of times. The same holds for the closure rules (β_4) and (β_5) since they can only be applied if there is a new pair (X,q) such that $X \xrightarrow{*}_{\widetilde{\Delta}} (q,-)$, and there is a finite number of such pairs. We can prove the following fact (the proof can be found in the appendix and in [38]):

Lemma 2. *For every process term t, and every $q \in Q \cup Q_R$ we have: (1) $t \xrightarrow{*}_{\widetilde{\Delta}} (q,0)$ iff $t \in Post^*(L_q)$ and $t = 0$, (2) $t \xrightarrow{*}_{\widetilde{\Delta}} (q,-)$ iff $t \in Post^*(L_q)$ and $t \in Const$, and (3) $t \xrightarrow{*}_{\widetilde{\Delta}} (q,\omega)$ iff $t \in Post^*(L_q)$ and $root(t) = \omega$, for $\omega \in \{\odot, \|\}$.*

Theorem 2. *Let $\Theta_\odot$ be an algorithm for computing $Post^*$ images of regular sets by prefix rewrite systems. Let $\mathcal{C}$ be a 1-rules closed, effectively semilinear class of multiset rewrite systems, and let $\Theta_\|$ be an algorithm for computing $Post^*$ images of semilinear sets by systems in $\mathcal{C}$. Then, for every system R in PRS[$\mathcal{C}$], and every CH-automaton $\mathcal{A}$, we have $Post^*_R\big(L(\mathcal{A})\big) = L(\mathcal{A}^*[R,\Theta_\odot,\Theta_\|])$.*

5 Application: Reachability Analysis of PRS

5.1 Computing Reachability Sets of PAD systems

As said in section 2, the class of PAD system is precisely the class PRS[BPP], and moreover, the class coPAD is equal to PRS[coBPP]. It has been shown that the reachability relation of BPP systems is semilinear and effectively constructible [20]. This implies that both classes BPP and coBPP are effectively semilinear. These classes are also 1-rules closed, by definition. Therefore, an immediate consequence of Theorem 2 is the following result:

Corollary 1. *For every PAD system R, and every CH-automaton $\mathcal{A}$, the sets $Post^*_R\big(L(\mathcal{A})\big)$ and $Pre^*_R\big(L(\mathcal{A})\big)$ are computable and effectively representable by CH-automata.*

5.2 Global Model Checking for EF

We consider the EF fragment of the temporal logic CTL: the set of formulas built over a finite set of atomic propositions, and closed under boolean operators and the EF operator (EFϕ means that there exists a computation path where ϕ is eventually true). We consider that atomic propositions are interpreted as CH-automata definable sets of process terms.

The global model checking problem is "given a PRS R and a formula ϕ, compute the set of all configurations (process terms) satisfying ϕ". Then, since EF corresponds to the Pre^* operator and CH-automata are closed under boolean operations (Theorem 1), we obtain as a consequence of Corollary 1 the fact that:

Corollary 2. *For every PAD system R, and for every formula ϕ in the EF fragment of CTL over CH-automata definable valuations of atomic propositions, the set of configurations of R satisfying ϕ is computable and effectively representable by a CH-automaton.*

5.3 Integrating Reachability Analysis Procedures for Pushdown systems and Petri nets

Our construction provides a framework for extending any procedure (exact or approximate) for symbolic reachability analysis of Petri nets using semilinear sets (or linear arithmetics) to a procedure for symbolic reachability analysis of PRS.

Indeed, we can compute upper approximations of reachability sets of PRS by instantiating the parameter $\Theta_{||}$ of the construction by an algorithm which computes, for any given multiset rewrite system, a semilinear upper approximation of its reachability set. There are several such algorithms, leading to different analysis procedures for PRS with different precisions, such as the Karp-Miller algorithm for computing the coverability set, and procedures which generate invariants based on, e.g, flow constraints and trap constraints (see [24]).

Other possible instantiations can be obtained by using various existing *approximate* algorithms or *exact* semi-algorithms for symbolic reachability analysis of counter automata (multiset rewrite systems can of course be seen as particular cases of such general models). Many of such procedures have been developed in the last few years using either polyhedra-based representations or automata-based representations for linear constraints or Presburger arithmetics [17, 8, 14, 39, 2, 18, 28, 7]. These procedures are quite efficient thanks to the use of different fixpoint acceleration techniques (such as widening-based techniques [17, 14, 2, 7], or meta-transition techniques [8, 28]) allowing to force or to help termination of the reachability analysis.

6 Conclusion

We have defined automata-based techniques for computing reachability sets of PRS. These techniques provide a general framework for modeling and analyzing programs with dynamic creation of concurrent processes and recursive procedure calls. Indeed, such programs can be modeled quite naturally using term rewrite systems combining prefix and multiset rewrite systems. Then, our generic construction allows to use procedures for reachability analysis of both kinds of rewrite systems in order to derive various analysis procedures for their combination. This allows in particular to establish new analysis and model-checking algorithms for the class of PAD systems. Our results generalize and unify all existing results on the kind of models we consider based on process rewrite systems. Furthermore, our construction provides a theoretical basis for the construction of a tool for symbolic analysis of multithreaded programs modeled as PRS, based on the integration of existing, and quite efficient, tools for symbolic reachability analysis of pushdown systems (e.g., [36]) and tools for symbolic reachability analysis of counter automata (e.g., [31, 3, 1, 6]).

References

1. P. Ammirati, G. Delzanno, P. Ganty, G. Geeraerts, J.-F. Raskin, and L. Van Begin. Babylon: An integrated toolkit for the specification and verification of parameterized systems. In *SAVE'02*, 2002.

2. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic Techniques for Parametric Reasoning about Counter and Clock Systems. In *CAV'00*. LNCS 1855, 2000.
3. A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A Tool for Reachability Analysis of Complex Systems. In *CAV'01*. LNCS 2102, 2001.
4. T. Ball, S. Chaki, and S. K. Rajamani. Parameterized verification of multithreaded software libraries. In *TACAS 2001, LNCS 2031*, 2001.
5. T. Ball, A. Podelski, and S. K. Rajamani. Boolean and cartesian abstractions for model checking c programs. In *TACAS 2001, LNCS 2031*, 2001.
6. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In *CAV'03*. LNCS 2725, 2003.
7. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. CAV'03*. LNCS 2725, 2003.
8. B. Boigelot and P. Wolper. Symbolic Verification with Periodic Sets. In *CAV'94*. LNCS 818, 1994.
9. A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.
10. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *Proc. of the 30th ACM Symp. on Principles of Programming Languages, (POPL'03)*, 2003.
11. A. Bouajjani, J. Esparza, and T. Touili. Reachability Analysis of Synchronized PA Systems. In *Proc. Infinity Workshop*, 2004.
12. Ahmed Bouajjani and Tayssir Touili. Reachability Analysis of Process Rewrite Systems. In *Proc. 23rd Intern. Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*. LNCS 2914, 2003.
13. A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Research report, 2001.
14. T. Bultan, R. Gerber, and C. League. Verifying Systems With Integer Constraints and Boolean Predicates: A Composite Approach. In *Proc. of the Intern. Symp. on Software Testing and Analysis*. ACM press, 1998.
15. D. Caucal. On the regular structure of prefix rewriting. *Theoret. Comput. Sci.*, 106:61–86, 1992.
16. T. Colcombet. Rewriting in the partial algebra of typed terms modulo ac. In *Electronic Notes in Theoretical Computer Science, volume 68. Elsevier Science Pub., Proc. Infinity Workshop*, 2002.
17. Patrick Cousot and Nicholas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL'78*. ACM, 1978.
18. G. Delzanno, L. Van Begin, and J.-F. Raskin. Attacking symbolic state explosion in parametrized verification. In *CAV'01*, 2001.
19. G. Delzanno, L. Van Begin, and J.-F. Raskin. Toward the automated verification of multithreaded java programs. In *TACAS 2002*, 2002.
20. J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Fundamentals of computation theory*, volume 965 of *LNCS*, 1995.
21. J. Esparza. Grammars as processes. In *Formal and Natural Computing*. LNCS 2300, 2002.
22. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. In *CAV'00*, volume 1885 of *LNCS*, 2000.
23. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FOSSACS'99*. LNCS 1578, 1999.
24. J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16, 2000.
25. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:85–107, 1994.

26. J. Esparza and S. Schwoon. A bdd-based model checker for recursive programs. In *In Proc. of CAV'01, number 2102 in Lecture Notes in Computer Science, pages 324-336. Springer-Verlag*, 2001.
27. Javier Esparza and Andreas Podelski. Efficient algorithms for pre * and post * on interprocedural parallel flow graphs. In *Symposium on Principles of Programming Languages*, pages 1–11, 2000.
28. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FSTTCS'02*. LNCS 2556, 2002.
29. A. Finkel, B. Willems, and P. Wolper. A Direct Symbolic Approach to Model Checking Pushdown Systems. In *Infinity'97*, 1997.
30. J. Hopcroft and J.-J. Pansiot. On The Reachability Problem for 5-Dimensional Vector Addition Systems. *Theoret. Comput. Sci.*, 8, 1979.
31. The Liège Automata-based Symbolic Handler (LASH), 2001. Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
32. D. Lugiez. Counting and equality constraints for multitree automata. In *FoSSaCS 2003*, pages 328–342, 2003.
33. D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98), Nice, France, Sep. 1998*, volume 1466, pages 50–66. Springer, 1998.
34. R. Mayr. Decidability and Complexity of Model Checking Problems for Infinite-State Systems. Phd. thesis, Techn. Univ. of Munich, 1998.
35. S. Qadeer, S.K. Rajamani, and J. Rehof. Procedure Summaries for Model Checking Multithreaded Software. In *POPL'04*. ACM, 2004.
36. Stefan Schwoon. *Model-Checking Pushdown Systems*. PhD thesis, Technische Universität München, 2002.
37. H. Seidl, Th. Schwentick, and A. Muscholl. Numerical Document Queries. In *PODS'03*. ACM press, 2003.
38. Tayssir Touili. Analyse symbolique de systèmes infinis basée sur les automates: Application à la vérification de systèmes paramétrés et dynamiques. Phd. thesis, University of Paris 7, 2003.
39. P. Wolper and B. Boigelot. Verifying Systems with Infinite but Regular State Spaces. In *CAV'98*. LNCS 1427, 1998.

A Proof of Lemma 2

In the sequel, since there is no possible confusion concerning the procedures Θ_ω, we simply write $R^*_\omega(L)$ instead of $\Theta_\omega(R_\omega, L)$.

Lemma 2 is a direct consequence of the three Lemmas we prove hereafter.

Lemma 3. *For every term v, and every $q \in Q \cup Q_R$ we have:*

- $v \xrightarrow{*}_{\widetilde{\Delta}} (q,0) \Rightarrow v \in Post^*(L_q)$*, and* $v = 0$*,*
- $v \xrightarrow{*}_{\widetilde{\Delta}} (q,-) \Rightarrow v \in Post^*(L_q)$*, and* $v \in Const$*,*
- $v \xrightarrow{*}_{\widetilde{\Delta}} (q,\omega) \Rightarrow v \in Post^*(L_q)$*, and* $root(v) = \omega$*, for* $\omega \in \{\odot, ||\}$*.*

Proof. We show by induction on i that for every term v, and every $q \in Q \cup Q_R$ we have:

(a) $[v \xrightarrow{*}_{\widetilde{\Delta}_i} (q,-) \Rightarrow v \in Post^*(L_q)$, and $v \in Const]$, $[v \xrightarrow{*}_{\widetilde{\Delta}_i} (q,0) \Rightarrow v \in Post^*(L_q)$, and $v = 0]$,

(b) $v \xrightarrow{*}_{\widetilde{\Delta}_i} (q,||) \Rightarrow v \in Post^*(L_q)$, and $root(v) = ||$,

(c) $v \xrightarrow{*}_{\widetilde{\Delta}_i} (q,\odot) \Rightarrow v \in Post^*(L_q)$, and $root(v) = \odot$.

Moreover, we show by induction on i that for every $q \in Q \cup Q_R$ we have:

(d) if $\odot(L) \to (q,\odot) \in \widetilde{\Delta}_i$, then for every $q_1,\ldots,q_n$ such that $q_1 \cdots q_n \in L$, for every $t_1,\ldots,t_n$ such that $t_j \in Post^*(L_{p_j})$ if $q_j \in \{(p_j,-),(p_j,||)\}$, and $t_j \in L_{q_j}$ if $q_j \in Q \cup Q_R$, $root(t_j) \neq \odot$, and $t_j \neq 0$, we have $\odot(t_1,\ldots,t_n) \in Post^*(L_q)$,

(e) if $||(\varphi) \to (q,||) \in \widetilde{\Delta}_i$, then for every $q_1,\ldots,q_n$ such that $Parikh(q_1 \cdots q_n) \models \varphi$, for every $t_1,\ldots,t_n$ such that $t_j \in Post^*(L_{p_j})$ if $q_j \in \{(p_j,-),(p_j,\odot)\}$, and $t_j \in L_{q_j}$ if $q_j \in Q \cup Q_R$, $root(t_j) \neq ||$, and $t_j \neq 0$, we have $||(t_1,\ldots,t_n) \in Post^*(L_q)$,

These statements have to be shown simultaneously. Let us start the induction:

• $i = 0$. Let us start with $\widetilde{\Delta}_0$ that contains the rules Δ, in addition to the rules (β_1) and (β_2). Then it is easy to see that the properties hold.

• $i > 0$.Let us show the properties (d) and (e) first. For simplicity we will not mention the nature of the terms t_i's (their roots and the information about whether they are null or not) considered in the statements. This property concerning the roots can be verified easily.

⋆ Suppose that $\odot(L) \to (q,\odot) \in \widetilde{\Delta}_i \setminus \widetilde{\Delta}_{i-1}$. There are many cases depending on the inference rule by which this transition rule has been created:

1. $\odot(L) \to (q,\odot)$ is the consequence of a β_4-rule. Let then X be the process variable such that $L = S^*_\odot(q_X)$, and $X \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (q,-)$. Then by induction it follows that $X \in Post^*(L_q)$. Let $p_1 \cdots p_n \in S^*_\odot(q_X)$. Let then $X_1,\ldots,X_n$ such that $p_j = q_{X_j}$, or $p_j = (q_{X_j},P)$ (this is due to the definition of $S_\odot$). Then by the definition of $S_\odot$, it follows that $\odot(X_1,\ldots,X_n) \in {R'_\odot}^*(X)$, i.e. $\odot(X_1,\ldots,X_n) \in Post^*(X)$. We are going to show the property by induction on the number l of times there is a state of the form $p_j = (q_{X_j},-)$ or $p_j = (q_{X_j},||)$ in $p_1 \cdots p_n$:
   - $l = 0$. The property holds since $L_{q_{X_j}} = \{X_j\}$,

- $l > 0$. There are two cases:
  (a) $p_1 = (q_{X_1}, ||)$ or $p_1 = (q_{X_1}, -)$. Then necessarily $q_{X_1} \odot p_2 \cdots p_n \in S_\odot^*(q_X)$ and $p_1 \cdots p_n \in S_\odot(q_{X_1} \odot p_2 \cdots p_n)$. Since the number of states of the form $p_j = (q_{X_j}, -)$ or $p_j = (q_{X_j}, ||)$ in $q_{X_1} \odot p_2 \cdots p_n$ is $l-1$, it follows by induction that for each $t_2, \ldots, t_n$ s.t. $t_j \in Post^*(L_{q_{X_j}})$ if $p_j \in \{(q_{X_j}, -), (q_{X_j}, ||)\}$, and $t_j \in L_{q_{X_j}} = \{X_j\}$ if $p_j = q_{X_j}$; we have $\odot(X_1, t_2, \ldots, t_n) \in Post^*(X)$. Therefore, for every term $t_1 \in Post^*(L_{q_{X_1}})$ $(L_{q_{X_1}} = \{X_1\})$,

$$\odot(t_1, t_2, \ldots, t_n) \in Post^*\big(\odot(X_1, t_2, \ldots, t_n)\big) \subseteq Post^*(X) \subseteq Post^*(L_q).$$

  (b) There exists an index $m > 1$ such that $p_1 \cdots p_{m-1} \in Q_R^*$, and $p_m \in \{(q_{X_m}, -), (q_{X_m}, ||)\}$. Then by the definition of $S_\odot$ it follows that $p_1 \cdots p_{m-1} = q_{X_1} \cdots q_{X_{m-1}} \in S_\odot^*(0)$, and that $p_1 \cdots p_n \in S_\odot^*(p_m \cdots p_n) \subseteq S_\odot^*(q_X)$. And from the previous case it follows that for every $t_m, \ldots, t_n$ such that $t_j \in Post^*(L_{q_{X_j}})$ if $p_j = (q_{X_j}, P)$, and $t_j \in L_{q_{X_j}} = \{X_j\}$ if $p_j = q_{X_j}$, we have $\odot(t_m, \ldots, t_n) \in Post^*(X)$. Therefore

$$\odot(X_1, \ldots, X_{m-1}, t_m, \ldots, t_n) \in Post^*(X) \subseteq Post^*(L_q).$$

  This shows that the property holds since for $j < m$, $L_{p_j} = L_{q_{X_j}} = \{X_j\}$.

2. $\odot(L) \to (q, \odot)$ is the consequence of a β_5-rule. Let $q_1 \cdots q_n \in L$, we are going to show that for every $t_1, \ldots, t_n$ such that $t_j \in Post^*(L_{s_j})$ if $q_j \in \{(s_j, -), (s_j, ||)\}$, and $t_j \in L_{q_j}$ if $q_j \in Q \cup Q_R$, we have $\odot(t_1, \ldots, t_n) \in Post^*(L_q)$.
Let then $\odot(L') \to q$ be a rule of Δ such that $L = {S'_{s_{i-1}}}^*\big(\alpha'(L')\big)$, where α' is the character-substitution defined by

$$\alpha'(s) = \{s\} \cup \{q_X \mid X \xrightarrow{*}_\Delta s\},$$

and $S'_{s_{i-1}}$ is the following set of seq-rules: $S'_{s_{i-1}} = S_\odot \cup \{s \to (s, P)\} \cup \{s \to q_X \mid X \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (s, -)\} \cup \{s \to 0 \mid 0 \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (s, 0)\}$.
Since $\odot(L') \to q$ is a rule of Δ, it follows that for every $u_1, \ldots, u_g$ such that $u_j \in L_{p_j}$ where $p_1 \cdots p_g \in \alpha'(L')$, we have (due to the definition of α') $\odot(u_1, \ldots, u_g) \in L_q$.
Suppose $q_1 \cdots q_n \in {S'_{s_{i-1}}}^k\big(\alpha'(L')\big)$. We proceed by induction on k:
  - $k = 0$. Then $q_1 \cdots q_n \in \big(\alpha'(L')\big)$. It follows from the previous remark that for every $u_1, \ldots, u_n$ s.t. $u_j \in L_{q_j}$, we have $\odot(u_1, \ldots, u_n) \in L_q \subseteq Post^*(L_q)$, and the property holds.
  - $k > 0$. Then $q_1 \cdots q_n \in {S'_{s_{i-1}}}^k\big(\alpha'(L')\big)$. Let $p_1, \ldots, p_h$ such that

$$q_1 \cdots q_n \in S'_{s_{i-1}}(p_1 \cdots p_h) \text{ and } p_1 \cdots p_h \in {S'_{s_{i-1}}}^{k-1}\big(\alpha'(L')\big)$$

  Then by induction on k it follows that for every $v_1, \ldots, v_h$ such that $v_j \in Post^*(L_{s_j})$ if $p_j \in \{(s_j, -), (s_j, ||)\}$, and $v_j \in L_{p_j}$ if $p_j \in Q \cup Q_R$, we have

$$\odot(v_1, \ldots, v_h) \in Post^*(L_q). \tag{3}$$

  Consider such v_j's. Now since $q_1 \cdots q_n \in S_\odot(p_1 \cdots p_h)$, it follows by the definition of the prefix rewriting rules $S'_{s_{i-1}}$ that there are several cases depending on the rule of $S'_{s_{i-1}}$ which has been applied:

(a) There exists a state $s \in Q \cup Q_R$ such that $p_1 = s$ and $q_1 \cdots q_n$ is $(s,-)p_2 \cdots p_h$ or $(s,||)p_2 \cdots p_h$. This means that $n = h$, and $q_j = p_j$ for $j > 1$ (one of the rule $p_1 \to (p_1,-)$ or $p_1 \to (p_1,||)$ has been applied).

(b) There exist a state $s \in Q \cup Q_R$ and a process variable X such that $p_1 = s$ and $q_1 \cdots q_n$ is $q_X p_2 \cdots p_h$. This means that $X \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (s,-)$, $n = h$, and $q_j = p_j$ for $j > 1$ (the rule $p_1 \to q_X$ has been applied).

(c) There exist a state $s \in Q \cup Q_R$ and a process variable X such that $p_1 = s$ and $q_1 \cdots q_n$ is $p_2 \cdots p_h$. This means that $0 \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (s,0)$, $n = h-1$, and $q_j = p_{j+1}$ for $j \geq 1$ (the rule $p_1 \to 0$ has been applied).

(d) $p_1 \cdots p_m = q_{X_1} \cdots q_{X_m}$, it follows that $v_1 = X_1, \ldots, v_m = X_m$, i.e., we have from (3) that

$$\odot(X_1, \ldots, X_m, v_{m+1}, \ldots, v_h) \in Post^*(L_q).$$

Since $q_{X_1} \cdots q_{X_m} \to q_{Y_1} \cdots q_{Y_l} \in S_{\odot}$, it follows that $X_1 \cdots X_m \to Y_1 \cdots Y_l \in \mathcal{R}'_{\odot}$, and therefore that $Y_1 \cdots Y_l \in Post^*(X_1 \cdots X_m)$. Consequently, $\odot(Y_1, \ldots, Y_l, v_{m+1}, \ldots, v_h) \in Post^*\big(\odot(X_1, \ldots, X_m, v_{m+1}, \ldots, v_h)\big) \subseteq Post^*(L_q)$.

Let us consider for example the cases (a), (b), and (d), the other case being similar:

- The case (a): $q_1 \cdots q_n$ is $(p_1,-)p_2 \cdots p_h$ or $(p_1,||)p_2 \cdots p_h$, and $p_1 \in Q \cup Q_R$. Let then $v'_1 \in Post^*(L_{p_1})$, and $v_1 \in L_{p_1}$ such that $v'_1 \in Post^*(v_1)$. Then from (3), we have $\odot(v_1, v_2 \ldots, v_h) \in Post^*(L_q)$, and therefore

$$\odot(v'_1, v_2 \ldots, v_h) \in Post^*(L_q).$$

- The case (b): $q_1 \cdots q_n$ is $q_X p_2 \cdots p_h$ such that $X \xrightarrow{*}_{\widetilde{\Delta}_{i-1}} (p_1,-)$, and $p_1 \in Q \cup Q_R$. This means by induction on i that $X \in Post^*(L_{p_1})$. Let then $v_1 \in L_{p_1}$ such that $X \in Post^*(v_1)$. Then from (3), we have $\odot(v_1, v_2 \ldots, v_h) \in Post^*(L_q)$, and therefore

$$\odot(X, v_2 \ldots, v_h) \in Post^*(L_q).$$

- The case (d): $p_1 \cdots p_m = q_{X_1} \cdots q_{X_m}$, it follows that $v_1 = X_1, \ldots, v_m = X_m$, i.e. we have from (3) that

$$\odot(X_1, \ldots, X_m, v_{m+1}, \ldots, v_h) \in Post^*(L_q).$$

Since $q_{X_1} \cdots q_{X_m} \to (q_Y, P) \in S_{\odot}$, it follows that $X_1 \cdots X_m \to Y \in R'_{\odot}$, and therefore that $Y \in Post^*(X_1 \cdots X_m)$. Hence for every $t \in Post^*(Y)$, $\odot(t, v_{m+1}, \ldots, v_h) \in Post^*\big(\odot(X_1, \ldots, X_m, v_{m+1}, \ldots, v_h)\big) \subseteq Post^*(L_q)$. This shows that the property holds.

$\star$ For the property (e), suppose that $||(\varphi) \to (q,||) \in \widetilde{\Delta}_i \setminus \widetilde{\Delta}_{i-1}$. The proof is similar to the previous one and even simpler since we do not have to impose that the rewritings occur only on the top of the stack. Rules β_4 play the same role as in the "$\odot$"-case, and rules β_5 are analogous to rules β_5.

Now let us show the properties (a), (b), and (c). The proof is very tedious and there are many different cases to consider. Here we only deal with the most relevent cases. Let q be a state of Q', and let $v \xrightarrow{*}_{\widetilde{\Delta}_i} q$ be a computation of $\widetilde{\Delta}_i$ on the term v. Let n be the number of times a transition in $\widetilde{\Delta}_i \setminus \widetilde{\Delta}_{i-1}$ is applied during this computation. In such a case, we denote the previous computation by $v \xrightarrow{*}{}^{n}_{\widetilde{\Delta}_i} q$. We proceed by induction on n:

• $n = 0$, then only the rules of $\widetilde{\Delta}_{i-1}$ are applied, and the result follows by induction on i,

• $n > 0$. There are three cases:

1. There exist a context C and a term $v' \in T_P$ such that $v = C[v']$, and

$$v = C[v'] \xrightarrow{*}{}^{n-1}_{\widetilde{\Delta}_i} C[s'] \rightarrow_{\widetilde{\Delta}_i} C[s] \xrightarrow{k}_{\widetilde{\Delta}_{i-1}} q \tag{4}$$

   We prove the property by induction on k:
   - $k = 0$. Then C is the trivial context, $v = v'$, and $q = s$. There are different cases depending on the nature of the states s and s'. For example, if there are two states p and p' in $Q \cup Q_R$ such that $s' = (p', -)$, and $s = (p, -)$, then by induction on n we get that

$$v' \in Post^*(L_{p'}) \text{ and } v' \in Const \tag{5}$$

     Then the fact that $C[(p', -)] \xrightarrow{1}_{\widetilde{\Delta}_i} C[(p, -)]$ implies that $(p', -) \rightarrow_{\widetilde{\Delta}_i} (p, -)$ is a new rule, which can be either a (β_3)-, or a (β_4)-rule. In both cases, it is easy to show that $v' \in Post^*(L_p)$. For example, assume that $(p', -) \rightarrow_{\widetilde{\Delta}_i} (p, -)$ is a (β_3)-rule. In such a case, there is in $\widetilde{\Delta}_{i-1}$ a rule of the form $\odot(L) \rightarrow (p, \odot)$ such that $\{(p', -), (p', ||)\} \subseteq L$. We get from property (d) that for every $t \in Post^*(L_{p'}) \cap Const$, $\odot(t) \in Post^*(L_p)$, which means that $t \in Post^*(L_p)$. This means that $Post^*(L_{p'}) \cap Const \subseteq Post^*(L_p)$, and we get $v \in Post^*(L_p)$. The property is then satisfied.
   - $k > 0$. There are three cases:
     (a) $v \xrightarrow{*}_{\widetilde{\Delta}_i} C[s] \xrightarrow{k-1}_{\widetilde{\Delta}_{i-1}} q' \xrightarrow{1}_{\widetilde{\Delta}_{i-1}} q$. Suppose for example that $q' = (q'_0, ||)$ and $q = (q_0, ||)$, the other cases are similar. Then by induction on k we get that $v \in Post^*(L_{q'_0})$, and $root(v) = ||$. Then as previously, since $(q'_0, ||) \xrightarrow{1}_{\widetilde{\Delta}_{i-1}} (q_0, ||)$ we get that $v \in Post^*(L_{q_0})$.
     (b) $q = (q_0, \odot)$, and there exist l terms $v_1, \dots, v_l$ such that $v = \odot(v_1, \dots, v_l)$. Since $s \notin Q \cup Q_R$ ($n > 0$), suppose that $s = (p, -)$, or $s = (p, ||)$, for some $p \in Q$. Let then a context C' such that $C[] = \odot(v_1, \dots, v_{m-1}, C'[], v_{m+1}, \dots, v_l)$, and l states $q_1, \dots, q_l \in Q'$ such that

$$v = \odot(v_1, \dots, v_l) \xrightarrow{*}_{\widetilde{\Delta}_i} C[s] = \odot(v_1, \dots, v_{m-1}, C'[s], v_{m+1}, \dots, v_l) \xrightarrow{k-1}_{\widetilde{\Delta}_{i-1}}$$

$$\odot(q_1, \dots, q_l) \xrightarrow{1}_{\widetilde{\Delta}_{i-1}} (q_0, \odot).$$

     Let $p_1, \dots, p_l \in Q \cup Q_R$ be such that $q_j = p_j$ or $q_j \in (p_j, P)$. Then by induction on k we get that $v_m \in Post^*(L_{p_m})$, and $root(v_m) \neq \odot$. Moreover, by induction on i it follows that for $j \neq m$, $v_j \in L_{p_j}$ if $p_j \in Q_{||} \cup Q_{-}$, and $v_j \in Post^*(L_{p_j})$ otherwise, and $root(v_j) \neq \odot$. Then the property holds thanks to property (d), since we get that $v \in Post^*(L_{q_0})$.

(c) $q = (q_0, ||)$, and there exist l terms $v_1, \ldots, v_l$ such that $v = ||(v_1, \ldots, v_l)$, a context C' such that

$$C[] = ||(v_1, \ldots, v_{m-1}, C'[], v_{m+1}, \ldots, v_l),$$

l states $q_1, \ldots, q_l$ in Q' such that

$$v = ||(v_1, \ldots, v_l) \xrightarrow{*}_{\tilde{\Delta}_i} C[s] = ||(v_1, \ldots, v_{m-1}, C'[s], v_{m+1}, \ldots, v_l) \xrightarrow{k-1}_{\tilde{\Delta}_{i-1}}$$

$$||(q_1, \ldots, q_l) \xrightarrow{1}_{\tilde{\Delta}_{i-1}} (q_0, ||)$$

This case is similar to the previous one. We use property (e) instead of property (d).

2. There exists a context C and an operator $f \in \{\odot, ||\}$ such that $v = C[f(v_1, \ldots, v_l)]$, and $v = C[f(v_1, \ldots, v_l)] \xrightarrow{*}{}^{n-1}_{\tilde{\Delta}_i} C[f(p_1, \ldots, p_l)] \rightarrow_{\tilde{\Delta}_i} C[s] \xrightarrow{k}_{\tilde{\Delta}_{i-1}} q$ such that there exists $n_1 < n, \ldots, n_l < n$ such that $\sum_{1 \leq i \leq l} n_i = n-1$, and

$$v_j \xrightarrow{*}{}^{n_j}_{\tilde{\Delta}_i} p_j$$

Let $p_j = (q_j, \sim)$, where $\sim \in \{-, \odot, ||\}$. Then necessarily by induction on n it follows that for every j, $1 \leq j \leq l$,

$$v_j \in Post^*(L_{q_j}), \tag{6}$$

Let $s = (p, \sim)$. According to the nature of the operator f, we can use property (d) or (e) to show that $f(v_1, \ldots, v_m) \in Post^*(L_p)$. Then we can show as previously by induction on k that $v \in Post^*(L_q)$.

3. There exists a context C such that $v = C[v_1, \ldots, v_m]$, and

$$v = C[v_1, \ldots, v_m] \xrightarrow{*}{}^{n}_{\tilde{\Delta}_i} C[p_1, \ldots, p_m] \xrightarrow{k}_{\tilde{\Delta}_{i-1}} q$$

such that there exists $n_1 < n, \ldots, n_m < n$ such that $\sum_{1 \leq i \leq m} n_i = n$, and

$$v_j \xrightarrow{*}{}^{n_j}_{\tilde{\Delta}_i} p_j$$

Let $p_j = (q_j, \sim)$. Then necessarily by induction on n it follows that for every j, $1 \leq j \leq m$,

$$v_j \in Post^*(L_{q_j}), \tag{7}$$

Then we can show as previously by induction on k that $v \in Post^*(L_q)$.

Lemma 4. *For every $q \in Q \cup Q_R$, and every $v \in T_p$, we have:*

- $v \in Post^*(L_q)$*, and* $root(v) = \omega$*, then* $v \in L_{(q,\omega)}$*, for* $\omega \in \{\odot, ||\}$*,*
- $v \in Post^*(L_q) \cap Const$*, then* $v \in L_{(q,-)}$*,*
- $v \in Post^*(L_q)$*, and* $v = 0$*, then* $v \in L_{(q,0)}$*.*

Proof. We are going to show by induction on k that:

- $v \in Post^k(L_q)$, and $root(v) = ||$, then $v \in L_{(q,||)}$,
- $v \in Post^k(L_q)$, and $root(v) = \odot$, then $v \in L_{(q,\odot)}$,
- $v \in Post^k(L_q) \cap Const$, then $v \in L_{(q,-)}$,
- $v \in Post^k(L_q)$, and $v = 0$, then $v \in L_{(q,0)}$.

For a state p, showing that $v \in L_p$ amounts to showing that $v \xrightarrow{*}_{\tilde{\Delta}} p$. Let us start the induction:

- $k = 0$. Then the property holds thanks to rules (β_1).
- $k > 0$. Let then $v \in Post^k(L_q)$, and $v' \in Post^{k-1}(L_q)$ such that $v \in Post(v')$. Assume that $root(v) = ||$, the other cases can be treated in a similar way. Then there are three cases:
  1. $v' \in Const$, and v is obtained from v' by applying a paral-rule. Let then X be the process variable such that $v' = X$. Then by induction we get that $X \in L_{(q,-)}$. Rules $(\beta_4 a)$ imply that $||(\varphi) \to (q, ||)$, where $\varphi = S^*_{||}(q_X)$. Since v is obtained from X by applying one rule from $R'_{||}$, it follows that $v = ||(X_1, \ldots, X_n)$ where $Parikh(q_{X_1} \cdots q_{X_n}) \models S_{||}(q_X)$, i.e. $Parikh(q_{X_1} \cdots q_{X_n}) \models \varphi$. Therefore, we get that $v \in L_{(q,||)}$.
  2. $v' = 0$, and v is obtained from v' by applying a paral-rule. Then by induction we get that $0 \in L_{(q,0)}$. Rules $(\beta_4 b)$ imply that $||(\varphi) \to (q, ||)$, where $\varphi = S^*_{||}(0)$. Since v is obtained from 0 by applying one rule from $R'_{||}$, it follows that $v = ||(X_1, \ldots, X_n)$ where $Parikh(q_{X_1} \cdots q_{X_n}) \models S_{||}(0)$, i.e. $Parikh(q_{X_1} \cdots q_{X_n}) \models \varphi$. Therefore, we get that $v \in L_{(q,||)}$.
  3. $root(v') = ||$. Then by induction it follows that $v' \in L_{(q,||)}$. We want to show that v also is in $L_{(q,||)}$. There are many cases depending on the type of the rule which has been applied:
     - A paral-rule of the form
       $$X_1 || \cdots || X_n \to X'_1 || \cdots || X'_m$$
       has been applied, and has rewritten $X_1, \ldots, X_n$, into $X'_1, \ldots, X'_m$. Then, there exists a context C such that
       $$v' = C[||(X_1, \ldots, X_n, v_1, \ldots, v_l)]$$
       and
       $$v = C[||(X'_1, \ldots, X'_m, v_1, \ldots, v_l)]$$
       There are two cases depending on the values of m and l:

(a) $m > 1$ or $l > 1$. Let then $p_1, \ldots, p_n$, $p'_1, \ldots, p'_l$, and $p \in Q \cup Q_R$ be such that $||(X_1, \ldots, X_n, v_1, \ldots, v_l) \xrightarrow{*}_{\widetilde{\Delta}} ||(p_1, \ldots, p_n, p'_1, \ldots, p'_l) \rightarrow_{\widetilde{\Delta}} p$ or $||(X_1, \ldots, X_n, v_1, \ldots, v_l) \xrightarrow{*}_{\widetilde{\Delta}} ||(p_1, \ldots, p_n, p'_1, \ldots, p'_l) \rightarrow_{\widetilde{\Delta}} (p, ||)$
• Suppose that the second case holds. This means that

$$v' = C[||(X_1, \ldots, X_n, v_1, \ldots, v_l)] \xrightarrow{*}_{\widetilde{\Delta}} C[(p, ||)] \xrightarrow{*}_{\widetilde{\Delta}} (q, ||)$$

Let then φ be a formula such that $||(\varphi) \rightarrow (p, ||)$ is a rule of $\widetilde{\Delta}$, and

$$Parikh(p_1 \cdots p_n, p'_1 \cdots p'_l) \models \varphi.$$

¿From Lemma 5, it follows that $Parikh(q_{X'_1} \cdots q_{X'_m} p'_1 \cdots p'_l) \models \varphi$, and hence $||(X'_1, \ldots, X'_m, v_1, \ldots, v_l) \xrightarrow{*}_{\widetilde{\Delta}} ||(q_{X'_1}, \ldots, q_{X'_m}, p'_1, \ldots, p'_l) \rightarrow_{\widetilde{\Delta}} (p, ||)$ which means that $v = C[||(X'_1, \ldots, X'_m, v_1, \ldots, v_l)] \xrightarrow{*}_{\widetilde{\Delta}} C[(p, ||)] \xrightarrow{*}_{\widetilde{\Delta}} (q, ||)$.
• Now suppose that $||(X_1, \ldots, X_n, v_1, \ldots, v_l) \xrightarrow{*}_{\widetilde{\Delta}} ||(p_1, \ldots, p_n, p'_1, \ldots, p'_l) \rightarrow_{\widetilde{\Delta}} p$ Then we get also by the same way that $v = C[||(X'_1, \ldots, X'_m, v_1, \ldots, v_l)] \xrightarrow{*}_{\widetilde{\Delta}} C[(p, ||)]$ and now we can show by structural induction on C that $C[(p, ||)] \xrightarrow{*}_{\widetilde{\Delta}} (q, ||)$.

(b) $m = 1$, and $l = 0$, which means that the paral-rule that has been applied is of the form $X_1 || \cdots || X_n \rightarrow X$ (or $X_1 || \cdots || X_n \rightarrow 0$). Let then C' be the context such that $v' = C'[\cdot(||(X_1, \ldots, X_n), u_1, \ldots, u_j)]$ and $v = C'[\odot(X, u_1, \ldots, u_j)]$. Let $v' = C'[\odot(||(X_1, \ldots, X_n), u_1, \ldots, u_j)] \xrightarrow{*}_{\widetilde{\Delta}} C'[\odot(\tilde{p}, q_1, \ldots, q_j))] \xrightarrow{*}_{\widetilde{\Delta}} C'[\tilde{q}'] \xrightarrow{*}_{\widetilde{\Delta}} (q, ||)$, where $\tilde{p} \in \{p, (p, ||)\}$ for some $p \in Q \cup Q_R$, and $\tilde{q}' \in \{q', (q', \odot)\}$ for some $q' \in Q \cup Q_R$. Let us consider the case where $\tilde{q}' = (q', \odot)$, the other case can be solved by structural induction on C'. As previously, consider the rule of $\widetilde{\Delta}$, $||(\varphi) \rightarrow (p, ||)$ such that

$$Parikh(q_{X_1} \cdots q_{X_n}) \models \varphi.$$

Since $q_X \in S^*_{||}(||(q_{X_1}, \ldots, q_{X_n}))$, it follows from Lemma 5 that the vector $vecu_{q_X}$ satisfies φ, and hence, by rules (β_3), it follows that $(q_X, -) \rightarrow (p, -)$ is in $\widetilde{\Delta}$, and $X \rightarrow_{\widetilde{\Delta}} q_X \rightarrow_{\widetilde{\Delta}} (q_X, -) \rightarrow_{\widetilde{\Delta}} (p, -)$.
Furthermore, let $\odot(L) \rightarrow (q', \odot)$ be a rule of $\widetilde{\Delta}$ such that $(p, P) q_1 \cdots q_j \in L$. Then it follows that $(p, -) q_1 \cdots q_j \in L$, which means that $v = C'[\odot(X, u_1, \ldots, u_j)] \xrightarrow{*}_{\widetilde{\Delta}} C'[\odot((p, -), q_1, \ldots, q_j)] \rightarrow_{\widetilde{\Delta}} C'[(q', \odot)] \xrightarrow{*}_{\widetilde{\Delta}} (q, ||)$

This ends the proof of this case.

• A seq-rule of the form

$$X \rightarrow X_1 \cdots X_m$$

has been applied, and has rewritten X, into $\odot(X_1, \ldots, X_m)$. Then, there exists a context C such that

$$v' = C[||(X, v_1, \ldots, v_l)]$$

and

$$v = C[||\big(\odot(X_1,\ldots,X_m),v_1,\ldots,v_l\big)].$$

Then suppose as previously that $||(X,v_1,\ldots,v_l) \xrightarrow{*}_{\widetilde{\Delta}} ||(p_1,p'_1,\ldots,p'_l) \rightarrow_{\widetilde{\Delta}} (p,||)$ and that $p_1 = (p',-)$, or $p_1 = p'$ for some state p' in $Q \cup Q_R$ (the case where $||(X,v_1,\ldots,v_l) \xrightarrow{*}_{\widetilde{\Delta}} p$ can be solved as explained previously by structural induction on C). Then since $X \xrightarrow{*}_{\widetilde{\Delta}} (p',-)$, it follows by rules (β_4) that

$$\odot(q_{X_1},\ldots,q_{X_m}) \xrightarrow{*}_{\widetilde{\Delta}} (p',\odot)$$

since $q_{X_1}\cdots q_{X_n} \in S^*_{\odot}(q_X)$.
Furthermore, let $||(\varphi) \rightarrow (p,||)$ be a rule of $\widetilde{\Delta}$ such that

$$Parikh(p_1p'_1\cdots p'_l) \models \varphi.$$

Then it follows by rules (β_5) that $Parikh((p',\odot)p'_1\cdots p'_l) \models \varphi$, which means that

$$v = C[||\big(\odot(X_1,\ldots,X_m),v_1,\ldots,v_l\big)] \xrightarrow{*}_{\widetilde{\Delta}} C[||\big(\odot(q_{X'_1},\ldots,q_{X'_m}),p'_1,\ldots,p'_l\big)]$$

$$\xrightarrow{*}_{\widetilde{\Delta}} C[||((p',\odot),p'_1,\ldots,p'_l)] \rightarrow_{\widetilde{\Delta}} C[(p,||)] \xrightarrow{*}_{\widetilde{\Delta}} (q,||).$$

- The other cases where

$$v = C[\odot(X_1,\ldots,X_n,v_1,\ldots,v_l)]$$

and

$$v' = C[\odot(X'_1,\ldots,X'_m,v_1,\ldots,v_l)]$$

or

$$v = C[\odot(X,v_1,\ldots,v_l)]$$

and

$$v' = C[\odot\big(||(X_1,\ldots,X_n),v_1,\ldots,v_l\big)]$$

can be treated similarly since the inference rules for the sequential operator are analogous to the rules for the parallel operator.

Lemma 5. • *Let* $\odot(X_1,\ldots,X_m,u_1,\ldots,u_n) \xrightarrow{*}_{\widetilde{\Delta}} \odot(p_1,\ldots,p_m,p'_1,\ldots,p'_n)$, *where*

$$p_1\cdots p_m p'_1\cdots p'_n \in L$$

for L *s.t.* $\odot(L) \rightarrow (q,\odot)$ *is a rule in* $\widetilde{\Delta}$. *If* $X_1\cdots X_m \rightarrow Y_1\cdots Y_k \in \mathcal{R}_{\odot}$, *then* $q_{Y_1}\cdots q_{Y_k}p'_1\cdots p'_n \in L'$.
• *Let* $||(X_1,\ldots,X_m,u_1,\ldots,u_n) \xrightarrow{*}_{\widetilde{\Delta}} ||(p_1,\ldots,p_m,p'_1,\ldots,p'_n)$,*where*

$$Parikh(p_1\cdots p_m p'_1\cdots p'_n) \models \varphi$$

for φ *s.t.* $||(\varphi) \rightarrow (q,||)$ *is a rule of* $\widetilde{\Delta}$. *If* $X_1||\cdots||X_m \rightarrow Y_1||\cdots||Y_k \in \mathcal{R}_{||}$, *then*

$$Parikh(q_{Y_1}\cdots q_{Y_k}p'_1\cdots p'_n) \models \varphi$$

Proof. We prove only the first statement, the second one can be shown in a similar way. We deal with the case where $X_1 \cdots X_m \to Y \odot Z \in R_\odot$.

There are two cases depending on how the rule $\odot(L) \to (q, \odot)$ has been created. Let us consider for example the case where it is the result of a (β_4)-rule. Let then $A \in Const$ such that $A \xrightarrow{*}_{\tilde{\Delta}} (q, -)$, and $L = S^*_\odot(q_A)$. We show hereafter that $q_{Y_1} \cdots q_{Y_k} p'_1 \cdots p'_n \in L$. We proceed by induction on the number K of indices i in $\{1, \ldots, m\}$ such that $p_i \neq q_{X_i}$:

1. $K = 0$, then $p_i = q_{X_i}$ for every i, $1 \leq i \leq m$. Then since $q_{Y_1} \cdots q_{Y_k} \in S_\odot(q_{X_1} \cdots q_{X_m})$, it is clear from the definition of L that $q_{Y_1} \cdots q_{Y_k} p'_1 \cdots p'_n \in L$.
2. $K > 0$. Let then i be the index such that $p_i = (q_{F_i}, -)$ for some process variable F_i different from X_i, and for every j, $1 \leq j < i$, $p_j = q_{X_j}$. Then necessarily, due to the prefix rewriting rules $S_\odot$, it follows that $(q_{F_i}, -) p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$, and $q_{X_1} \cdots q_{X_{i-1}} \in S^*_\odot(0)$. Therefore, from the definition of $S_\odot$, we get that $q_{F_i} p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$. Moreover, since $X_i \xrightarrow{*}_{\tilde{\Delta}} (q_{F_i}, -)$, it follows from Lemma (3) that $X_i \in Post^*(F_i)$, i.e. by the definition of $R'_\odot$ and $R'_{||}$ that $X_i \in {R'_\odot}^*(F_i)$ (and $X_i \in {R'_{||}}^*(F_i)$). It follows that $q_{X_i} \in {S_\odot}^*(q_{F_i})$ (note that it is important here to have $R'_\odot$ in the definition of $S_\odot$, $R_\odot$ does not suffice), and therefore that $q_{X_i} p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$. Since $q_{X_1} \cdots q_{X_{i-1}} \in S^*_\odot(0)$, we get that

$$q_{X_1} \cdots q_{X_{i-1}} q_{X_i} p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L,$$

   Now since the number of states $p_j \neq q_{X_j}$ (of the form $(s, -)$) in $q_{X_1} \cdots q_{X_{i-1}} q_{X_i} p_{i+1} \cdots p_m$ is less than K, we infer by induction that $q_{Y_1} \cdots q_{Y_k} p'_1 \cdots p'_n \in L$.

Now, let us consider the case where the rule $\odot(L) \to (q, \odot)$ is the result of a (β_5)-rule. We proceed exactly as previously. The only difference is that now, in the induction step, we have also to consider the case where there is an index i such that $p_i = (p, -)$ for some p in Q, and for every j, $1 \leq j < i$, $p_j = q_{X_j}$. Then necessarily, due to the prefix rewriting rules $S'_\odot$, it follows that $(p, -) p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$, and $q_{X_1} \cdots q_{X_{i-1}} \in S^*_\odot(0)$. Therefore, from the definition of $S'_\odot$, we get that $p p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$. Moreover, since $X_i \xrightarrow{*}_{\tilde{\Delta}} (p, -)$, it follows from the definition of $S'_\odot$ that $q_{X_i} \in {S'_\odot}^*(p)$, and therefore that $q_{X_i} p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L$. Since $q_{X_1} \cdots q_{X_{i-1}} \in S^*_\odot(0)$, we get that

$$q_{X_1} \cdots q_{X_{i-1}} q_{X_i} p_{i+1} \cdots p_m p'_1 \cdots p'_n \in L,$$

Now since the number of states $p_j \neq q_{X_j}$ (of the form $(s, -)$) in $q_{X_1} \cdots q_{X_{i-1}} q_{X_i} p_{i+1} \cdots p_m$ is less than K, we infer by induction that $q_{Y_1} \cdots q_{Y_k} p'_1 \cdots p'_n \in L$.