

# Forward Analysis of Dynamic Network of Pushdown Systems is Easier without Order

Denis Lugiez  
LIF UMR 6166  
Aix-Marseille Université- CNRS, FRANCE.  
*email:denis.lugiez@lif.univ-mrs.fr*

## Abstract

Dynamic networks of Pushdown Systems (*PDN* in short) have been introduced to perform static analysis of concurrent programs that may spawn threads dynamically. In this model the set of successors of a regular set of configurations can be non-regular, making forward analysis of these models difficult. We refine the model by adding the associative-commutative properties of parallel composition, and we define Presburger weighted tree automata, an extension of weighted automata and tree automata, that accept the set of successors of a regular set of configurations. This allows forward analysis of *PDN* since these automata have a decidable emptiness problem and are closed under intersection.

## Introduction

Dynamic networks of Pushdown Systems is a model of concurrent programs that models thread generation and it has been introduced for performing the static analysis of these programs [BMOT05]. This model follows from a stream of works that have advocated the use of automata techniques for the static analysis of programs for more and more complex problems (from intraprocedural analysis to interprocedural concurrent analysis [EP00]) and more and more complex models, from Pushdown system [AB97, RSJ03] to Process Algebra [LS98] and networks of Pushdown systems [BMOT05], possibly involving data structure or synchronization [KG07]. In [BMOT05], the authors consider pushdown processes that can generate new processes yielding configurations that are sets of ordered unranked trees and they prove that the set of predecessors of a regular set of configurations is regular where regularity refers to hedge automata, the standard extension of tree automata to unranked trees. However, the set of successors can be non-regular making forward analysis of such systems difficult. In this paper we enrich this model by assuming that parallel composition is an associative-commutative operation therefore the execution of threads generated by some pushdown system is independent of their order. This amounts to considering configurations that are *unranked unordered trees* and using the notion of regularity relying on Presburger automata [ZL06]. The main result of the paper is to prove that the set of successors of a regular set of configuration can be non-regular but is accepted by a Presburger weighted tree automaton, an

extension of Presburger automata which enjoy properties that allow to perform forward analysis. The regularity of the set of predecessors of a regular set is derived from the results of [BMOT05].

Section 1 presents the basic definitions while section 2 introduces pushdown systems and dynamic networks of pushdown systems. Regular sets are defined in section 4. Weighted word and tree automata are defined in section 5 and the computation of the set of successors is explained in section 6. Section 7 shows how to derive the computation of the set of predecessors using known results.

## 1 Analysis of Transition Systems

A system  $S$  is given by an (infinite) set of configurations  $\mathcal{C}$  and a transition relation  $\rightarrow_S$  between configurations. Configurations are formal objects (words, trees, ...) that describe the current state of the system, and the dynamic behavior of the systems is given by the relation  $\rightarrow_S$  which is usually defined by a finite set of transition rules  $R$ . The reflexive transitive closure of  $\rightarrow_S$  is denoted as  $\xrightarrow{*}_S$ .

The set of *successors* of a configuration  $c$  is the set  $Post_S^*(c) = \{c' \in \mathcal{C} \mid c \xrightarrow{*}_S c'\}$  and for  $L \subseteq \mathcal{C}$ ,  $Post_S^*(L) = \bigcup_{c \in L} Post_S^*(c)$ .

The set of *predecessors* of a configuration  $c$  is the set  $Pred_S^*(c) = \{c' \in \mathcal{C} \mid c' \xrightarrow{*}_S c\}$  and for  $L \subseteq \mathcal{C}$ ,  $Pred_S^*(L) = \bigcup_{c \in L} Pred_S^*(c)$ .

The set of all possible initial configurations  $Init$  and the set of all bad configurations  $Bad$  can be infinite and are usually defined as regular language for some appropriate notion of regularity. *Backward analysis* tests the emptiness of the set  $Init \cap Pred_S^*(Bad)$  when *Forward analysis* test the emptiness of the set  $Post_S^*(Init) \cap Bad$ . These analyzes allow to detect statically if the execution of the system  $S$  can lead to an error state. When the languages  $Init$  and  $Bad$  are regular for a good notion of regularity (that enjoys closure under boolean operations and decision of emptiness) the feasibility of these analysis boils down to proving that  $Pred_S^*(L)$  and  $Post_S^*(L)$  are regular when  $L$  is regular and providing effective constructions of devices accepting these sets. Forward analysis is more difficult than backward analysis, since regularity is easier to preserve under inverse image than under direct image, see [CDJ<sup>+</sup>99] for regular tree languages.

In the following, we drop the subscript  $S$  of  $\rightarrow_S, \dots$  when  $S$  is clear from the context

## 2 Pushdown Systems

A *pushdown system* (PDS in short, see [AB97] for details)  $P$  is a triple  $(Q, \Sigma, R)$  where  $Q$  is a finite set of states,  $\Sigma$  is a finite stack alphabet and  $R$  is a finite set of transition rules  $qa \rightarrow q'\gamma$  with  $q, q' \in Q, a \in \Sigma, \gamma \in \Sigma^*$  (also called rewrite rules in the following). The set  $\mathcal{C}$  of configurations is the set of words  $qw$  with  $q \in Q$  (the state),  $w \in \Sigma^*$  (the content of the stack). The transition relation  $\rightarrow$  between configurations is the relation defined by  $qw \rightarrow q'w'$  iff  $w = aw''$  and  $w' = \gamma.w''$  and there is a rule  $qa \rightarrow q'\gamma \in R$ . The relation  $\xrightarrow{*}$  is exactly the prefix rewriting relation defined by the set of rewrite rules  $R$ .

Let  $P$  be a pushdown system, let  $L \subseteq \mathcal{C}$  be a regular language. Then  $Pred^*(L)$  and  $Post^*(L)$  are regular languages, see [Buc64, AB97]. Actually the

relation  $R$  on pairs of configurations defined by  $qw \mathcal{R} q'w'$  iff  $qw \xrightarrow{*} q'w'$  is a rational relation (see [Cau00] for extensions).

The following proposition states that the successors of a regular set of configurations  $L$  is the set of successors of a unique configuration, provided that (i) we extend the initial PDS with new states and new rules (ii) that we keep only the configurations that corresponds to the initial alphabet.

**Proposition 1** *Let  $P = (Q, \Sigma, R)$  be a PDS and  $L \subseteq C$  be a regular language. There exists  $P' = (Q \cup Q', \Sigma \cup \{\$\}, R \cup R')$  such that  $Post_P^*(L) = Post_{P'}^*(q_0\$) \cap C$  where  $q_0 \in Q'$ .*

### 3 Dynamic Network of Pushdown Systems

Dynamic networks of pushdown processes [BMOT05] generalize PDS since (i) a configuration may have several PDS running in parallel, (ii) a transition rule of a PDS not only changes the state and stack of the PDS, but may also spawn one (or more) new PDS which is a son of the process. There is no limitation in the creation of processes (a process has an arbitrary number of sons) and in the recursion depth for process creation (each process may create sons that can create sons, ...). Therefore configurations are isomorphic to unranked tree-like structures. In this work, we enrich the original model by assuming that parallel composition is associative-commutative, hence trees are also unordered.

#### 3.1 Configurations

The set  $PDN$  of configurations and the set  $PDN_{\parallel}$  of parallel configurations are defined by:

$$\begin{aligned} PDN \ni c & ::= qw & | & \quad qw(c_{\parallel}) & q \in Q, w \in \Sigma^* \\ PDN_{\parallel} \ni c_{\parallel} & ::= c_1 \parallel \dots \parallel c_n & & & n \geq 1, \forall i = 1, \dots, n \ c_i \in PDN \end{aligned}$$

The parallel composition is independent of the order of its arguments and the equality  $\equiv$  between configurations is defined by:

$$\begin{aligned} qw \equiv q'w' & \text{ iff } q = q' \text{ and } w = w' \\ qw(c_1 \parallel \dots \parallel c_n) \equiv q'w'(c'_1 \parallel \dots \parallel c'_m) & \text{ iff } q = q', w = w', n = m \text{ and } \exists \sigma \\ & \text{ permutation of } \{1, \dots, n\} \\ & \text{ such that } c_i \equiv c'_{\sigma(i)} \end{aligned}$$

The set  $Sub(t)$  of process subterms of a configuration  $t$  is defined by:

$$\begin{aligned} Sub(qw) &= \{qw\} \\ Sub(qw(t_1 \parallel \dots \parallel t_n)) &= \{qw(t_1 \parallel \dots \parallel t_n)\} \cup \{qw\} \cup \bigcup_{i=1, \dots, n} Sub(t_i) \end{aligned}$$

A context  $C[\square]$  is a configuration  $t$  where some process subterm is replaced by the symbol  $\square$ . The notation  $C[s]$  denotes the configuration obtained by replacing  $\square$  by  $s \in PDN$ .

**Example 1** *Let  $c = qa(qaa \parallel q'b \parallel q(qa \parallel q'b))$ . We can draw this configuration as sets of vertical lines (each one being a PDS) combined in a tree-like structure:*

$$\begin{array}{ccccccc} & & q & & & & \\ & & a & & & & \\ q & \parallel & q' & \parallel & & q & \\ a & & b & & q & \parallel & q' \\ a & & & & a & & b \end{array}$$

We have  $Sub(c) = \{q, qa, qaa, q'b, qb\}$  and  $c \equiv qa(q'b \parallel q(q'b \parallel qa) \parallel qaa)$ .

### 3.2 PDN and their Transition Relation

A *dynamic network of pushdown systems* (PDN in short)  $P$  is a triple  $(Q, \Sigma, R)$  where  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet and  $R$  is a finite set of rules of the form  $qa \rightarrow q'\gamma$  or  $qa \rightarrow q'\gamma(q_1\gamma_1 \parallel \dots \parallel q_n\gamma_n)$  where  $w, \gamma, \gamma_i$  for  $i = 1, \dots, n$  belong to  $\Sigma^*$ ,  $q, q', q_i$  for  $i = 1, \dots, n$  belong to  $Q$ .

The relation  $\rightarrow$  induced by a PDN  $P$  on pairs of configurations is defined by:

- if  $qa \rightarrow q'\gamma \in R$  then  $qaw \rightarrow q'\gamma w$
- if  $qa \rightarrow q'\gamma(q_1\gamma_1 \parallel \dots \parallel q_n\gamma_n) \in R$  then
  - $qaw \rightarrow q'\gamma w(q_1\gamma_1 \parallel \dots \parallel q_n\gamma_n)$
  - $qaw(s_1 \parallel \dots \parallel s_m) \rightarrow q'\gamma w(q_1\gamma_1 \parallel \dots \parallel q_n\gamma_n \parallel s_1 \dots \parallel s_m)$
- if  $c \rightarrow c'$  and  $c \in Sub(t)$ , then  $t = C[c] \rightarrow t' = C[c']$ ,
- if  $c \rightarrow c'$  and  $\bar{c} \equiv c, \bar{c}' \equiv c'$  then  $\bar{c} \rightarrow \bar{c}'$ .

**Example 2** Let  $P = (\{q, q'\}, \{a, b\}, \{qa \rightarrow q'bb, q'b \rightarrow qab(qa \parallel qa)\})$  be a PDN. A sequence of transition of  $P$  is:

$$\begin{aligned} qa \rightarrow q'bb &\rightarrow qabb(qa \parallel qa) \rightarrow qabb(qa \parallel q'bb) \rightarrow q'bbbb(qa \parallel q'bb) \\ &\rightarrow qabbbb(qa \parallel qa \parallel qa \parallel q'bb) \rightarrow qabbbb(qa \parallel qa \parallel qa \parallel qabb(qa \parallel qa)) \end{aligned}$$

The next proposition states that a transition sequence can be done by applying transition rules to the top PDS first and then on the arguments of parallel compositions.

**Proposition 2**  $qw(t_1 \parallel \dots \parallel t_n) \xrightarrow{*} q'w'(u_1 \parallel \dots \parallel u_m)$  iff  $qw(t_1 \parallel \dots \parallel t_n) \xrightarrow{*} q'w'(t_1 \parallel \dots \parallel t_n \parallel s_{n+1} \parallel \dots \parallel s_m) \xrightarrow{*} q'w'(u_1 \parallel \dots \parallel u_m)$

## 4 Regular Sets of PDN Configurations

**Configurations as Unranked-Unordered Trees** A configuration is isomorphic to a unranked-unordered tree on the signature  $\{\#\} \cup Q \cup \Sigma \cup \{\|\}$  where  $\#$  is a constant, each symbol in  $Q$  or  $\Sigma$  is now a monadic symbol. The symbol  $\parallel$  is a permutative variadic symbol. The operations  $c2t$  and  $t2c$  that transform a configuration  $c$  into a tree  $t$  and conversely are defined as follows:

$$\begin{aligned} c2t(qa_1 \dots a_n) &= q(a_1(\dots (a_n(\#)))) \\ c2t(qa_1 \dots a_n(c_1 \parallel \dots \parallel c_n)) &= q(a_1(\dots (a_n(\parallel (c2t(c_1), \dots, c2t(c_n)))))) \\ t2c(q(a_1(\dots (a_n(\#)))) &= qa_1 \dots a_n \\ t2c(q(a_1(\dots (a_n(\parallel (c_1, \dots, c_n)))))) &= qa_1 \dots a_n(t2c(c_1) \parallel \dots \parallel t2c(c_n)) \end{aligned}$$

Since the transformation is one-to-one, results and definitions stated for configurations are valid for trees. The set of nodes of a tree  $t$  is denoted by  $Nodes(t)$ . A symbol of  $\{\#\} \cup Q \cup \Sigma \cup \{\|\}$  is attached to each node  $N$  of  $t$ .

**Top-down Presburger Automata** Presburger arithmetic is the first-order theory of  $\mathbb{N}, +, =$  and a Presburger formula is a formula of this theory. For instance  $\exists z x = y + 2z + 1$  is a Presburger formula in the free variables  $x, y$ . This theory is decidable [Pre29].

Word regular languages and tree regular languages have been generalized for unranked-unordered trees using Presburger automata [ZL06]. To fit the framework of PDN, we slightly change the definitions and we use *top-down automata* instead of bottom-up automata.

**Definition 1** A top-down Presburger automaton is a tuple  $\mathcal{A} = (S, S_I, R \cup R')$  where:

- $S = \{s_1, \dots, s_p\}$  is a finite ordered set of states,
- $S_I \subseteq S$  is a set of initial states,
- $R$  is a set of rules of the form  $s \rightarrow \#$  or  $s \xrightarrow{a} s'$  where  $s, s' \in S, a \in Q \cup \Sigma$ ,
- $R'$  is a set of rules  $s \xrightarrow{\parallel} \phi(x_1, \dots, x_p)$  where  $\phi(x_1, \dots, x_p)$  is a Presburger formula.

A run of the automaton  $\mathcal{A}$  on a tree  $t$  is a labelling  $r : Nodes(t) \rightarrow S$  of the nodes of  $t$  by the states of  $S$  such that:

- if  $\#$  is the symbol of a node  $N$  of  $t$ , the node  $N$  is labelled by  $r(N) = s$  such that the rule  $s \rightarrow \#$  belongs to  $R$ ,
- if  $a \in Q \cup \Sigma$  is the symbol of a node  $N$  of  $t$ , and  $N$  is labelled by  $r(N) = s$ , then the unique son of  $N$  is labelled by  $s'$  such that the rule  $s, a \rightarrow s'$  belongs to  $R$ ,
- if  $\parallel$  is the symbol of a node  $N$  of  $t$  and  $N$  is labelled by  $r(N) = s$ , if  $N_1, \dots, N_m$  are the sons of  $N$ , then
  - (i) there are  $n_1$  sons of  $N$  labelled by  $s_1, \dots, n_p$  sons of  $N$  labelled by  $s_p$ ,
  - (ii) there is a rule  $s \xrightarrow{\parallel} \phi(x_1, \dots, x_p) \in R'$ , such that  $\phi(n_1, \dots, n_p)$  is true.

A tree  $t$  is accepted by  $\mathcal{A}$  if there is a run of  $\mathcal{A}$  on  $t$  such that the root of  $t$  is labelled by a state  $s \in S_I$ .  $L(\mathcal{A})$  is the set of trees accepted by  $\mathcal{A}$  and a language  $L$  is a *Presburger regular language* iff  $L = L(\mathcal{A})$ . When there is no ambiguity, we say regular language. By construction  $t \in L(\mathcal{A})$  and  $s \equiv t$  implies  $s \in L(\mathcal{A})$ .

**Example 3** Let  $q(a^*(\dots))$  denote any  $q(a^n(\dots))$  for  $n \geq 0$ . The set of trees of the form  $q(a^*(\parallel (q(b^*(\#)), \dots, q(b^*(\#)), q(c^*(\#)), \dots, q(c^*(\#))))$  where the parallel operator  $\parallel$  has as many arguments  $qb^*\#$  as arguments  $qc^*\#$  is a Presburger regular language since it is accepted by the automaton  $\mathcal{A} = (S = \{s_a, s_b, s_c\}, S_I = \{s_a\}, R = \{s_a \xrightarrow{q} s_a; s_a \xrightarrow{a} s_a; s_b \xrightarrow{q} s_b; s_b \xrightarrow{b} s_b; s_c \xrightarrow{q} s_c; s_c \xrightarrow{c} s_c; s_b \rightarrow \#; s_c \rightarrow \#\}, R' = \{s_a \xrightarrow{\parallel} x_a = 0 \wedge x_b = x_c\})$  with  $x_a$  the variable for  $s_a$ ,  $x_b$  the variable for  $s_b$ ,  $x_c$  the variable for  $s_c$ .

The usual constructions on tree automata can be adapted to Presburger automata which yields the decidability of the emptiness of  $L(\mathcal{A})$  and the closure of regular languages under boolean operations, see [ZL06].

## 5 Weighted Automata for PDN

### 5.1 Semilinear Sets

Let  $b \in \mathbb{N}^m$ , let  $P = \{p_1, \dots, p_n\}$  be a finite subset of  $\mathbb{N}^m$ . The linear set  $L(b, P)$  of  $\mathbb{N}^m$  is the set  $\{b + \sum_{i=1}^n \lambda_i p_i \mid \lambda_i \in \mathbb{N}\}$ . A semilinear set is a finite union of linear sets. The  $+$  operation on subsets of  $\mathbb{N}^m$  is defined by  $L + M = \{x + y \mid x \in L, y \in M\}$ . The  $*$  operation is defined by  $L^* = \cup_{n \geq 0} L^n$  where  $L^0 = \{(0, \dots, 0)\}$ , and  $L^n = \underbrace{L + \dots + L}_n$ .

The set of rational expressions of semilinear sets is defined by

$$\text{Rat} ::= L \mid \text{Rat} + \text{Rat} \mid \text{Rat} \cup \text{Rat} \mid \text{Rat}^*$$

where  $L$  denotes any semilinear set. A rational expression  $R \in \text{Rat}$  denotes a set  $[R]$  inductively defined by

$$[L] = L \quad [R + R'] = [R] + [R'] \quad [R \cup R'] = [R] \cup [R'] \quad [R^*] = [R]^*$$

**Proposition 3** *Let  $R \in \text{Rat}$ . Then there exists a effectively constructible semilinear set  $L$  such that  $L = [R]$ .*

Semilinear sets and Presburger arithmetic are closely related since the set of valuations such that a formula  $\phi(x_1, \dots, x_p)$  is true is an effectively constructible semilinear set [GS66]. From now on, for the sake of simplicity, a semilinear set which contains only one element  $c$  will be written  $c$  (instead of  $\{c\}$ ).

### 5.2 Presburger Weighted Word Automata

A semiring is a structure  $(K, \oplus, \otimes, 0, 1)$  such that (i)  $K, \oplus$  is a commutative monoid with 0 as neutral element, (ii)  $K, \otimes$  is a monoid with 1 as neutral element, (iii)  $x \otimes (y \oplus z) = (y \oplus z) \otimes x = x \otimes y \oplus x \otimes z$  (iv) for all  $x \in K$ ,  $0 \otimes x = x \otimes 0 = 0$ . Let  $\mathcal{S}\mathcal{L}_m$  be the set of semilinear sets of  $\mathbb{N}^m$ . Then  $\mathcal{S}_m = (\mathcal{S}\mathcal{L}_m, \cup, +, \emptyset, (0, \dots, 0))$  is a semiring. Weighted automata are word automata where the transitions are labelled by element of a semiring  $K$ . Weighted automata have already used for *PDS* analysis provided that  $K$  satisfies additional properties [RSJ03]. Presburger weighted automata have a similar definition, but the semiring for labels is  $\mathcal{S}_p$  and the definition of the transition relation is slightly distinct from the standard one. Furthermore, these automata enjoy particular properties used in the reachability analysis of *PDN*.

**Definition 2** *A Presburger weighted word automaton is an automaton  $\mathcal{A} = (S, s_0, S_F, \mathcal{S}_m, \Delta)$  where  $S_F$  is a set of pairs  $(s, \phi_s)$  with  $s \in S, \phi_s \in \mathcal{S}_m, \Delta \subseteq S \times \Sigma \cup \{\epsilon\} \times \mathcal{S}_m \times S$ . A transition rule of  $\Delta$  is denoted  $s \xrightarrow{a, C} s'$ .*

*The transition relation  $\xrightarrow{*}_{\mathcal{A}} \subseteq \Sigma^* \times \mathbb{N}^m \times S$  is inductively defined by:*

- $\epsilon, (0, \dots, 0) \xrightarrow{*}_{\mathcal{A}} s_0$
- if  $w, c \xrightarrow{*}_{\mathcal{A}} s$  then  $wa, c + c' \xrightarrow{*}_{\mathcal{A}} s'$  if there is a rule  $s \xrightarrow{a, C'} s'$  with  $c' \in C'$ ,
- if  $w, c \xrightarrow{*}_{\mathcal{A}} s$  then  $w, c + c' \xrightarrow{*}_{\mathcal{A}} s'$  if there is a rule  $s \xrightarrow{\epsilon, C'} s'$  with  $c' \in C'$

A pair  $w, c$  is accepted iff  $w, c \xrightarrow{*} \mathcal{A} s$  and  $c \in \phi_s$  for  $(s, \phi_s) \in S_F$ . The language accepted by  $\mathcal{A}$  is the set  $L(\mathcal{A})$  of pairs accepted by  $\mathcal{A}$ .

**Proposition 4** For each  $s \in S$ , the set  $L(s_0, s) = \{c \in \mathcal{S}\mathcal{L}_m \mid \exists w \in \Sigma^* w, c \xrightarrow{*} \mathcal{A} s\}$  is an effectively computable semilinear set.

PROOF. By proposition 3, and the property that the set of words reaching a state of a (usual) word automaton can be described by a rational expression, we get that for each state  $s$  of a Presburger weighted word automaton, we can compute the semilinear set  $L(s_0, s) = \{c \in \mathcal{S}\mathcal{L}_m \mid \exists w \in \Sigma^* w, c \xrightarrow{*} \mathcal{A} s\}$ .  $\square$

Presburger weighted word automaton enjoy other properties that they share with Presburger weighted tree automaton and they are given in the next section.

### 5.3 Presburger Weighted Tree Automata

Presburger weighted tree automata are designed to accept set of trees corresponding to configurations of  $PDN$ .

**Definition 3** A top-down Presburger weighted tree automaton is a tuple  $\mathcal{A} = (S, S_I, S_m, R \cup R')$  where:

- $S = \{s_1, \dots, s_p\}$  is a finite set of states,
- $S_I \subseteq S$  is a set of initial states,
- $S_m$  is the semiring  $(\mathcal{S}\mathcal{L}_m, \cup, +, \emptyset, (0, \dots, 0))$ ,
- $R$  is a set of rules of the form  $s \xrightarrow{(0, \dots, 0)} \#$  or  $s \xrightarrow{a, C} s'$  where  $s, s' \in S, a \in Q \cup \Sigma \cup \{\epsilon\}, C \in \mathcal{S}_n$
- $R'$  is a set of rules  $s \parallel \phi(x_1, \dots, x_p, y_1, \dots, y_m)$  where  $\phi(x_1, \dots, x_p, y_1, \dots, y_m)$  is a Presburger formula.

A run of the automaton  $\mathcal{A}$  on a tree  $t$  is a labelling  $r : Nodes(t) \times \mathbb{N}^m \rightarrow S$  of the nodes of  $t$  by the states of  $S$  and weights of  $\mathbb{N}^m$  such that:

- if  $\#$  is the symbol of a node  $N$  of  $t$ , the node  $N$  is labelled by  $r(N) = (s, (0, \dots, 0))$  such that the rule  $s \xrightarrow{(0, \dots, 0)} \#$  belongs to  $R$ .
- if  $a \in Q \cup \Sigma$  is the symbol of a node  $N$  of  $t$ , and  $N$  is labelled by  $r(N) = (s, c)$ , then the unique son of  $N$  is labelled by  $(s', c + c')$  s.t. there is a sequence of rules  $s_1 \xrightarrow{\epsilon, C_1} s_2, \dots, s_{i-1} \xrightarrow{\epsilon, C_{i-1}} s_i, s_i \xrightarrow{a, C} s_{i+1}, s_{i+2} \xrightarrow{\epsilon, C_{i+2}} s_{i+3}, \dots, s_n \xrightarrow{\epsilon, C_n} s_{n+1}$  where  $s_1 = s, s_{n+1} = s'$  and  $c' = c_1 + \dots + c_n$  with  $c_i \in C_i$  for  $i = 1, \dots, n$ .
- if  $\parallel$  is the symbol of a node  $N$  of  $t$  and  $N$  is labelled by  $r(N) = (s, (c_1, \dots, c_m))$ , if  $N_1, \dots, N_n$  are the sons of  $N$ , then
  - (i) there are  $n_1$  sons of  $N$  labelled by  $(s_1, (0, \dots, 0)), \dots, n_p$  sons of  $N$  labelled by  $(s_p, (0, \dots, 0))$ ,
  - (ii) there is a rule  $s \parallel \phi(x_1, \dots, x_p, y_1, \dots, y_m) \in R'$ , such that  $\phi(n_1, \dots, n_p, c_1, \dots, c_m)$  is true.

$L(\mathcal{A})$  denotes the set of trees accepted by  $\mathcal{A}$ .

**Example 4** Let  $q(b^+(\#))$  denote  $q(b^n(\#))$  with  $n > 0$ . The (non-regular) language  $\mathcal{L}^1$

$$\{q(a^{n+1}(\| \underbrace{(q(b^+(\#)), \dots, q(b^+(\#))}_n, \underbrace{q(c^+(\#)), \dots, q(c^+(\#))}_n \|)) \mid n \geq 1\}$$

is accepted by the Presburger weighted tree automaton  $\mathcal{A} = (S, S_I, S_2, R \cup R')$

$$\begin{aligned} S &= \{s_0, s_{qa}, s_{qb}, s_{qc}, s_a, s_b, s_c, s_q\}, S_I = \{s_0\}, \\ R &= \{s_0 \xrightarrow{q, (0,0)} s_q, s_q \xrightarrow{a, (0,0)} s_a, s_a \xrightarrow{a, (1,1)} s_a, s_q \xrightarrow{q, (0,0)} s_b, s_b \xrightarrow{b, (0,0)} s_b, \\ &\quad s_b \xrightarrow{(0,0)} \#, s_{qc} \xrightarrow{q, (0,0)} s_c, s_c \xrightarrow{q, (0,0)} s_c, s_c \xrightarrow{(0,0)} \#, \}, \\ R' &= \{s_a \rightarrow x_{s_{qb}} = y_1 \wedge x_{s_{qc}} = y_2 \wedge \bigwedge_{s \neq s_{qb}, s_{qc}} x_s = 0 \} \end{aligned}$$

The tree  $q(a(a(\| (q(b(\#)), q(c(c(\#))\|))))$  is accepted by a run that labels the node  $\|$  by  $(s_a, (1, 1))$ , the first son of this node by  $s_{qb}$ , the second son by  $s_{qc}$ . The tree  $q(a(a(a(\| (q(b(\#)), q(c(\#))\|))))$  is not accepted since the node  $\|$  can be labelled only by  $(s_a, (2, 2))$  and this node has only two sons (when two nodes labelled by  $s_{qb}$  and two nodes labelled by  $s_{qc}$  are required).

A main feature of these automata is that the non-regular behavior is generated by weight computations. The Presburger formula of rules of  $R'$  can only be used to add additional regular constraints. For instance, we can build an automaton accepting the subset of  $\mathcal{L}$  corresponding to even values of  $n$  by replacing in  $\mathcal{A}$  the rule  $s_a \rightarrow x_{s_{qb}} = y_1 \wedge x_{s_{qc}} = y_2 \wedge \bigwedge_{s \neq s_{qb}, s_{qc}} x_s = 0$  by the rule  $s_a \rightarrow x_{s_{qb}} = y_1 \wedge x_{s_{qc}} = y_2 \wedge x_{s_{qb}} \% 2 = 0 \wedge x_{s_{qc}} \% 2 = 0 \wedge \bigwedge_{s \neq s_{qb}, s_{qc}} x_s = 0$ .

**Proposition 5** Let  $\mathcal{A}$  be a Presburger weighted tree automaton.

- There is a Presburger weighted tree automaton  $\mathcal{B}$  without  $\epsilon$ -rules such that  $L(\mathcal{B}) = L(\mathcal{A})$ .
- It is decidable if  $L(\mathcal{A})$  is empty.
- Let  $\mathcal{B}$  be a Presburger weighted tree automaton. Then there is an effectively computable Presburger weighted tree automaton  $\mathcal{C}$  s.t.  $L(\mathcal{C}) = L(\mathcal{A}) \cap L(\mathcal{B})$ .

Since a Presburger tree automaton can be seen as a Presburger weighted tree automaton by taking  $\mathcal{S}_p = \mathbb{N}$  and replacing rules  $s \xrightarrow{a} s'$  by  $s \xrightarrow{a, 0} s'$ , we can build a Presburger weighted tree automaton that accepts  $L(\mathcal{A}) \cap L(\mathcal{B})$  for  $\mathcal{B}$  a Presburger tree automaton.

## 6 Forward Analysis of PDN

In this section we consider a PDN  $P = (Q, \Sigma, R)$ . From now on, we say that a subset  $L \subseteq \mathcal{C}$  is a regular language iff the set  $c2t(L) = \{t \mid t = c2t(c), c \in L\}$  is a regular tree language.

**Example 5** Let  $P = (\{q\}, \{a, b, c\}, \{qa \rightarrow qaa \mid (qb, qc), qb \rightarrow qbb, qc \rightarrow qcc\})$ . The automaton of example 4 accepts  $\text{Post}^*(qa)$ .

<sup>1</sup>non-regularity follows from a straightforward pumping argument

Since this example has a non-regular set of successors, we get:

**Proposition 6**  $Post^*(L)$  can be a non-regular language.

Our goal is to show that Presburger weighted tree automata can be used to perform the forward analysis of PDN.

## 6.1 Preliminary Computations

The two following propositions are used to simplify the computation of  $Post^*(L)$ .

**Proposition 7** Let  $L \subseteq C$  be a regular language. There exists  $P' = (Q \cup Q', \Sigma \cup \{\$, R \cup R') such that  $Post_P^*(L) = Post_{P'}^*(q_0\$) \cap C$  where  $q_0 \in Q'$ .$

**Proposition 8** There exists a PDN  $P'$  such that (i)  $Post_P^*(qa) = Post_{P'}^*(qa) \cap C$  (ii) the rules of  $P'$  have the form  $qa \rightarrow q'$  or  $qa \rightarrow q'bc$  or  $qa \rightarrow q'(q_1a_1 \parallel \dots \parallel q_ma_m)$  or  $qa \rightarrow q'bc(q_1a_1 \parallel \dots \parallel q_ma_m)$

PROOF. (Sketch) A rule  $qa \rightarrow q'a_1 \dots a_n$  is replaced by rules  $qa \rightarrow \bar{q}_{n-1}\#a_n$ ,  $\bar{q}_{n-1}\# \rightarrow \bar{q}_{n-2}a_{n-1}, \dots, \bar{q}_1\# \rightarrow q'a_1$  where the  $\bar{q}_i$  and  $\#$  are new symbols. This can be done for all rules and yields a new PDN  $P'$  of size linear in the size of the initial PDN  $P$ . By construction  $Post_P^*(L) = Post_{P'}^*(L) \cap C$ .  $\square$

From now on, we assume that the rules of  $P = (Q, \Sigma, R)$  have the required form. Let  $m = |Q||\Sigma|$  and let us define an ordering of the set  $\{qa \mid q \in Q, a \in \Sigma\}$ . The  $i^{th}$  element  $qa$  in this ordering is identified with the tuple  $C(qa) = (0, \dots, 0, 1, 0, \dots, 0)$ . Therefore a parallel composition  $c_{\parallel} = q_{i_1}a_{j_1} \parallel \dots \parallel q_{i_k}a_{j_k}$  can be identified with a tuple  $C(c_{\parallel})$  of  $\mathbb{N}^m$ .

**Example 6** For the PDN of example 2,  $m = 4$  and assuming the ordering  $qa, qb, q'a, q'b$ , if  $c_{\parallel} = qa \parallel qb \parallel q'b$  then  $C(c_{\parallel}) = (1, 1, 0, 1)$  and if  $c'_{\parallel} = qa \parallel qb \parallel qb$  then  $C(c'_{\parallel}) = (1, 2, 0, 0)$ .

The transition relation  $\rightsquigarrow$  is the head-rewriting relation generated by  $R$ , i.e. it is the restriction of  $\rightarrow$  to the initial PDS of a configuration  $qw(c_{\parallel})$  (i.e. no transition is applied to any element of  $c_{\parallel}$ ). The reflexive transitive closure of  $\rightsquigarrow$  is denoted by  $\rightsquigarrow^*$ .

The next construction is inspired by [BMOT05] and computes, for each  $q, a, q'$ , a context-free grammar  $G$  such that the Parikh mapping of  $L(G)$  is the set  $\{C(c_{\parallel}) \mid qa \rightsquigarrow^* q'(c_{\parallel})\}$ , i.e. it describes all possible parallel compositions generated by transitions  $qa \rightsquigarrow^* q'(\dots)$ . In the following, we shall identify the  $i^{th}$  element  $qa$  to the letter  $l_i$  of an alphabet  $\{l_1, \dots, l_m\}$ . We recall that the Parikh mapping  $\#_P(w)$  of a word  $w$  is the tuple  $(n_1, \dots, n_m) \in \mathbb{N}^m$  such that the letter  $l_1$  has  $n_1$  occurrences in  $w, \dots$ , the letter  $l_m$  has  $n_m$  occurrences in  $w$ .

- The set of terminals is  $\{l_1, \dots, l_m\}$ . In the following,  $w(c_{\parallel})$  denotes the word  $l_1^{n_1} \dots l_m^{n_m}$  where  $(n_1, \dots, n_m) = C(c_{\parallel})$  (with notation  $l_i^0 = \epsilon$ ).
- The set of non-terminals is  $\{X_{q,a,q'} \mid q, q' \in Q, a \in \Sigma\}$ . They generate the words  $c$  such that  $\#_P(c) = C(c_{\parallel})$  iff  $qa \xrightarrow{*} q'(c_{\parallel})$ .
- The set  $\Delta$  of production rules is defined by

- if  $qa \rightarrow q'(c_{\parallel})$  belongs to  $R$ , then  $X_{q,a,q'} \rightarrow w(c_{\parallel}) \in \Delta$ ,
- if  $qa \rightarrow \bar{q}bc(c_{\parallel})$  belongs to  $R$ , then  $X_{q,a,q'} \rightarrow X_{\bar{q},b,\bar{q}}X_{\bar{q},c,q'}w(c_{\parallel}) \in \Delta$ ,

**Proposition 9** For all  $q, q'' \in Q, a \in \Sigma$ ,  $X_{q,a,q'} \xrightarrow{*}_G w$  iff  $qa \xrightarrow{*} q'(c_{\parallel})$  and  $C(c_{\parallel}) = \#_P(w)$

Parikh's theorem and the results of [VSS05] yield that:

**Proposition 10** The set  $C(q, a, q') = \{C(c_{\parallel}) \mid qa \xrightarrow{*} q'(c_{\parallel})\}$  is a semilinear set and an existential Presburger arithmetic formula defining  $C(q, a, q')$  can be computed in polynomial time.

## 6.2 A Presburger Weighted Tree Automaton accepting $Post^*(qa)$

Firstly, we construct a Presburger weighted word automaton  $\mathcal{A} = (S, s_0, S_F, S_m, \Delta)$  such that  $qa \xrightarrow{*} q'w(c_{\parallel})$  iff  $\mathcal{A}$  accepts  $q'w, C(c_{\parallel})$ .

- $S = \{s_0\} \cup \{s_q \mid q \in Q\} \cup \{s_{qa} \mid s \in Q, a \in \Sigma\}$ ,  $S_F = \{(s_{qa}, \mathbb{N}^m)\}$ . States  $s_q$  are reached by the unique word  $q$ , and the state  $s_{qa}$  is reached by all pairs  $q'w, c$  such that  $qa \xrightarrow{*} q'w(c_{\parallel})$  and  $c = C(c_{\parallel})$ .

- $\Delta$  is defined by:

- $s_0 \xrightarrow{q, (0, \dots, 0)} s_q \in \Delta$  and  $s_q \xrightarrow{a, (0, \dots, 0)} s_{qa} \in \Delta$
- if  $R$  contains the rule  $qa \rightarrow q'(c_{\parallel})$  then  $s_{q'} \xrightarrow{\epsilon, C(c_{\parallel})} s_{qa} \in \Delta$
- if  $R$  contains the rule  $qa \rightarrow q'bc(c_{\parallel})$  then
  - \*  $s_{q'b} \xrightarrow{c, C(c_{\parallel})} s_{qa} \in \Delta$
  - \*  $s_{q''c} \xrightarrow{\epsilon, C(q', b, q'') + C(c_{\parallel})} s_{qa} \in \Delta$

**Proposition 11**  $\bar{q}w, c \xrightarrow{*}_{\mathcal{A}} s_{qa}$  iff  $qa \xrightarrow{*} \bar{q}w(c_{\parallel})$  and  $c = C(c_{\parallel})$ .

The previous Presburger weighted word automaton  $\mathcal{A} = (S, s_0, S_F, S_m, R)$  is extended into a Presburger weighted tree automaton  $\mathcal{B} = (S_{\mathcal{B}}, S_I, S_m, R_{\mathcal{B}})$  that accepts  $Post^*(q_0a_0)$ , where  $m = |\{s_0^{qa} \mid q \in Q, a \in \Sigma\}|$  and:

- $S_{\mathcal{B}} = \{s^{qa} \mid s \in S, qa \in Q \times \Sigma\}$  and  $S_I = \{s_0^{q_0a_0}\}$ .

The states of  $S_{\mathcal{B}}$  are ordered  $s_1, \dots, s_m$  where  $s_i = s_0^{qa}$  iff  $i$  is the  $i^{th}$  component of  $S_p$  (i.e. corresponds to the ordering of  $qa$  defined for the word automaton  $\mathcal{A}$ ).

- The set of rules contains

- $s^{qa} \xrightarrow{a, C} s'qa$  if  $\mathcal{A}$  contains the rule  $s \xrightarrow{a, C} s'$ .
- $s^{qa} \xrightarrow{(0, \dots, 0)} \#$ , this rule is needed since  $\mathcal{A}$  deals with words and  $\mathcal{B}$  deals with trees,
- $s^{qa} \xrightarrow{\parallel} \bigwedge_{i=1}^{i=m} x_i = y_i \wedge \bigwedge_{i>m} x_i = 0$

The last rule states that the arguments of a parallel composition are exactly the processes generated by the  $q(w(\dots))$  part above this parallel composition.

**Proposition 12**  $L(\mathcal{B}) = c2t(Post^*(qa))$ .

PROOF. The proof is by structural induction on the tree structure. We prove that for any  $q, a, t \in c2t(Post^*(qa))$  iff  $t$  is accepted by  $\mathcal{A}$  with a run labelling the root of  $t$  by  $s_0^{qa}$ .

Base case.  $t = q(w(\#))$ . By definition of the rules of  $\mathcal{B}$  and adapting the proof of proposition 11, we have  $q(w(\#)) \in Post^*(qa)$  iff a run of  $\mathcal{B}$  labels  $q(w(\#))$  by  $(s_0^{qa}, (0, \dots, 0))$  at the root and  $q_{qa}^{qa}$  at the leaf.

Induction step.  $t = q(w(\parallel (t_1, \dots, t_n)))$ . We assume that there is a run of  $\mathcal{B}$  labelling each  $t_i$  by  $(s_0^{qa}, (0, \dots, 0))$  iff  $t_i \in Post^*(qa)$ . By definition of the rules of  $\mathcal{B}$  and adapting the proof of proposition 11,  $qa \xrightarrow{*} t2c(t)$  there is a rule labelling the node  $\parallel$  by  $(s_{qa}^{qa}, (n_1, \dots, n_m))$  such that  $n_1$  trees  $t_i$  are labelled by  $(s_0^{(qa)^1}, (0, \dots, 0)), \dots, n_m$  trees  $t_i$  are labelled by  $(s_0^{(qa)^m}, (0, \dots, 0))$  and no  $t_i$  is labelled by another state (where  $(qa)_i$  denotes the  $i^{th}$  element in the sequence of  $qa$ 's). By induction hypothesis, there is a run of  $\mathcal{B}$  accepting  $t$  iff  $t2c(t) \in Post^*(qa)$ . □

This proposition and the properties of Presburger weighted trees automata yield that forward analysis of  $PDN$  is decidable.

## 7 Backward Analysis of $PDN$

To perform backward analysis, we can rely on the results of [BMOT05] that state that  $Pred^*(L)$  is accepted by a hedge automaton and the fact that the closure of a regular hedge language under commutativity is a Presburger regular language. Therefore we get:

**Proposition 13** *The set  $Pred^*(L)$  is accepted by a Presburger tree automaton.*

## Conclusion

We have enriched the model of dynamic networks of pushdown systems by taking parallel composition as an associative-commutative operator. Using a new class of tree automata we have been able to do forward and backward analysis. Forward analysis involves an exponential blowup in the construction of the  $PDN$  of proposition 7 since a semilinear set equivalent to a Presburger formula can be exponentially larger. This problem occurs usually when switching from a word framework to a natural number framework. Some interesting questions remain and will be further investigating. The first one is to extend our result using constrained rules as in [BMOT05]. Another one is to consider a flat model where all parallel compositions are at the same level and a configuration is now a set of  $PDS$  instead of a tree. This model could allow a easier approach to synchronization in the flavor of [KG07].

## References

- [AB97] O. Maler A. Bouajjani, J. Esparza. Reachability analysis of push-down automata: Application to model-checking. In *Proceedings 8th International Conference on Concurrency Theory (CONCUR)*, volume 1243 of *LNCS*, pages 14–25. Springer-Verlag, 1997.
- [BMOT05] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 473–487. Springer, 2005.
- [Buc64] R. Buchi. Regular canonical systems. *Archiv fur Mathematische Logik und Grundlagenforschung*, 6(91-111), 1964.
- [Cau00] D. Caucal. On word rewriting systems having a rational derivation. In Jerzy Tiuryn, editor, *FoSSaCS*, volume 1784 of *LNCS*, pages 48–62. Springer, 2000.
- [CDJ<sup>+</sup>99] H. Comon, M. Dauchet, F. Jacquemard, C. Loeding, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata on their application*. Freeware Book, 1999. available at <http://tata.gforge.inria.fr/>.
- [EP00] J. Esparza and A. Podelski. Efficient algorithms for pre<sup>\*</sup> and post<sup>\*</sup> on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000.
- [GS66] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.
- [KG07] V. Kahlon and A. Gupta. On the analysis of interacting pushdown systems. In Martin Hofmann and Matthias Felleisen, editors, *POPL*, pages 303–314. ACM, 2007.
- [LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on p-processes. In D.Sangiorgi and R.de Simone, editors, *Concur'98*, volume 1466 of *LNCS*, pages 50–66. Springer-Verlag, September 1998.
- [Pre29] M. Presburger. Uber die vollstandigkeit eines gewissen system der arithmetik ganzer zahlen in welchem die addition als einzige operation hervortritt. In *Comptes Rendus du I Congres des Mathematiciens des Pays Slaves, Warszawa*, 1929.
- [RSJ03] T. W. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In Radhia Cousot, editor, *SAS*, volume 2694 of *Lecture Notes in Computer Science*, pages 189–213. Springer, 2003.
- [VSS05] K.N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*, pages 337–352. Springer, 2005.
- [ZL06] S. Dal Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):337–377, 2006.

## Appendix

### 7.1 Properties of Semilinear Sets

We recall how semilinear sets are preserved under the  $\cup, +, *$  operations.

- This holds by definition for  $\cup$ .
- $(L \cup M) + K = L + K \cup M + K$  and  $L(b, P) + L(b', P') = L(b + b', P \cup P')$  therefore the property holds for  $+$ .
- $(L \cup M)^* = L^* + M^*$  and  $L(b, P)^* = (0, \dots, 0) \cup L(b, P \cup \{b\})$  which proves the property for  $*$ .

### 7.2 Properties of Pushdown Systems

The following proposition states that the computation of  $Post^*(L)$  is equivalent to the computation of  $Post^*(c)$  using a slightly modified *PDS*.

**Proposition 14** *Let  $P = (Q, \Sigma, R)$  be a PDS and  $L \subseteq C$  be a regular language. There exists  $P' = (Q \cup Q', \Sigma \cup \{\$, \}, R \cup R')$  such that  $Post_P^*(L) = Post_{P'}^*(q_0\$) \cap C$  where  $q_0 \in Q'$ .*

PROOF. A regular set  $L$  on the alphabet  $\Sigma' = Q \cup \Sigma$  can be generated by a left-linear grammar  $G = (Prod, X, \Sigma', X_0)$  where the productions have the form  $X \rightarrow X'a$  or  $X \rightarrow a$  with  $a \in \Sigma'$  (and  $a \in Q$  for the second type of productions if  $L \subseteq C$ ).

Let  $P' = (Q \cup Q', \Sigma \cup \{\$, \}, R \cup R')$  where  $Q' = \{q_X \mid X \in X\}$ ,  $R' = \{q_X\$ \rightarrow q_{X'}\$a \mid X \rightarrow X'a \in Prod\} \cup \{q_X\$ \rightarrow a \mid X \rightarrow a \in Prod\}$ . Let  $q_0 = q_{X_0}$ .

The transition relation of  $P'$  is equivalent to  $\xrightarrow{*}_R \circ \xrightarrow{*}_{R'}$  since (i) no rule of  $R$  is applicable until a rule  $q_X\$ \rightarrow a$  is applied, (ii) after such a rule is applied only rules of  $R$  can be used.

By definition of  $P'$ , each derivation  $q_0 \xrightarrow{*} qw$  of an element  $qw \in L$  by the grammar  $G$  is simulated by a derivation  $q_0\$ \xrightarrow{*} qw$  of  $P'$  using only rules of  $R'$ . Conversely, each derivation  $q_0\$ \xrightarrow{*} qw$  of  $P'$  using only rules of  $R'$  simulates a derivation of  $G$ .

- $Post_P^*(L) \subseteq Post_{P'}^*(q_0\$)$ . Let  $q'w' \in Post_P^*(L)$ . By definition there exists  $qw \in L$  such that  $qw \xrightarrow{*}_R q'w'$ . By the previous remark  $q_0\$ \xrightarrow{*}_{R \cup R'} qw$ , and  $qw \xrightarrow{*}_{R \cup R'} q'w'$  since  $P'$  contains the rules of  $R$ .
- $Post_{P'}^*(q_0\$) \cap \Sigma' \subseteq Post_P^*(L)$ .

Let  $c \in Post_{P'}^*(q_0\$) \cap \Sigma'$ . Since  $c \in \Sigma'$ , then  $c = qw$ . Since the transition relation of  $P'$  is  $\xrightarrow{*}_R \circ \xrightarrow{*}_{R'}$ ,  $q_0\$ \xrightarrow{*}_{R'} q'w' \xrightarrow{*}_R qw$  with  $q'w' \in L$ , therefore  $qw \in Post^*(L)$ .

□

The previous proposition can be extended to *PDN*. Let  $P = (Q, \Sigma, R)$  be a *PDN*.

**Proposition 15** *Let  $L \subseteq C$  be a regular language. There exists  $P' = (Q \cup Q', \Sigma \cup \{\$, \}, R \cup R')$  such that  $Post_P^*(L) = Post_{P'}^*(q_0\$) \cap C$  where  $q_0 \in Q'$ .*

PROOF. Let  $\mathcal{A} = (S, S_I, \mathcal{R} \cup \mathcal{R}')$  be a top-down Presburger tree automaton accepting  $L' = c2t(L)$ . For each rule  $s \xrightarrow{\parallel} \phi(x_1, \dots, x_p) \in \mathcal{R}'$  we compute the semilinear set  $L_{s,\phi}$  of  $\mathbb{N}^m$  equivalent to  $\phi(x_1, \dots, x_p)$ . This set is a finite union of linear sets  $SL_{s,\phi}^i = L(b_i, \mathcal{P}_i)$ .

The *PDN*  $P'$  performs transitions that generates  $L$  and then performs the transitions of  $P$ . In the first computations, the new symbol  $\$$  is used to prevent the application of rules of  $P$ .

- The set  $Q'$  is defined as follows:
  - $Q'$  contains a starting state  $q_0$ ,
  - $Q'$  contains states  $q_0(s)$  for each  $s \in S$ ,
  - if  $s, s' \in S$  then  $Q'$  contains  $q(s, s')$ ,
  - if  $SL_{s',\phi} = L(b, \mathcal{P})$  is a semilinear set corresponding to a rule  $s' \xrightarrow{\parallel} \phi(x_1, \dots, x_p)$ , then  $Q'$  contains  $q(s, s', \parallel, \phi)$  and  $q(s, s', \parallel, L(b, \mathcal{P}))$ .
- The set of rules  $R'$  is defined by:
  - $P'$  chooses non-deterministically to generate a configuration  $qw$  or a configuration  $qw(\dots \parallel \dots \parallel \dots)$ . Furthermore a run of  $\mathcal{A}$  labels the root of  $c2t(qw)$  by  $s$  and the last node of  $qw$  by  $s'$ . The corresponding rules are:
    - \*  $q_0\$ \rightarrow q(s, s')\$$ ,
    - \*  $q_0\$ \rightarrow q(s, s', \parallel, \phi)\$$ ,
  - $P'$  performs the same choice as above, but from  $q_0(s)$  instead of  $q_0$  ( $q_0\$$  occurs at the root of the initial configuration, when  $q_0(s)\$$  is an initial configuration of an argument of a parallel composition)
    - \*  $q_0(s)\$ \rightarrow q(s, s')\$$ ,
    - \*  $q_0(s)\$ \rightarrow q(s, s', \parallel, \phi)\$$ ,
  - $q(s, s')\$ \rightarrow q(s, s'')\$a$  if  $s'' \xrightarrow{a} s' \in \mathcal{R}$ . This rule simulates the automaton rule  $s'' \xrightarrow{a} s'$  in a backward way (which amounts to using a left-linear grammar as in the proof of proposition 1).
  - $q(s, s')\$ \rightarrow q$  if  $\mathcal{A}$  contains a rule  $s \xrightarrow{q} s'$ .
  - if there is an automaton rule  $s' \xrightarrow{\parallel} \phi(x_1, \dots, x_p)$  and  $L(b, \mathcal{P})$  is a linear set associated to  $\phi$ , the *PDN*  $P$  simulates the generation of the parallel composition with the rules:
    - \*  $q(s, s', \parallel, \phi)\$ \rightarrow q(s, s', \parallel, L(b, \mathcal{P}))\$(q_0(S)^b)$
    - \*  $q(s, s', \parallel, L(b, \mathcal{P}))\$ \rightarrow q(s, s', \parallel, L(b, \mathcal{P}))\$(q_0(S)^p)$  for any  $p \in \mathcal{P}$ ,
where  $q_0(S)^b$  for  $b = (b_1, \dots, b_p)$  (resp.  $q_0(S)^p$  for  $p = (p_1, \dots, p_p)$ ) denotes  $q_0(s_1)\$^{b_1} \parallel \dots \parallel q_0(s_p)\$^{b_p}$  (resp.  $q_0(s_1)\$^{p_1} \parallel \dots \parallel q_0(s_p)\$^{p_p}$ ) where (i)  $q(s_i)\$^l$  is a parallel composition of  $l$  instances  $q(s_i)\$$ , (ii)  $S = \{s_1, \dots, s_p\}$ .
- $q(s, s', \parallel, L(b, \mathcal{P}))\$ \rightarrow q(s, s')\$$ . This rule stops the generation of arguments of  $\parallel$  and starts the generation of the *PDS* string like part  $qw$ .

If a transition of  $R$  is applied to a configuration  $c$  and a rule of  $R'$  is applied later, we can exchange the order of application and get the same result. Therefore the transition relation  $\xrightarrow{*}_{P'}$  can be defined as  $\xrightarrow{*}_P \circ \xrightarrow{*}_{R'}$  since the rules of  $R$  and  $R'$  are independent.

To end the proof, we simply  $c \in \text{Post}_{R'}^*(q_0\$) \cap C$  iff  $c \in L$ . The proof is by structural induction on  $c$ .

Base case.  $c = qw$ . The proof is similar to the proof of proposition 1.

Inductive step.  $c = qw(c_{\parallel})$ . Then  $c$  is generated by a sequence of transitions  $q_0\$ \rightarrow q(s, s', \phi)\$ \xrightarrow{*} \dots$ . From  $q(s, s', \phi)$  any parallel composition  $c_1 \parallel \dots \parallel c_m$  such that  $n_1$  are generated by  $q_0(s_1), \dots, n_p$  are generated by  $q_0(s_p)$  such that  $\phi(n_1, \dots, n_p)$  is true since  $(n_1, \dots, n_p) \in L(b, \mathcal{P})$  by definition of rules. Conversely any such configuration can be generated since any tuple of  $L(b, \mathcal{P})$  can be generated. By induction hypothesis  $c2t(c_i)$  is in the language generated by  $s_{i_j}$  in the automaton  $\mathcal{A}$ . Furthermore the transition contains the transition rules  $q(s, s', L(b, \mathcal{P})) \xrightarrow{*} q(s, s', \$)$  that generate the initial part  $qw$  of the configuration (with root labelled by  $s$  and last node by  $s'$ ).

Therefore  $c \in \text{Post}_{R'}^*(q_0\$) \cap C$  iff  $c \in L$ .  $\square$

### 7.3 Properties of Presburger Weighted Word and Tree Automata

**Proposition 16** *Let  $\mathcal{A}$  be a Presburger weighted word automaton. There there exists a Presburger weighted word automaton  $\mathcal{B}$  without  $\epsilon$ -rule such that  $L(\mathcal{A}) = L(\mathcal{B})$ .*

PROOF. We consider  $\mathcal{A}$  as a usual word on the alphabet  $(a, C)$ . For each pair  $s, s'$  of states of  $S$ , a classical algorithm on word automata computes the rational expression  $R(s, s')$  giving the language  $\{C_1 \dots C_n \mid s.(\epsilon, C_1) \dots (a, C_i) \dots (\epsilon, C_n) \xrightarrow{*} s'\}$ . By proposition 4, this rational expression is a effectively computable semi-linear set.

The automaton  $\mathcal{B}$  is obtained by eliminating all  $\epsilon$ -rule and replacing each  $s \xrightarrow{a, C} s'$  by  $s \xrightarrow{a, R(s, s')} s'$ . A routine reasoning on derivation length shows that  $L(\mathcal{A}) = L(\mathcal{B})$ .  $\square$

The proposition is easily extended to Presburger weighted tree automata.

Let  $\mathcal{A} = (S, S_I, S_m, R_{\mathcal{A}} \cup R'_{\mathcal{A}})$  be a Presburger weighted tree automaton.

**Proposition 17** *It is decidable if  $L(\mathcal{A})$  is empty.*

PROOF. First we eliminate unreachable states using a classical marking algorithm. We eliminate rules of  $R'$  such that  $\phi$  is unsatisfiable.

We write  $s \rightarrow_R s'$  iff there is a rule  $s \xrightarrow{a, C} s'$  and  $\xrightarrow{*}_R$  is the reflexive transitive closure of  $\rightarrow_R$ , and  $\xrightarrow{+}_R = \xrightarrow{*}_R \circ \rightarrow_R$ . We write  $s \xrightarrow{*}_R \#$  if  $s \xrightarrow{*} s'$  and there is a rule  $s' \xrightarrow{(0, \dots, 0)} \#$ .

Let  $\text{Prod} = \{s \in S \mid s \xrightarrow{*}_R \#\}$  (productive states)

Restricting the rules to  $R$ , for each  $s \in S$ , we derive a Presburger weighted word automaton  $\mathcal{A}_s = (S, s, S_F = \{(s, \text{True}) \mid s \xrightarrow{(0, \dots, 0)} \# \in R\}, S_m, R)$  from  $\mathcal{A}$ . We compute  $L(s, s')$  the set of possible weights for each  $s' \in S$  (see proposition 4).

The following algorithm combines the algorithm for deciding emptiness of Presburger automata [ZL06] and weights computations. This algorithm returns empty iff  $L(\mathcal{A})$  is empty.

```

Prod = { $s \mid s \xrightarrow{*}_R \#$ }
while a new state can be added to Prod do
  choose  $s \notin \textit{Prod}$ 
  for each  $s'$  such that  $s \xrightarrow{+} s'$ 
    for each rule  $s' \xrightarrow{\parallel} \phi(x_1, \dots, x_p, y_1, \dots, y_m)$ 
      if  $(y_1, \dots, y_m) \in L(s, s') \wedge \phi(x_1, \dots, x_p, y_1, \dots, y_m) \wedge \bigwedge_{s_i \notin \textit{Prod}} x_i = 0$ 
        is satisfiable,
      then add  $s$  to Prod
od
if  $S_I \cap \textit{Prod} \neq \emptyset$  then return non-empty.
else return empty

```

The computation terminates because the *while* loop can be executed at most  $|S|$  times.

We prove that  $s \in \textit{Prod}$  iff there is a tree  $t$  and a run of  $\mathcal{A}$  labelling the root of  $t$  by  $s$ .

- Base case. This is true before entering the loop.
- Induction step. Assume that the property holds for each state of *Prod* at the  $k^{\text{th}}$  iteration. Let  $s$  be added at iteration  $k + 1$ . By definition, there are trees  $t_i$  that a run of  $\mathcal{A}$  labels by  $s_i \in \textit{Prod}$ , and by definition of  $s'$  and the satisfiability of  $(y_1, \dots, y_m) \in L(s, s') \wedge \phi(x_1, \dots, x_p, y_1, \dots, y_m) \wedge \bigwedge_{s_i \notin \textit{Prod}} x_i = 0$  there is a tree  $t = q(w(\parallel (t_{i_1}, \dots, t_{i_k})))$  that a run of  $\mathcal{A}$  labels by  $s$  at the root.

Conversely we prove that if a run labels a tree  $t$  by  $s$ , then  $s \in \textit{Prod}$ .

The proof is by structural induction on  $t$ .

- $t = q(w(\#))$ . The initial computation of *Prod* ensures that  $s \in \textit{Prod}$ .
- Inductive step.  $t = q(w(\parallel (t_1, \dots, t_k)))$ . Let  $r$  be a run of  $\mathcal{A}$  labelling the root by  $s$ , the  $\parallel$  node by  $s'$  and the roots of the  $t_i$ 's by  $s_i$ .

By induction hypothesis,  $s_i \in \textit{Prod}$  for  $i = 1, \dots, m$ . By definition of a run, there is a rule  $s' \xrightarrow{\parallel} \phi(x_1, \dots, x_p, y_1, \dots, y_m)$  such that (i) the condition  $\phi(x_1, \dots, x_p, y_1, \dots, y_m)$  is true for  $y_1, \dots, y_m \in L(s, s')$  and (ii) all  $x_i$  corresponding to states different from any state  $s_i$  labelling the root of the  $t_i$  are equal to 0. Therefore,  $s$  can be added to *Prod*.

The last instruction of the algorithm tests whether any initial state can label a tree in a successful run of  $\mathcal{A}$ . □

**Proposition 18** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be two Presburger weighted tree automaton. Then there is a effectively computable Presburger weighted tree automaton  $\mathcal{C}$  such that  $L(\mathcal{C}) = L(\mathcal{A}) \cap L(\mathcal{B})$ .*

PROOF. Let  $\mathcal{A}$  and  $\mathcal{B}$  be two Presburger weighted tree automaton such that  $\mathcal{A} = (S = \{s_1, \dots, s_p\}, S_I, \mathcal{S}_I, R_{\mathcal{A}} \cup R'_{\mathcal{A}})$  and  $\mathcal{B} = (S' = \{s_1, \dots, s_{p'}\}, S'_I, \mathcal{S}'_I, R_{\mathcal{B}} \cup R'_{\mathcal{B}})$ . We can assume that  $\mathcal{A}$  and  $\mathcal{B}$  have no  $\epsilon$ -transition

We define  $\mathcal{C} = \mathcal{A} \times \mathcal{B} = (S \times S', S_I \times S'_I, \mathcal{S}_I + \mathcal{S}'_I, R, R')$  where the rules are defined by:

- the set  $S \times S'$  is ordered as  $(s_1, s'_1), \dots, (s_1, s'_{p'}), \dots, (s_p, s'_1), \dots, (s_p, s'_{p'})$ ,
- $(s_1, s_2) \xrightarrow{a, C_1, C_2} (s'_1, s'_2) \in R$  iff
 
$$\begin{aligned} s_1 &\xrightarrow{a, C_1} s'_1 \in R_{\mathcal{A}} \\ s_2 &\xrightarrow{a, C_2} s'_2 \in R_{\mathcal{B}} \\ C_1, C_2 &= \{(n_1, \dots, n_l, m_1, \dots, m_{l'}) \mid (n_1, \dots, n_l) \in \mathbb{N}^l, (m_1, \dots, m_{l'}) \in \mathbb{N}^{l'}\} \end{aligned}$$
- $(s_1, s_2) \xrightarrow{\parallel} \phi_1 \times \phi_2(x_{1,1}, \dots, x_{p,p'}, y_1, \dots, y_l, z_1, \dots, z_{l'}) \in R'$  if
 
$$s_1 \xrightarrow{\parallel} \phi_1(x_1, \dots, x_p, y_1, \dots, y_m) \in R'_{\mathcal{A}} \quad s_2 \xrightarrow{\parallel} \phi_2(x_1, \dots, x_{p'}, z_1, \dots, z_{l'}) \in R'_{\mathcal{B}}$$
 where  $\phi_1 \times \phi_2(x_{1,1}, \dots, x_{p,p'}, y_1, \dots, y_l, z_1, \dots, z_{l'})$  is the formula

$$\begin{aligned} \exists X_1, \dots, X_p, Y_1, \dots, Y_{l'} \quad &\bigwedge_{i=1}^{i=p} X_i = \sum_{j=1}^{j=p'} x_{i,j} \wedge \bigwedge_{j=1}^{j=p'} Y_j = \sum_{i=1}^{i=p} x_{i,j} \\ &\wedge \phi_1(X_1, \dots, X_p, y_1, \dots, y_l) \wedge \phi_2(Y_1, \dots, Y_{l'}, z_1, \dots, z_{l'}) \end{aligned}$$

a simpler formula without new variables exists, but the new existential variables make the definition clearer.

1.  $L(\mathcal{C}) \subseteq L(\mathcal{A}) \cap L(\mathcal{B})$ .

Let  $r$  be an accepting run of  $\mathcal{C}$  on  $t$ . By construction of  $\mathcal{C}$ , we can derive an accepting run  $r_{\mathcal{A}}$  of  $\mathcal{A}$  on  $t$  by keeping only the first components in states and weights: all nodes in string like parts  $q(w(\dots))$  are labelled according to the rules of  $\mathcal{A}$  and the weight labelling a  $\parallel$  node also satisfies the formula  $\phi_1(n_1, \dots, n_p, c_1, \dots, c_l)$  where  $n_i$  is the number of nodes labelled  $(s_i, -)$  by  $r$ , hence labelled  $s_i$  by  $r_{\mathcal{A}}$ . Therefore  $t \in L(\mathcal{A})$ . The same reasoning on the second component yields  $t \in L(\mathcal{B})$ .

2.  $L(\mathcal{A}) \cap L(\mathcal{B}) \subseteq L(\mathcal{C})$ .

Let  $t \in L(\mathcal{A}) \cap L(\mathcal{B})$ . There exists an accepting run  $r_{\mathcal{A}}$  of  $\mathcal{A}$  on  $t$  and an accepting run  $r_{\mathcal{B}}$  of  $\mathcal{B}$ . We compose these two runs to get an accepting run  $r$  of  $\mathcal{C}$ : if  $r_{\mathcal{A}}$  labels  $N$  by  $(s, c)$  and  $r_{\mathcal{B}}$  labels  $N$  by  $(s', c')$  we label  $N$  by  $(s \times s', (c, c'))$  (remember that the automata have no  $\epsilon$ -rule).

A node  $\parallel$  is labelled  $(s \times s', (c_1, \dots, c_l, c'_1, \dots, c'_{l'}))$  by  $r$  and  $n_{i,j}$  sons of  $N$  are labelled by  $(s_i \times s'_j, (0, \dots, 0))$  such that:

- $\phi_1(n_1, \dots, n_p, c_1, \dots, c_l)$  is true with  $n_i = \sum_{j=1}^{j=p'} n_{i,j}$  for  $i = 1, \dots, p$
- $\phi_2(m_1, \dots, m_p, c'_1, \dots, c'_{l'})$  is true with  $m_j = \sum_{i=1}^{i=p} n_{i,j}$  for  $j = 1, \dots, p'$

Therefore  $r$  is defined according to the rules of  $\mathcal{C}$  and we get  $t \in L(\mathcal{C})$ . □

## 7.4 Presburger Weighted Automata and PDN

The next proposition states the main property of the grammar of section 6.

**Proposition 19** For all  $q, q' \in Q, a \in \Sigma, X_{q,a,q'} \xrightarrow{*}_G w$  iff  $qa \rightsquigarrow q'(c_{\parallel})$  and  $C(c_{\parallel}) = \#_P(w)$

PROOF.  $\implies$  direction.

The proof is by induction on the length  $n$  of the derivation.

$n = 1$ . Straightforward.

Inductive step.

Case 1:  $X_{q,a,q'} \rightarrow w(c_{\parallel})$  Straightforward.

Case 2:  $X_{q,a,q'} \rightarrow X_{\bar{q},b,\bar{q}} X_{\bar{q},c,q'} w(c_{\parallel}) \xrightarrow{*} w$ .

By induction hypothesis  $X_{\bar{q},b,\bar{q}} \xrightarrow{*} w_1$  and  $\bar{q}b \rightsquigarrow \tilde{q}(c_{\parallel}^1)$  with  $C(c_{\parallel}^1) = \#_P(w_1)$  and  $X_{\bar{q},c,q'} \xrightarrow{*} w_2$  and  $\tilde{q}c \rightsquigarrow q'(c_{\parallel}^2)$  with  $C(c_{\parallel}^2) = \#_P(w_2)$ .

Therefore  $X_{q,a,q'} \rightarrow w = w_1 w_2 w(c_{\parallel})$  with  $qa \rightsquigarrow \bar{q}bc \rightsquigarrow \tilde{q}c((c_{\parallel} \parallel c_{\parallel}^1) \rightsquigarrow q'(c_{\parallel} \parallel c_{\parallel}^1 \parallel c_{\parallel}^2))$  and  $\#_P(w_1 w_2 w(c_{\parallel})) = C(c_{\parallel}) + C(c_{\parallel}^1) + C(c_{\parallel}^2) = C(c_{\parallel}^1 \parallel c_{\parallel}^2 \parallel c_{\parallel})$

$\Leftarrow$  direction. The proof is by induction on the length of the transition  $qa \xrightarrow{*} q'(c_{\parallel})$ .

Case 1.  $qa \rightsquigarrow q'(c_{\parallel})$  with rule  $qa \rightarrow q'(c_{\parallel})$ : straightforward.

Case 2.  $qa \rightsquigarrow \bar{q}bc(c_{\parallel}) \rightsquigarrow \tilde{q}c(c_{\parallel}^1 \parallel c_{\parallel}) \rightsquigarrow q'(c_{\parallel}^2 \parallel c_{\parallel}^1 \parallel c_{\parallel})$ .

By induction hypothesis  $X_{\bar{q},b,\bar{q}} \rightsquigarrow w_1$  such that  $\#_P(w_1) = C(c_{\parallel}^1)$  and  $X_{\bar{q},c,q'} \rightsquigarrow w_2$  such that  $\#_P(w_2) = C(c_{\parallel}^2)$ . Therefore  $X_{q,a,q'} \rightsquigarrow w = w_1 w_2 w(c_{\parallel})$  and  $\#_P(w) = C(c_{\parallel}^1 \parallel c_{\parallel}^2 \parallel c_{\parallel})$  □

Let  $\mathcal{A}$  be the automaton constructed in section 6

**Proposition 20**  $\bar{q}w, c \xrightarrow{*}_{\mathcal{A}} s_{qa}$  iff  $qa \rightsquigarrow \bar{q}w(c_{\parallel})$  and  $c = C(c_{\parallel})$ .

PROOF. Case 1:  $\implies$  direction. The proof is by induction on the length of the derivation  $\bar{q}w, c \xrightarrow{*}_{\mathcal{A}} qa$ . By construction, each state  $s_q$  is reached by the unique word  $q$ .

1. The last rule of the derivation is  $s_0 \xrightarrow{q,(0,\dots,0)} s_q$  or  $s_q \xrightarrow{a,(0,\dots,0)} s_{qa}$ . Straightforward.
2. The last rule of the derivation is  $s_{q'} \xrightarrow{\epsilon, C(c_{\parallel})} s_{qa}$ . Then  $\bar{q}w = q'$  with  $qa \rightarrow q'(c_{\parallel}) \in R$  hence  $qa \rightsquigarrow q'(c_{\parallel})$ .
3. The last rule of the derivation is  $s_{q'b} \xrightarrow{c, C(c_{\parallel})} s_{qa}$ .

Then  $\bar{q}w = \bar{q}w'c$  where  $\bar{q}w', c' \xrightarrow{*}_{\mathcal{A}} s_{q'b}$  and  $\bar{q}w'c, c' + c'' \xrightarrow{*}_{\mathcal{A}} s_{qa}$  with  $c'' \in C(c_{\parallel})$ . Since  $C(c_{\parallel})$  is the semilinear set containing the unique tuple  $C(c_{\parallel})$ , we get  $c'' = C(c_{\parallel})$ .

By induction hypothesis  $q'b \rightsquigarrow \bar{q}w'(c'_{\parallel})$  with  $C(c'_{\parallel}) = c'$ .

Therefore  $qa \rightsquigarrow q'bc(c_{\parallel}) \rightsquigarrow \bar{q}w'c(c'_{\parallel} \parallel c_{\parallel})$  with  $C(c'_{\parallel} \parallel c_{\parallel}) = c' + C(c_{\parallel})$

4. The last rule of the derivation is  $s_{q''c} \xrightarrow{\epsilon, C(q', b, q'') + C(c_{\parallel})} s_{qa}$   
 $\bar{q}w, c'' \xrightarrow{*} s_{q''c}$  and  $\bar{q}w, c'' + c' \xrightarrow{*} s_{qa}$  with  $c'' \in C(q', b, q'') + C(c_{\parallel})$  i.e.  
 $c'' = c(q', b, q'') + C(c_{\parallel})$  with  $c(q', b, q'') \in C(q', b, q'')$  (same remark on  
 $C(c_{\parallel})$  as in the previous case).

By induction hypothesis  $q''c \xrightarrow{*} \bar{q}w(c'_{\parallel})$  with  $C(c'_{\parallel}) = c''$ .

By definition of  $C(q', b, q'')$ ,  $q'b \xrightarrow{*} q''(c'_{\parallel}, b, q'')$  with  $C(c'_{\parallel}, b, q'') = c(q', b, q'')$ .

By definition the rule  $qa \rightarrow q'bc(c_{\parallel}) \in R$ , hence

$$qa \rightsquigarrow q'bc(c_{\parallel}) \xrightarrow{*} q''c(c'_{\parallel}, b, q'') \parallel c_{\parallel} \xrightarrow{*} \bar{q}w(c'_{\parallel} \parallel c'_{\parallel}, b, q'') \parallel c_{\parallel}$$

Case 2:  $\Leftarrow$  direction.

The proof is by induction on the length of the derivation  $qa \xrightarrow{*} \bar{q}w(\bar{c}_{\parallel})$

1.  $qa \rightsquigarrow q'(c_{\parallel}) \xrightarrow{*} \bar{q}w(\bar{c}_{\parallel})$  where  $qa \rightarrow q'(c_{\parallel}) \in R$ . Since no rewrite can take place on  $q'$  we have  $q' = \bar{q}w$  and  $\bar{c}_{\parallel} = c_{\parallel}$ . Since  $q', C(c_{\parallel}) \rightarrow s_{qa}$  we are done.
2.  $qa \rightsquigarrow q'bc(c_{\parallel}) \xrightarrow{*} \bar{q}w(\bar{c}_{\parallel})$  where  $qa \rightarrow q'bc(c_{\parallel}) \in R$ . Two cases may occur:

- $q'bc(c_{\parallel}) \xrightarrow{*} q''w'c(c_{\parallel} \parallel c'_{\parallel}) = \bar{q}w(\bar{c}_{\parallel})$  with  $q'' = \bar{q}$ ,  $w'c = w$ ,  $c_{\parallel} \parallel c'_{\parallel} = \bar{c}_{\parallel}$ .

Therefore  $q'b \xrightarrow{*} \bar{q}w'(c'_{\parallel})$  and the induction hypothesis applies to this transition sequence.

Therefore  $\bar{q}w', C(c'_{\parallel}) \xrightarrow{*} s_{q'b}$  and  $\bar{q}w'c, C(c'_{\parallel}) + c_{\parallel} \xrightarrow{*} s_{qa}$  by the automaton rule  $s_{q'b} \xrightarrow{c, C(c_{\parallel})} s_{qa}$

- $q'bc(c_{\parallel}) \xrightarrow{*} q''c(c'_{\parallel} \parallel c_{\parallel}) \xrightarrow{*} \bar{q}w(\bar{c}_{\parallel})$ . with  $C(c'_{\parallel}) \in C(q', b, q'')$  and  $\bar{c}_{\parallel} = c'_{\parallel} \parallel c_{\parallel}$  and  $q''c \xrightarrow{*} \bar{q}w(c'_{\parallel})$ .

By induction hypothesis  $\bar{q}w, C(c'_{\parallel}) \xrightarrow{*} s_{q''c}$ .

Since  $C(c_{\parallel}) \in C(q', b, q'')$ , applying rule  $s_{q''c} \xrightarrow{\epsilon, C(q', b, q'') + C(c_{\parallel})} s_{qa}$ , yields  $\bar{q}w, C(c'_{\parallel}) + C(c_{\parallel}) \xrightarrow{*} s_{qa}$ .

□

## 7.5 Backward analysis

We prove that  $Pred^*(L)$  is accepted by a Presburger tree automaton if  $L$  is accepted by a Presburger tree automaton.

### Results for non-commutative parallel

We recall the definitions and results when  $\parallel$  is assumed to be associative but not commutative, which is the initial framework of [BMOT05].

Hedge automata accepts sets of configurations that are seen as ordered-unranked trees. The definition of a top-down hedge automaton  $S = (S, S_I, R \cup R')$  is similar to the definition of a Presburger tree automaton but the rules of

$R'$  have the form  $s \xrightarrow{\parallel} Reg$  where  $Reg$  is a regular expression on the alphabet  $S$ . A language is called a regular hedge language if it is accepted by a hedge automaton. The following result is stated in the aforementioned paper: *Let  $P = (Q, \Sigma, R)$  and let  $L$  be a regular hedge language, then  $Pred^*(L)$  is a regular hedge language*

### Relationship between regular languages and regular hedge languages

The closure  $Cl_{AC}(L)$  of a regular hedge language  $L$  is  $\{t = c2t(c) \mid \exists c' \in L, c \equiv c'\}$ .

Let  $L = L(\mathcal{A})$  for some hedge automaton  $\mathcal{A} = (S, S_I, R \cup R')$ . Then  $Cl_{AC}(L)$  is accepted by the Presburger automaton  $\mathcal{A}_P = (S, S_I, R \cup R'_P)$  where  $s \xrightarrow{\parallel} \phi \in R'_P$  iff there exists a rule  $s \xrightarrow{\parallel} Reg \in R_P$  and  $\phi$  is the Presburger formula defining the Parikh mapping of  $Reg$ .

Let us write  $\rightarrow_A$  when the rules of  $P$  are used with an associative parallel composition and let us write  $\rightarrow_{AC}$  when the rules of  $P$  are used with an associative-commutative parallel composition. Similarly  $Pred^*_A$  (resp.  $Pred^*_{AC}$ ) denotes the set of predecessors in the associative case (resp. associative-commutative case).

**Proposition 21**  $c \xrightarrow{*}_A c'$  iff  $\forall \bar{c} \equiv c, \forall \bar{c}' \equiv c', \bar{c} \xrightarrow{*}_{AC} \bar{c}'$ .

PROOF. By definition of rules, if a rule  $r$  applies to a configuration  $c$ , it applies to any configuration obtained by a permutation of the arguments of  $\parallel$  and the result is equivalent to the application of  $r$  to  $c$ . A straightforward induction on the length of the transition yields the result.  $\square$

A immediate consequence is:

**Proposition 22**  $Cl_{AC}(Pred^*_A(L)) = Pred^*_{AC}(Cl_{AC}(L))$

Given a Presburger regular language  $\bar{L}$ , we can compute a regular hedge language  $L$  such that  $Cl_{AC}(L) = \bar{L}$  (since given a semilinear set  $SL$ , we can easily compute a regular language  $L$  such that  $\#_P(L) = SL$ ).

Since  $Pred^*_A(L)$  is a regular hedge language and since the closure of a regular hedge language is a Presburger regular language, we get:

**Proposition 23** *Let  $L$  be a Presburger regular language. Then  $Pred^*(L)$  is a Presburger regular language.*