International Journal of Foundations of Computer Science
Vol. 22, No. 4 (2011) 843–862
© World Scientific Publishing Company
DOI: 10.1142/S0129054111008453

FORWARD ANALYSIS OF DYNAMIC NETWORK OF PUSHDOWN SYSTEMS IS EASIER WITHOUT ORDER

DENIS LUGIEZ
LIF UMR 6166, Aix-Marseille Université-CNRS, France
denis.lugiez@lif.univ-mrs.fr

Received 22 December 2009
Accepted 30 July 2010
Communicated by Oliver Bournez and Igor Potapov

Dynamic networks of Pushdown Systems ($DNPS$ 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 yields decidability of the forward analysis of $DNPS$. Finally, we extend this result to the model where configurations are sets of threads running in parallel.

Keywords: Concurrent programs; static analysis; weighted tree automata.

1. Introduction

Dynamic networks of Pushdown Systems are models of concurrent programs allowing thread generation and they were introduced for performing the static analysis of such programs [1]. 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 intra-procedural analysis to inter-procedural concurrent analysis [2]) and more and more complex models, from Pushdown system [3, 4] to Process Algebra [5] and networks of Pushdown systems [1], possibly involving data structure or synchronization [6]. In [1], 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. This work has been extended to handle regularity constraints and synchronization in [7]. However, the set of successors of a regular set can be non-regular which seems to preclude the forward analysis of such systems. In this paper, we refine the model by setting the parallel composition as an associative-commutative (AC) operation which expresses the fact that the execution of threads generated by a 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 [8]. The first main result of the paper states that the set of successors of a regular set of configurations can be non-regular but is accepted by a Presburger weighted tree automaton, an extension of Presburger automata which enjoy properties allowing to perform forward analysis. The regularity of the set of predecessors of a regular set is derived from the results of [1]. Then we switch to a more realistic but more complex model where configurations are simply sets of threads running in parallel and don't keep track of how a thread has been generated (which is meaningless when specifying bad configurations). Thanks to our previous study, we prove that forward analysis is still decidable, which is stronger than the result of [9] (which solves the problem *is a term s reachable from a term t* and uses a reachability algorithm in Petri net) but in a weaker model (since they consider ground term rewriting modulo AC).

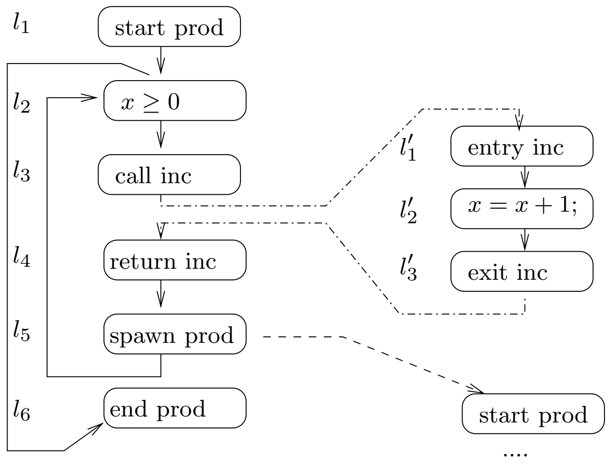
Section 2 presents the basic definitions while section 3.1 introduces pushdown systems and dynamic networks of pushdown systems. Regular sets are defined in section 4. Weighted word and tree automata are defined in section 5 and the computation of the set of successors is explained in section 6 and backward analysis is given in section 7 using known results. The extension to sets of pushdown systems is dealt with in section 8. The conclusion presents some future extensions of this work. Missing proofs can be found in [10].

2. Static Analysis of Programs

Static analysis of programs aims at proving properties of programs by analyzing a model that approximates their behavior. The following toy example presents a variant of the consumer-producer schema and explains how its behavior is described by a transition system derived from the *control flow graph*: a thread called *init* initializes a shared variable x and spawns two threads, *prod* (a producer) and *cons* (a consumer). A producer thread *prod* (described in figure 1) enters a loop that tests the value of x and terminates if $x < 0$, otherwise calls a function *inc* that increments x and spawn a new instance of *prod*. The *cons* thread is similar but calls a function that decrements x and spawns a new instance of *cons*. Figure 1 also gives the control flow graph corresponding to the *prod* thread with the procedure call to *inc*. This graph is incomplete since it doesn't contain a copy of the control flow graph of the thread *prod* generated at program point l_5. The behavior of the process *prod* can be modelled by the following pushdown system with dynamic thread generation:

$$\begin{array}{llll}
(q,l_1) \to (q,l_2) & (q,l_2) \to (q,l_3) & (q,l_2) \to (q,l_6) & (q,l_3) \to (q,l'_1,l_4) \\
(q,l_4) \to (q,l_5) & (q,l_5) \to (q,l_2) \parallel \mathit{prod} & (q,l_6) \to (q,\epsilon) & \\
(q,l'_1) \to (q,l'_2) & (q,l'_2) \to (q,l'_3) & (q,l'_3) \to (q,\epsilon) &
\end{array}$$

where q is the initial state of the system and l_i, l'_i are the program points. For instance, the rule $(q,l_3) \to (q,l'_1,l_4)$ shows that in state q and at program point l_3,



```
thread prod(){
    while (x ≥ 0) {
          inc( );
          spawn(prod);
    }
}


void inc( ){
     x = x + 1;
}
```

Fig. 1. A control flow graph.

the routine $inc()$ is invoked, hence l'_1 is pushed on the stack, but also l_4 since the execution of *prod* will resume at control point l_4 after $inc()$ is executed.

Therefore the inter-procedural analysis of programs generating threads can be done with the formal model of dynamic network of pushdown systems that we describe in the next section. This is used to check properties like *there is a reachable configuration which has more consumers than producers.*

To summarize, the control flow graph of a program models the program as a transition system $\mathcal{S}$ operating on a (usually infinite) set of configurations $\mathcal{C}$ with a transition relation $\rightarrow_S$. Configurations are formal objects (words, trees,...) that describe the current state of the system, and the relation $\rightarrow_S$ is 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 defined as regular languages for some appropriate notion of regularity (enjoying some closure properties and decision of emptiness). *Backward analysis* tests the emptiness of the set $Init \cap Pred^*_S(Bad)$ when *Forward analysis* tests 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, the feasibility of these analyzes boils down to proving that $Pred^*_S(L)$ and $Post^*_S(L)$ are regular when L is regular and providing effective constructions of acceptors for these sets. Forward analysis is more difficult than backward analysis, since regularity is easier to preserve under inverse image that under direct image, see [11] for regular tree languages. In the following, we drop the subscript S of $\rightarrow_S, \ldots$ when S is clear from the context.

3. Dynamic Network of Pushdown Systems

3.1. *Pushdown systems*

A *pushdown system* (PDS in short, see [3] for details) P is a triple (Q, Σ, R) where Q is a finite set of states, Σ is a finite stack alphabet and R is a finite set of transition rules $qa \to q'\gamma$ 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 $\to$ between configurations is the relation defined by $qw \to q'w'$ iff $w = aw''$ and $w' = \gamma.w''$ and there is a rule $qa \to q'\gamma \in R$. 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 [12, 3]. Actually the relation $\mathcal{R}$ on pairs of configurations defined by $qw \ \mathcal{R}\ q'w'$ iff $qw \xrightarrow{*} q'w'$ is a rational relation (see [13] for extensions).

3.2. *Modelling concurrent threads*

Dynamic networks of pushdown processes [1] 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 child of the process. There is no limitation in the creation of processes (a process has an arbitrary number of childs) and in the recursion depth for process creation (each process may create children that may create childs, ...). To have a uniform presentation, we shall assume that states and stack symbols are unary symbols, and that there is a unique constant #, which allows to identify a word $qa_1 \dots a_n$ to a term $q(a_1(\dots(a_n(\#))))$. For simplicity, we still use the word notation $qa_1 \dots a_n$ to denote $q(a_1(\dots(a_n(\#))))$. Then the signature is extended by a parallel composition symbol $\|$ which is associative-commutative. This means that configurations are unranked-unordered trees.

The set $DNPS$ of configurations and the set $DNPS_{\|}$ of parallel configurations are defined by:

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

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

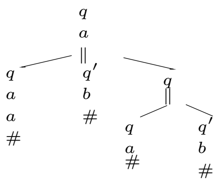
$$\begin{array}{lll} qw \equiv q'w' & \text{iff} & q = q' \text{ and } w = w' \\ qw(c_1 \parallel \dots \parallel c_n) \equiv q'w'(c'_1 \parallel \dots \parallel c'_m) & \text{iff} & q = q', w = w', n = m \text{ and } \exists \sigma \\ & & \text{permutation of } \{1, \dots, n\} \\ & & \text{such that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

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

$Sub(qw) = \{qw\}$

$Sub(qw(c_1 \parallel \dots \parallel c_n)) = \{qw(c_1 \parallel \dots \parallel c_n)\} \cup \{qw\} \cup \bigcup_{i=1,\dots,n} Sub(c_i)$

Example 1. *Let* $c = qa(qaa \parallel q'b \parallel q(qa \parallel q'b))$*. This configuration is the tree*



where all vertical lines $qa, qaa, q'b, q, qa, q'b$ represent PDS *running in parallel. We have* $Sub(c) = \{qa(qaa \parallel q'b \parallel q(qa \parallel q'b)), qaa, q'b, q(qa \parallel q'b), qa\}$ *and* $c \equiv qa(q'b \parallel q(q'b \parallel qa) \parallel qaa)$*.*

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

3.3. *DNPS and their transition relation*

A *dynamic network of pushdown systems* ($DNPS$ in short) P is a triple (Q, Σ, R) where Q is a finite set of states, Σ is a finite alphabet and R is a finite set of rules of the form $qa \to q'\gamma$ or $qa \to q'\gamma(q_1\gamma_1 \parallel \ldots \parallel q_n\gamma_n)$ where γ, γ_i for $i = 1, \ldots, n$ belong to Σ^*, q, q', q_i for $i = 1, \ldots, n$ belong to Q. The relation $\to$ induced by P on pairs of configurations is defined by:

- if $qa \to q'\gamma \in R$ then $qaw \to q'\gamma w$
- if $qa \to q'\gamma(q_1\gamma_1 \parallel \ldots \parallel q_n\gamma_n) \in R$ then
  - $qaw \to q\gamma w(q_1\gamma_1 \parallel \ldots \parallel q_n\gamma_n)$
  - $qaw(s_1 \parallel \ldots \parallel s_m) \to q\gamma w(q_1\gamma_1 \parallel \ldots \parallel q_n\gamma_n \parallel s_1 \ldots \parallel s_m)$
- if $c \to c'$ *and* $c \in Sub(\bar{c})$, then $\bar{c} = C[c] \to \bar{c}' = C[c']$,
- if $c \to c'$ and $\bar{c} \equiv c, \bar{c'} \equiv c'$ then $\bar{c} \to \bar{c'}$.

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

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

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 3. $qw(t_1 \parallel \ldots \parallel t_n) \xrightarrow{*} q'w'(u_1 \parallel \ldots \parallel u_m)$ *iff* $qw(t_1 \parallel \ldots \parallel t_n) \xrightarrow{*} q'w'(t_1 \parallel \ldots \parallel t_n \parallel s_{n+1} \parallel \ldots \parallel s_m) \xrightarrow{*} q'w'(u_1 \parallel \ldots \parallel u_m)$

4. Regular Sets of *DNPS* Configurations

Our analysis relies on regular languages and we describe now how to extend the notion of regular tree languages to configurations. Configurations are actually unordered unranked trees, therefore we shall use the name *tree* (and letters $t, t', \ldots$) as well as the name *configuration* (and letters $c, c', \ldots$).

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 [14]. Word regular languages and tree regular languages have been generalized for unranked-unordered trees using Presburger automata [8]. To fit the framework of $DNPS$, we slightly change the definitions and we use *top-down automata* instead of bottom-up automata.

Definition 4. *A top-down Presburger automaton is a tuple* $\mathcal{A} = (Q \cup \Sigma, S, S_I, R \cup R')$ *where:*

- $S = \{s_1, \ldots, s_p\}$ *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* $s \xrightarrow{\|} \varphi(x_1, \ldots, x_p)$ *where* $\varphi(x_1, \ldots, x_p)$ *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 child of N is labelled by s' such that the rule $s, a \rightarrow s'$ belongs to R,
- if $\|$ is the symbol of a node N of t and N is labelled by $r(N) = s$, if $N_1, \ldots, N_m$ are the children of N, then
  - (i) there are n_i children of N labelled by s_i for $1 \leq i \leq p$,
  - (ii) there is a rule $s \xrightarrow{\|} \varphi(x_1, \ldots, x_p) \in R'$, such that $\varphi(n_1, \ldots, n_p)$ is true.

A tree t is accepted by $\mathcal{A}$ if there is a successful run r of $\mathcal{A}$ on t, i.e. r labels the root of t 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 5. *Let* $q(a^*(\ldots))$ *denote any* $q(a^n(\ldots))$ *for* $n \geq 0$*. The set of trees of the form* $q(a^*(\| \ (q(b^*(\#)), \ldots, q(b^*(\#)), q(c^*(\#)), \ldots, q(c^*(\#))))$ *where the parallel operator* $\|$ *has as many arguments* $q(b^*(\#))$ *as arguments* $q(c^*(\#))$ *is a Presburger regular language accepted by the top-down Presburger automaton* $\mathcal{A} = (Q \cup \Sigma = \{q, a, b, c\}, S = \{s_a, s_b, s_c\}, S_I = \{s_a\}, R = \{s_a \xrightarrow{q} s_a; s_a \xrightarrow{a} s_a; s_b \xrightarrow{q} s_b; s_b \xrightarrow{b} s_b; s_c \xrightarrow{q} s_c; s_c \xrightarrow{c} s_c; s_b \rightarrow \#; s_c \rightarrow \#\}$,

$R' = \{s_a \overset{\|}{\rightarrow} x_a = 0 \wedge x_b = x_c\}$) *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 [8].

Given a *DNPS* $P = (Q, \Sigma, R)$, we may ask if $Post^*(L)$ is regular for a regular language L. The following proposition shows that regular languages are too weak.

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

A counter-example is given in the following example:

Example 7. *Let* $P = (\{q\}, \{a, b, c\}, \{qa \rightarrow qaa \parallel (qb, qc), qb \rightarrow qbb, qc \rightarrow qcc\})$. *The set* $Post^*(qa)$ *is the language*

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

which is non-regular (by a standard pumping argument).

5. Weighted Tree Automata for *DNPS*

In the following, we extend Presburger tree automata into more expressive tree automata that suit our purpose.

5.1. *Semilinear sets*

Let $b \in \mathbb{N}^m$, let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be a finite subset of $\mathbb{N}^m$. The linear set $L(b, \mathcal{P})$ of $\mathbb{N}^m$ is the set $\{b + \Sigma_{i=1}^{i=n} \lambda_i p_i \mid \lambda_i \in \mathbb{N}\}$. 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 $L^0 = \{(0, \ldots, 0)\}$, and $L^n = \underbrace{L + \ldots + L}_{n}$. The set of rational expressions of semilinear sets is defined by

$$Rat ::= L \mid Rat + Rat \mid Rat \cup Rat \mid Rat^*$$

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

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

The next proposition states classical results about semilinear sets (see chapter 3 of [15] for instance) that are used throughout the paper.

Proposition 8. *Let $R \in Rat$. Then there exists an 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 $\phi(x_1, \ldots, x_p)$ is true is an effectively constructible semilinear set [16]. 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 $\mathcal{SL}_m$ be the set of semilinear sets of $\mathbb{N}^m$. Then $\mathcal{S}_m = (\mathcal{SL}_m, \cup, +, \emptyset, (0, \ldots, 0))$ is a semiring. Weighted automata are word automata where the transitions are labelled by element of a semiring K. Weighted word automata have already used for PDS analysis provided that K satisfies additional properties [4]. 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 $DNPS$.

Definition 9. *A Presburger weighted word automaton is an automaton* $\mathcal{A} = (\Sigma, S, s_0, S_F, \mathcal{S}_m, \Delta)$ *where* S_F *is a set of pairs* (s, ϕ_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* Δ *is denoted* $s \overset{a,C}{\rightarrow} s'$.

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

- $\epsilon, (0, \ldots, 0) \xrightarrow{*}_{\mathcal{A}} s_0$
- *if* $w, c \xrightarrow{*}_{\mathcal{A}} s$ *then* $wa, c + c' \xrightarrow{*}_{\mathcal{A}} s'$ *if there is a rule* $s \overset{a,C'}{\rightarrow} s'$ *with* $c' \in C'$,
- *if* $w, c \xrightarrow{*}_{\mathcal{A}} s$ *then* $w, c + c' \xrightarrow{*}_{\mathcal{A}} s'$ *if there is a rule* $s \overset{\epsilon,C'}{\rightarrow} s'$ *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 10. *For each* $s \in S$, *the set* $L(s_0, s) = \{c \in \mathcal{SL}_m \mid \exists w \in \Sigma^*\ w, c \xrightarrow{*}_{\mathcal{A}} s\}$ *is an effectively computable semilinear set.*

Proof. By proposition 8, 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 $L(s_0, s) = \{c \in \mathcal{SL}_m \mid \exists w \in \Sigma^*\ w, c \xrightarrow{*}_{\mathcal{A}} s\}$. □

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

5.3. *Presburger weighted tree automata*

Presburger weighted tree automata are designed to accept sets of configurations of $DNPS$.

Definition 11. *A top-down Presburger weighted tree automaton is a tuple $\mathcal{A} = (Q \cup \Sigma, S, S_I, \mathcal{S}_m, R \cup R')$ where:*

- *$S = \{s_1, \ldots, s_p\}$ is a finite set of states,*
- *$S_I \subseteq S$ is a set of initial states,*
- *$\mathcal{S}_m$ is the semiring $(\mathcal{SL}_m, \cup, +, \emptyset, (0, \ldots, 0))$,*
- *R is a set of rules of the form $s \overset{(0,\ldots,0)}{\rightarrow} \#$ or $s \overset{a,C}{\rightarrow} s'$ where $s, s' \in S, a \in Q \cup \Sigma \cup \{\epsilon\}, C \in \mathcal{S}_m$*
- *R' is a set of rules $s \overset{\|}{\rightarrow} \phi(x_1, \ldots, x_p, y_1, \ldots, y_m)$ where $\phi(x_1, \ldots, x_p, y_1, \ldots, y_m)$ 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 $r(N) = (s, (0, \ldots, 0))$ such that the rule $s \overset{(0,\ldots,0)}{\rightarrow} \#$ belongs to R and the parent node of N is labelled by $(\_, (0, \ldots, 0))$.
- 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 child of N is labelled by $(s', c + c')$ s.t. there is a sequence of rules $s_1 \overset{\epsilon,C_1}{\rightarrow} s_2, \ldots, s_{i-1} \overset{\epsilon,C_{i-1}}{\rightarrow} s_i, s_i \overset{a,C}{\rightarrow} s_{i+1}, s_{i+2} \overset{\epsilon,C_{i+2}}{\rightarrow} s_{i+3}, \ldots, s_n \overset{\epsilon,C_n}{\rightarrow} s_{n+1}$ where $s_1 = s, s_{n+1} = s'$ and $c' = c_1 + \ldots + c_n$ with $c_i \in C_i$ for $i = 1, \ldots, n$.
- if $\|$ is the symbol of a node N of t and N is labelled by $r(N) = (s, (c_1, \ldots, c_m))$, if $N_1, \ldots, N_n$ are the children of N, then
  - (i) there are n_i children of N labelled by $(s_i, (0, \ldots, 0))$ for $1 \leq i \leq p$,
  - (ii) there is a rule $s \overset{\|}{\rightarrow} \phi(x_1, \ldots, x_p, y_1, \ldots, y_m) \in R'$, such that $\phi(n_1, \ldots, n_p, c_1, \ldots, c_m)$ is true.

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

Example 12. *Let $q(b^+(\#))$ denote $q(b^n(\#))$ with $n > 0$. The (non-regular) language L*

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

is accepted by the Presburger weighted tree automaton $\mathcal{A} = (Q \cup \Sigma, S, S_I, \mathcal{S}_2, R \cup R')$

$Q \cup \Sigma = \{q, a, b, c\}$

$S = \quad \{s_0, s_{qa}, s_{qb}, s_{qc}, s_a, s_b, s_c, s_q\}, S_I = \{s_0\},$

$R = \quad \{s_0 \overset{q,(0,0)}{\rightarrow} s_q, s_q \overset{a,(0,0)}{\rightarrow} s_a, s_a \overset{a,(1,1)}{\rightarrow} s_a, s_{qb} \overset{q,(0,0)}{\rightarrow} s_b, s_b \overset{b,(0,0)}{\rightarrow} s_b,$
$\quad s_b \overset{(0,0)}{\rightarrow} \#, s_{qc} \overset{q,(0,0)}{\rightarrow} s_c, s_c \overset{c,(0,0)}{\rightarrow} s_c, s_c \overset{(0,0)}{\rightarrow} \#, \},$

$R' = \quad \{s_a \rightarrow x_{s_{qb}} = y_1 \wedge x_{s_{qc}} = y_2 \wedge \bigwedge_{s \neq s_{qb}, s_{qc}} x_s = 0$

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 child of this node by s_{qb}, the second child 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 children (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 Ł 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 13. *Let* $\mathcal{A}$ *be a Presburger weighted tree automaton.*

- *There is a Presburger weighted tree automaton* $\mathcal{B}$ *without* ϵ*-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 a 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 *DNPS*

The next step is to show that $Post^*(L)$is accepted by a Presburger weighted tree automaton.

6.1. *Preliminary computations*

Firstly, we simplify the computation of $Post^*(L)$ by generalizing a result on PDS stating that $Post^*_P(L)$ can be computed as the set of successors of a single 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 14. *Let* $L \subseteq \mathcal{C}$ *be a regular language. There exists* $P' = (Q \cup Q', \Sigma \cup \{\$\}, R \cup R')$ *such that* $Post^*_P(L) = Post^*_{P'}(q_0\$) \cap \mathcal{C}$ *where* $q_0 \in Q'$.

Proof. Let $\mathcal{A} = (Q \cup \Sigma, S, S_I, \mathcal{R} \cup \mathcal{R}')$ be a top-down Presburger weighted tree automaton accepting $L' = L$. For each rule $s \overset{\|}{\rightarrow} \phi(x_1, \ldots, x_p) \in \mathcal{R}'$ we compute the semilinear set $L_{s,\phi}$ of $\mathbb{N}^m$ equivalent to $\phi(x_1, \ldots, x_p)$. This set a finite union of linear sets $SL^i_{s,\phi} = L(b_i, P_i)$.

The $DNPS$ 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 $s' \xrightarrow{\|} \phi(x_1, \ldots, x_p)$, then Q' contains $q(s, s', \|, \phi)$ and $q(s, s', \|, L(b, \mathcal{P}))$.
- The set of rules R' is defined by:
  - P' chooses non-deterministically to generate a configuration qw or a configuration $qw(\ldots \| \ldots \| \ldots)$. Furthermore a run of $\mathcal{A}$ labels the root of qw by s and the last node of qw by s'. The corresponding rules are:
    - $q_0\$ \to q(s, s')\$$, meaning: generate qw such that a run of $\mathcal{A}$ labels the root of $q(w(\#)$ by s,
    - $q_0\$ \to q(s, s', \|, \phi)\$$, meaning: generate $qw(t_1 \| \ldots \| t_m)$ such that a run of $\mathcal{A}$ labels the root of $q(w(\| (t_1, \ldots, t_m)$ by s,
  - 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)
    - $q_0(s)\$ \to q(s, s')\$$,
    - $q_0(s)\$ \to q(s, s', \|, \phi)\$$,
  - $q(s, s')\$ \to q(s, s'')\$a$ if $s'' \xrightarrow{a} s' \in \mathcal{R}$. This rule simulates the automaton rule $s'' \xrightarrow{a} s'$ in a backward way.
  - $q(s, s')\$ \to q$ if $\mathcal{A}$ contains a rule $s \xrightarrow{q} s'$.
  - if there is an automaton rule $s' \xrightarrow{\|} \phi(x_1, \ldots, x_p)$ and $L(b, \mathcal{P})$ is a linear set associated to ϕ, the $DNPS$ P simulates the generation of the parallel composition with the rules:
    - $q(s, s', \|, \phi)\$ \to q(s, s', \|, L(b, \mathcal{P}))\$(q_0(S)^b)$
    - $q(s, s', \|, L(b, \mathcal{P}))\$ \to q(s, s', \|, L(b, \mathcal{P}))\$(q_0(S)^p)$ for any $p \in \mathcal{P}$,

    where $q_0(S)^b$ for $b = (b_1, \ldots, b_p)$ (resp. $q_0(S)^p$ for $p = (p_1, \ldots, p_p)$) denotes $q_0(s_1)\$^{b_1} \| \ldots \| q_0(s_p)\$^{b_p}$ (resp. $q_0(s_1)\$^{p_1} \| \ldots \| q(s_p)\$^{p_p}$) where (i) $q(s_i)\$^l$ is a parallel composition of l instances $q(s_i)\$$, (ii)$S = \{s_1, \ldots, s_p\}$.
- $q(s, s', \|, L(b, \mathcal{P}))\$ \to q(s, s')\$$. This rules stops the generation of arguments of $\|$ 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, a routine proof by induction yields $c \in Post^*_{R'}(q_0\$) \cap \mathcal{C}$ iff $c \in L$. □

The next proposition states that we can use a $DNPS$ with simpler rules.

Proposition 15. *There exists a DNPS P' such that (i) $Post^*_P(qa) = Post^*_{P'}(qa) \cap \mathcal{C}$ (ii) the rules of P' have the form $qa \to q'$ or $qa \to q'bc$ or $qa \to q'(q_1a_1 \parallel \ldots \parallel q_ma_m)$ or $qa \to q'bc(q_1a_1 \parallel \ldots \parallel q_ma_m)$*

Proof. (Sketch) A rule $qa \to q'a_1 \ldots a_n$ is replaced by rules $qa \to \bar{q}_{n-1}\#a_n$, $\bar{q}_{n-1}\# \to \bar{q}_{n-2}a_{n-1}$,..., $\bar{q}_1\# \to q'a_1$ where the $\bar{q}_i$ and $\#$ are new symbols. This can be done for all rules and yields a new $DNPS$ P' of size linear in the size of the initial $DNPS$ P. By construction $Post^*_P(L) = Post^*_{P'}(L) \cap \mathcal{C}$. □

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 $C(qa) = (0, \ldots, 0, 1, 0, \ldots, 0)$. Therefore a parallel composition $c_\parallel = q_{i_1}a_{j_1} \parallel \ldots \parallel q_{i_k}a_{j_k}$ can be identified with a tuple $C(c_\parallel)$ of $\mathbb{N}^m$.

Example 16. *For the DNPS 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 $\to$ 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 $\overset{*}{\rightsquigarrow}$.

The next construction is inspired by [1] and computes, for each q, a, q', a context-free grammar G such that the Parikh mapping of $L(G)$ is the set $\{C(c_\parallel) \mid qa \overset{*}{\rightsquigarrow} q'(c_\parallel)\}$, i.e. it describes all possible parallel compositions generated by transitions $qa \overset{*}{\rightsquigarrow} q'(\ldots)$. In the following, we shall identify the i^{th} element qa to the letter l_i of a alphabet $\{l_1, \ldots, l_m\}$. We recall that the *Parikh mapping* $\#_P(w)$ of a word w is the tuple $(n_1, \ldots, n_m) \in \mathbb{N}^m$ such that the letter l_1 has n_1 occurrences in w,..., the letter l_m has n_m occurrences in w.

- The set of terminals is $\{l_1, \ldots, l_m\}$. In the following, $w(c_\parallel)$ denotes the word $l_1^{n_1} \cdot \ldots \cdot l_m^{n_m}$ where $(n_1, \ldots, n_m) = C(c_\parallel)$ (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 $qa \overset{*}{\to} q'(c_\parallel)$.
- The set Δ of production rules is defined by
  - if $qa \to q'(c_\parallel)$ belongs to R, then $X_{q,a,q'} \to w(c_\parallel) \in \Delta$,
  - if $qa \to \bar{q}bc(c_\parallel)$ belongs to R , then $X_{q,a,q'} \to X_{\bar{q},b,\tilde{q}}X_{\tilde{q},c,q'}w(c_\parallel) \in \Delta$,

Proposition 17. *For all $q, q' \in Q, a \in \Sigma$, $X_{q,a,q'} \overset{*}{\to}_G w$ iff $qa \overset{*}{\rightsquigarrow} q'(c_\parallel)$ and $C(c_\parallel) = \#_P(w)$*

Parikh's theorem and the results of [17] yield that:

Proposition 18. *The set* $C(q,a,q') = \{C(c_{\|}) \mid qa \overset{*}{\rightsquigarrow} q'(c_{\|})\}$ *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 $\mathcal{A} = (Q \cup \Sigma, S, s_0, S_F, \mathcal{S}_m, \Delta)$ such that $qa \overset{*}{\rightsquigarrow} q'w(c_{\|})$ iff $\mathcal{A}$ accepts $q'w, C(c_{\|})$.

- $S = \{s_0\} \cup \{s_q \mid q \in Q\} \cup \{s_{qa} \mid s \in Q, a \in \Sigma\}$, $S_F = \{(s_{qa}, \mathbb{N}^m)\}$. 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 $qa \overset{*}{\rightsquigarrow} q'w(c_{\|})$ and $c = C(c_{\|})$.
- Δ is defined by:
  - $s_0 \overset{q,(0,\dots,0)}{\rightarrow} s_q \in \Delta$ and $s_q \overset{a,(0,\dots,0)}{\rightarrow} s_{qa} \in \Delta$
  - if R contains the rule $qa \to q'(c_{\|})$ then $s_{q'} \overset{\epsilon,C(c_{\|})}{\rightarrow} s_{qa} \in \Delta$
  - if R contains the rule $qa \to q'bc(c_{\|})$ then
    - $s_{q'b} \overset{c,C(c_{\|})}{\rightarrow} s_{qa} \in \Delta$
    - $s_{q''c} \overset{\epsilon,C(q',b,q'')+C(c_{\|})}{\rightarrow} s_{qa} \in \Delta$

Proposition 19. $\bar{q}w, c \overset{*}{\rightarrow}_{\mathcal{A}} s_{qa}$ *iff* $qa \overset{*}{\rightsquigarrow} \bar{q}w(c_{\|})$ *and* $c = C(c_{\|})$.

Proof.

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

(1) The last rule of the derivation is $s_0 \overset{q,(0,\dots,0)}{\rightarrow} s_q$ or $s_q \overset{a,(0,\dots,0)}{\rightarrow} s_{qa}$. Straightforward.

(2) The last rule of the derivation is $s_{q'} \overset{\epsilon,C(c_{\|})}{\rightarrow} s_{qa}$ Then $\bar{q}w = q'$ with $qa \to q'(c_{\|}) \in R$ hence $qa \rightsquigarrow q'(c_{\|})$.

(3) The last rule of the derivation is $s_{q'b} \overset{c,C(c_{\|})}{\rightarrow} s_{qa}$.

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

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

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

(4) The last rule of the derivation is $s_{q''c} \overset{\epsilon,C(q',b,q'')+C(c_{\|})}{\rightarrow} s_{qa}$

$\bar{q}w, c'' \overset{*}{\rightarrow}_{\mathcal{A}} s_{q''c}$ and $\bar{q}w, c'' + c' \rightarrow_{\mathcal{A}} s_{qa}$ with $c'' \in C(q',b,q'') + C(c_{\|})$ i.e. $c'' = c(q',b,q'') + C(c_{\|})$ with $c(q',b,q'') \in C(q',b,q'')$ (same remark on $C(c_{\|})$ as in the previous case).

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

By definition of $C(q',b,q'')$, $q'b \overset{*}{\rightarrow} q''(c_{\|}^{q',b,q''})$ with $C(c_{\|}^{q',b,q''}) = c(q',b,q'')$.

By definition the rule $qa \rightarrow q'bc(c_\|) \in R$, hence
$qa \rightsquigarrow q'bc(c_\|) \overset{*}{\rightsquigarrow} q''c(c_\|^{q',b,q''} \parallel c_\|) \overset{*}{\rightsquigarrow} \bar{q}w(c''_\| \parallel c_\|^{q',b,q''} \parallel c_\|)$

Case 2: $\Leftarrow$ direction.
The proof is by induction on the length of the derivation $qa \overset{*}{\rightsquigarrow} \bar{q}w(\bar{c}_\|)$

(1) $qa \rightsquigarrow q'(c_\|) \overset{*}{\rightsquigarrow} qw(\bar{c}_\|)$ where $qa \rightarrow q'(c_\|) \in R$. Since no rewrite can take place on q' we have $q' = \bar{q}w$ and $\bar{c}_\| = c_\|$. Since $q', C(c_\|) \rightarrow_{\mathcal{A}} s_{qa}$ we are done.

(2) $qa \rightsquigarrow q'bc(c_\|) \overset{*}{\rightsquigarrow} \bar{q}w(\bar{c}_\|)$ where $qa \rightarrow q'bc(c_\|) \in R$. Two cases may occur:

- $q'bc(c_\|) \overset{*}{\rightsquigarrow} q''w'c(c_\| \parallel c'_\|) = \bar{q}w(\bar{c}_\|)$ with $q'' = \bar{q}, w'c = w, c_\| \parallel c'_\| = \bar{c}_\|$.
  Therefore $q'b \overset{*}{\rightsquigarrow} \bar{q}w'(c'_\|)$ and the induction hypothesis applies to this transition sequence.
  Therefore $\bar{q}w', C(c'_\|) \overset{*}{\rightarrow}_{\mathcal{A}} s_{q'b}$ and $\bar{q}w'c, C(c'_\|) + c_\| \overset{*}{\rightarrow}_{\mathcal{A}} s_{qa}$ by the automaton rule $s_{q'b} \overset{c,C(c_\|)}{\rightarrow} s_{qa}$
- $q'bc(c_\|) \overset{*}{\rightsquigarrow} q''c(c'_\| \parallel c_\|) \overset{*}{\rightsquigarrow} \bar{q}w(\bar{c}_\|)$ with $C(c'_\|) \in C(q', b, q'')$ and $\bar{c}_\| = c''_\| \parallel c'_\| \parallel c_\|$ and $q''c \overset{*}{\rightsquigarrow} \bar{q}w(c''_\|)$.
  By induction hypothesis $\bar{q}w, C(c''_\|) \overset{*}{\rightarrow}_{\mathcal{A}} s_{q''c}$.
  Since $C(c_\|) \in C(q', b, q'')$, applying rule $s_{q''c} \overset{\epsilon, C(q',b,q'')+C(c_\|)}{\rightarrow} s_{qa}$, yields $\bar{q}w, C(c''_\|) + C(c'_\|) + C(c_\|) \overset{*}{\rightarrow}_{\mathcal{A}} s_{qa}$. □

The previous Presburger weighted word automaton $\mathcal{A} = (Q \cup \Sigma, S, s_0, S_F, \mathcal{S}_m, R)$ is extended into a Presburger weighted tree automaton $\mathcal{B} = (Q \cup \Sigma, S_{\mathcal{B}}, S_I, \mathcal{S}_m, R_{\mathcal{B}})$ 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 $\mathcal{S}_{\mathcal{B}}$ are ordered $s_1, \ldots, s_m$ where $s_i = s_0^{qa}$ iff i is the i^{th} component of $\mathcal{S}_p$ (i.e. corresponds to the ordering of qa defined for the word automaton $\mathcal{A}$).
- The set of rules contains
  - $s^{qa} \overset{a,C}{\rightarrow} s'^{qa}$ if $\mathcal{A}$ contains the rule $s \overset{a,C}{\rightarrow} s'$.
  - $s_{qa}^{qa} \overset{(0,\ldots,0)}{\rightarrow} \#$, this rule is needed since $\mathcal{A}$ deals with words and $\mathcal{B}$ deals with trees,
  - $s_{qa}^{qa} \overset{\|}{\rightarrow} \bigwedge_{i=1}^{i=m} x_i = y_i \wedge \bigwedge_{i>m} x_i = 0$

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

Proposition 20. *$L(\mathcal{B}) = Post^*(qa)$.*

Proof. The proof is by structural induction on the tree structure. We prove that for any q, a, $t \in Post^*(qa)$ 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 19, we have $q(w(\#) \in Post^*(qa)$ iff a run of $\mathcal{B}$ labels $q(w(\#))$ by $(s_0^{qa}, (0,\ldots,0))$ at the root and q_{qa}^{qa} at the leaf.

Induction step. $t = q(w(\| \ (t_1,\ldots,t_n)))$. We assume that there is a run of $\mathcal{B}$ labelling each t_i by $(s_0^{qa}, (0,\ldots,0))$ iff $t_i \in Post^*(qa)$. By definition of the rules of $\mathcal{B}$ and adapting the proof of proposition 19, $qa \overset{*}{\rightsquigarrow} t$ there is a rule labelling the node $\|$ by $(s_{qa}^{qa}, (n_1,\ldots,n_m))$ such that n_1 trees t_i are labelled by $(s_0^{(qa)_1}, (0,\ldots,0))$,..., n_m trees t_i are labelled by $(s_0^{(qa)_m}, (0,\ldots,0))$ 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 $t \in Post^*(qa)$. □

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

7. Backward Analysis of *DNPS*

To perform backward analysis, we can rely on the results of [1] 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 21. *The set $Pred^*(L)$ is accepted by a Presburger tree automaton.*

8. A More Realistic Model of DNPS : the Flat Model

Since threads are thin processes that actually share some common resources, one may argue against our model of configuration: a representation closer to the reality would describe a configuration as a parallel composition of an unbounded number of threads. In our model, a configuration keeps the relationship between a thread and its childs, which is usually not relevant. For instance, a configuration $qa(qb \parallel qc)$ contains the information that the process qa has generated two processes qb and qc when a more natural representation is $qa \parallel qb \parallel qc$. Therefore, the configurations of $Post^*(L)$ should be *flattened* before computing the intersection with the set of bad configurations. Unfortunately, example 7 shows that this flattening may yield non-regular languages. Therefore, we shall proceed the other way around: we add an additional constraint to Presburger weighted tree automata that is used to test if a configuration of $Post^*(L)$ has a flattened version that belongs to a regular set of (flat) bad configurations.

8.1. *Flattening configurations*

Firstly, we define the set of flat configurations:

$$FDNPS \ni c ::= q_1 w_1 \parallel \ldots \parallel q_n w_n \quad n \geq 1$$

Contrary to the initial notion of configuration, a flat configuration is equivalent to a set of PDS configurations since the $\|$ operation is associative-commutative.

A *(flat) dynamic network of pushdown systems* ($FDNPS$ in short) P is a triple (Q, Σ, R) where Q is a finite set of states, Σ is a finite alphabet and R is a finite set of rules of the form $qa \to q'\gamma$ or $qa \to q'\gamma \parallel q_1\gamma_1 \parallel \dots \parallel q_n\gamma_n$ where γ, γ_i for $i = 1, \dots, n$ belong to Σ^*, q, q', q_i for $i = 1, \dots, n$ belong to Q. The transition relation between configurations is defined by:

$$q_1w_1 \parallel \dots \parallel q_iw_i \parallel \dots \parallel q_pw_p \to q_1w_1 \parallel \dots \parallel q'_i\gamma w'_i \parallel q'_1\gamma_1 \dots q'_n\gamma_n \parallel \dots \parallel q_pw_p$$

if $q_iw_i = qaw'_i$ and R contains a rule $qa \to q'_i\gamma \parallel q'_1\gamma_1 \parallel \dots \parallel q'_n\gamma_n$.

Example 22. *Let* $P = (\{q, q'\}, \{a, b\}, \{qa \to q'bb, q'b \to qab \parallel qa \parallel qa\})$ *be a* $FDNPS$. *A sequence of transition of* P *is:*

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

We state the relationship between the tree model and the flat model. The flattening $flat$ operation transforms a configuration in the tree model into a configuration in the flat model as follows:

$$\begin{aligned} &flat(qw) = qw \\ &flat(qw(c_1 \parallel \dots \parallel c_p)) = qw \parallel flat(c_1) \parallel \dots \parallel flat(c_p) \end{aligned}$$

Let $P = (Q, \Sigma, R)$ be a DNPS in the tree model and let $flat(P) = (Q, \Sigma, flat(R))$ be defined by $qa \to q\gamma(c_\parallel) \in R$ iff $qa \to flat(q\gamma(c_\parallel)) \in flat(R)$. Then the following proposition holds:

Proposition 23. $C \xrightarrow{*}_P C'$ *iff* $flat(C) \xrightarrow{*}_{flat(P)} flat(C')$

The proof is a straightforward structural induction on configurations.

8.2. *Forward analysis for flat languages*

Firstly, we show how to compute some additional information from runs. Let r be a run of a Presburger weighted tree automaton $\mathcal{A} = (Q \cup \Sigma, S, S_I, \mathcal{S}_m, R \cup R')$ on a configuration t where $S = \{s_1, \dots, s_p\}$. Let $\#_r(s)$ be the number of times r labels by s the root of a subterm $qw(\dots)$ of t. Then each run r defines a tuple $\#_r = (\#_r(s_1), \dots, \#_r(s_p))$.

Proposition 24. *The set* $\#(\mathcal{A}) = \{\#_r \mid r\ successful\ run\ of\ \mathcal{A}\}$ *is a semilinear set of* $\mathbb{N}^p$ *definable by a Presburger formula.*

Proof. The proof idea is to construct a context-free grammar G such that the Parikh mapping of $L(G)$ is $\{\#_r \mid r\ successful\ run\ of\ \mathcal{A}\}$.

Firstly, we recall some properties of weighted tree automata and semilinear sets.

Let $L(s, s')$ be the set of weights $(n_1, \ldots, n_m)$ such that there is a run r of $\mathcal{A}$ on a term qw labelling the root of qw by $(s, (0, \ldots, 0))$ and the leaf by $(s', (n_1, \ldots, n_m))$. A proof similar to the proof of proposition 10 yields that $L(s, s')$ is a semilinear set.

Let $L = L(b, \mathcal{P} = \{p_1, \ldots, p_n\})$ be a linear set and V be the alphabet $\{S_1, \ldots, S_p\}$. Let w_b (resp. w_i) be a word of V^* such that the Parikh mapping of w_b is b (resp. p_i). Let G_L be the grammar: $X_L \to w_b \mid w_b X_{\mathcal{P}}$
$X_{\mathcal{P}} \to w_i \mid w_i X_{\mathcal{P}}\ i = 1, \ldots, n$
The definition of G_L yields that the Parikh mapping of $L(G_L)$ is $L(b, \mathcal{P})$. This construction is easily extended to get a grammar G_L for a semilinear set L.

The second step is to define the grammar G.

- The alphabet of the grammar G is $\Sigma = \{1_1, \ldots, 1_p\}$.
- The set of non-terminal symbols contains an axiom S, the set $\{S_1, \ldots, S_p\}$ and the non-terminals of grammars G_L for semilinear sets L related to the $L(s, s')$'s.
- The set of rules R_G is defined as follows:
  - $S \to S_i \in R_G$ for each S_i.
  - For each pair $s_i, s_j \in S$, $s_j \overset{(0,\ldots,0)}{\rightarrow} \# \in R$ and $(0, \ldots, 0) \in L(s_i, s_j)$ then $S_i \to 1_i \in R_G$.
  - For each pair $s_i, s_j \in S$, such that $s_j \overset{\parallel}{\rightarrow} \phi(x_1, \ldots, x_p, y_1, \ldots, y_m) \in R$, let L be the semilinear set of tuples $(n_1, \ldots, n_p)$ validating the formula $\exists y_1, \ldots, y_m\ ((y_1, \ldots, y_m) \in L(s_i, s_j) \wedge \phi(x_1, \ldots, x_p, y_1, \ldots, y_m))$. Then all rules of the context-free grammar G_L defining L are in R_G and $S_i \to 1_i X_L \in R_G$ where X_L is the axiom of G_L.

We prove that $S_i \overset{*}{\to} w \in \Sigma^*$ iff there is a run r on a configuration t such that r labels the root of t by s_i and the Parikh mapping of w is $(\#_r(s_1), \ldots, \#_r(s_p))$.

- $\Rightarrow$ direction. The proof is by induction of the number n of derivation steps.

  Base case: $n = 1$ and the rule is $S_i \to 1_i$. By definition of G and of the $L(s_i, s_j)$'s, there is a successful run on a configuration qw labelling the root by s_i. Then $\#(r) = (0, \ldots, 0, 1, 0, \ldots, 0)$ which is equal to the Parikh mapping of 1_i.

  Induction step: The derivation starts by $S_i \to 1_i X_L$. By definition of the grammar G_L, X_L generates a word $w_S = S_{i_1} \ldots S_{i_k}$ of $\{S_1, \ldots, S_p\}^*$ such that the Parikh mapping of this word (as a word of V^* belongs to L.

  By induction hypothesis, each derivation issued from a letter S_i of w_S generates a word w_i and corresponds to a run r_i on a configuration t_i labelling the root of t_i by s_i. By definition of the rules of G, the Parikh mapping of w_S belongs to $L = L(s_i, s_j)$ for some s_j. Therefore there is a successful run r on a tree $t = qw(t_{i_1} \parallel \ldots \parallel t_{i_k})$ labelling the root of t by s_i, and $\#_r = (0, \ldots, 0, 1, 0, \ldots) + \#r_{i_1} + \ldots + \#r_{i_k}$ by induction hypothesis. The word generated by the derivation is $1_i w_{i_1} \ldots w_{i_k}$ and by induction hypothesis $\#_{r_i}$ is the Parikh mapping of w_i for each $i = i_1, \ldots, i_k$. Hence the result holds.

- $\Leftarrow$ direction. The proof is by structural induction on the tree t that is labelled by r and is a direct consequence of the definitions of rules of G. $\square$

Let L_R be a regular set of flat configurations, we can assume that a Presburger tree automaton accepting L is $\mathcal{A}_R = (Q \cup \Sigma, \bar{S}, \{\bar{s}_0\}, \bar{R} \cup \bar{R}')$ where $\bar{R}' = \{s_0 \to \varphi(x_1, \ldots, x_p)\}$ and $\bar{R}$ consists of rules $\bar{s} \to \#$ or $\bar{s} \xrightarrow{a} \bar{s}'$ with $\bar{s}, \bar{s}' \neq s_0$.

Let L be a set of configurations accepted by a Presburger weighted tree automaton $\mathcal{A} = (Q \cup \Sigma, S, S_I, \mathcal{S}_m, R \cup R')$. We can assume that no ϵ-rule occurs in R nor $\bar{R}$ and we use a product-like construction to combine $\mathcal{A}_R$ and $\mathcal{A}$. Let p (resp. $\bar{p}$) be the number of states of $\mathcal{A}$ (resp. $\mathcal{A}_R$).

Let $\mathcal{B} = (Q \cup \Sigma, S \times \bar{S}, S_I \times \{\bar{s}_0\}, \mathcal{S}_m, \Delta \cup \Delta')$ be the weighted tree automaton such that:

- if $s \xrightarrow{a,C} s' \in R$, $\bar{s} \xrightarrow{a} \bar{s}' \in \bar{R}$, then $s \times \bar{s} \xrightarrow{a,C} s' \times \bar{s}' \in \Delta$,
- if $s \xrightarrow{(0,\ldots,0)} \# \in R$, $\bar{s} \to \# \in \bar{R}$, then $s \times \bar{s} \xrightarrow{(0,\ldots,0)} \# \in \Delta$,
- if $s \xrightarrow{\|} \phi(x_1, \ldots, x_p, y_1, \ldots, y_m) \in R$ and $\bar{s} \to \# \in \bar{R}$ then $s \times \bar{s} \xrightarrow{\|} \phi(\Sigma_{j=1}^{j=\bar{p}} x_1^j, \ldots, \Sigma_{j=1}^{j=\bar{p}} x_p^j, y_1, \ldots, y_m) \in \Delta'$.

The proof of the following proposition is similar to the proof of the closure of weighted tree automata by intersection (see [10]):

Proposition 25. *A run of $\mathcal{B}$ on a tree t labels the root of a subterm $qw(\_)$ of t by $(s, \bar{s})$ iff a run of $\mathcal{A}$ on t labels the root of $qw(\_)$ by s and a run of $\mathcal{A}_R$ on $qw(\#)$ labels the root of $qw(\#)$ by $\bar{s}$.*

By definition of $\mathcal{B}$, a successful run r of $\mathcal{B}$ on a tree t yields a successful run of $\mathcal{A}$ on t (simply forget the second component of product). It will also provide a successful run of $\mathcal{A}_R$ on $flat(t)$ provided an additional constraint is satisfied. Let n_j be the number $\Sigma_{i=1}^{i=p} \#r(s_i \times \bar{s}_j)$ for $j = 1, \ldots, \bar{p}$. Since a subterm $qw(\_)$ of t yields a subterm $qw(\#)$ of $flat(t)$, we get that $flat(t)$ is accepted by A_R iff $\varphi(n_1, \ldots, n_{\bar{p}})$ holds. Since the set $\{\#_r \mid r \; successful \; run \; of \; \mathcal{B}\}$ is a semilinear set and since $\varphi(x_1, \ldots, x_{\bar{p}})$ is a semilinear set, by the decidability of Presburger arithmetic, one can decide the existence of a successful run of $\mathcal{A}_R$ on a configuration $c = flat(t)$ such that $\mathcal{A}$ has a successful run on t. This is summarized in the following theorem:

Theorem 26. *Let P a DNPS and let L and Bad be regular languages of flat configurations. Then it is decidable if $flat(Post^*(L)) \cap Bad$ is empty or not.*

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 14 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. A more ad-hoc approach is likely to yield (non-deterministic) polynomial bounds but will require to define a more restricted class of tree automata. Similarly, extending our result using constrained rules as in [1] requires to restrict the kind of constraints (i.e. semilinear sets) that we allow and is not compatible with the flat model. Finally, adding synchronization in the flavor [6] for a finite set of processes is a extension of the model which should work in the same way as in the case of backward analysis [7].

References

[1] 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 *Lecture Notes in Computer Science*, pages 473–487. Springer, 2005.

[2] J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL*, pages 1–11, 2000.

[3] O. Maler A. Bouajjani, J. Esparza. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings 8th International Conference on Concurrency Theory (CONCUR)*, volume 1243 of *Lecture Notes in Computer Science*, pages 14–25. Springer-Verlag, 1997.

[4] 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.

[5] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on pa-processes. In D.Sangiorgi and R.de Simone, editors, *Concur'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 50–66. Springer-Verlag, September 1998.

[6] 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.

[7] P. Lammich, M. Mller-Olm, and A. Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In *Proc. of 21st Conf. on Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 525–539. Springer, 2009.

[8] 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.

[9] R. Mayr and M. Rusinowitch. Reachability is decidable for ground ac rewrite systems. In *Proceedings of the 3rd INFINITY Workshop*, pages 53–64, 1998.

[10] D. Lugiez. Forward analysis of dynamic network of pushdown systems is easier without order, 2009. Available at http://www.lif.univ-mrs.fr/˜lugiez.

[11] H. Comon, M. Dauchet, F. Jacquemard, C. Loeding, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata and their application.* Freeware Book, 1999. available at http://tata.gforge.inria.fr/.

[12] R. Buchi. Regular canonical systems. *Archiv fur Matematische Logik und Grundlagenforschung*, 6(91-111), 1964.

[13] D. Caucal. On word rewriting systems having a rational derivation. In Jerzy Tiuryn, editor, *FoSSaCS*, volume 1784 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2000.

[14] 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.

[15] C. Reutenauer. *The Mathematics Of Petri Nets*. Prentice-Hall, 1990. ISBN 0135618878.

[16] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[17] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.