# The Regular Viewpoint on PA-Processes


D. Lugiez

*Lab. d'Informatique de Marseille, Univ. Aix-Marseille & CNRS URA 1787, 39, r. Joliot-Curie, 13453 Marseille Cedex 13, FRANCE*

and

Ph. Schnoebelen

*Lab. Specification and Verification, ENS de Cachan & CNRS UMR 8643, 61 av. Pdt Wilson, 94235 Cachan Cedex, FRANCE*



**Abstract**

PA is the process algebra allowing non-determinism, sequential and parallel compositions, and recursion. We suggest viewing PA-processes as *trees*, and using *tree-automata techniques* for verification problems on PA.

Our main result is that the set of iterated predecessors of a regular set of PA-processes is a regular tree language, and similarly for iterated successors. Furthermore, the corresponding tree-automata can be built effectively in polynomial-time. This has many immediate applications to verification problems for PA-processes, among which a simple and general model-checking algorithm.



*Key words:* Process algebra, verification of infinite-state systems, tree-automata.


## Introduction

**Verification of Infinite State Processes** is a very active field of research today in the concurrency-theory community. There has been an active Petri-nets community for many years, but researchers involved in process algebra and model-checking really became interested in infinite state processes after the proof that bisimulation was decidable for normed BPA-processes [BBK87]. This prompted the investigation of decidability issues for BPP and BPA, with


[1] The first author is supported by INRIA Action Presysa.

or without the normedness condition (see [CHM94,Mol96,BE97] for a partial survey).

**From BPA and BPP to PA:** BPA denotes the "non-determinism + sequential composition + recursion" fragment of process algebra. BPP is the "non-determinism + parallel composition + recursion" fragment. PA [BW90] combines both and is much less tractable. A few years ago, while more and more decidability results for BPP and BPA were presented, PA was still beyond the reach of the current techniques. Then R. Mayr showed the decidability of reachability for PA processes [May97c], and extended this into decidability of model-checking for PA w.r.t. the EF fragment of CTL [May97b]. This was an important breakthrough, allowing Mayr to successfully attack more powerful process algebras [May97a] while other decidability results for PA were presented by him and other researchers (e.g., [Kuč96,Kuč97,JKM98,HJ99]).

**A field asking for new insights:** The decidability proofs from [May97b] (and the following papers) are certainly not trivial. The constructions are quite complex and hard to check. It is not easy to see in which directions the results and/or the proofs could be adapted or generalized without too much trouble. Probably, this complexity cannot be avoided with the techniques currently available in the field. We believe we are at a point where it is more important to look for new insights, concepts and techniques that will simplify the field, rather than trying to further extend already existing results.

**Our contribution:** In this article, we show how *tree-automata techniques* greatly help dealing with PA. Our main results are two *Regularity Theorems*, stating that $Post^*(L)$ and $Pre^*(L)$, the sets of configurations reachable from (resp. allowing to reach) a configuration in some set $L$, is a regular tree language when $L$ is, and giving simple polynomial-time constructions for the associated automata. Many important consequences follow directly, including a simple algorithm for model-checking PA-processes.

**Why does it work?** The regularity of $Post^*(L)$ and $Pre^*(L)$ could only be obtained after we had the combination of two main insights:

**1.** the tree-automata techniques that proved to be very powerful in several fields (see [CKSV97]) are useful for process-algebraic problems as well. After all, PA is a simple term-rewrite system with a special context-sensitive rewriting strategy, not unlike head-rewriting, in the presence of the sequential composition operator.

**2.** the syntactic congruences used to simplify notations in simple process algebras bring one closer to the intended semantics of processes, but they break the regularity of the behavior. The decidability results are much simpler when one only introduces syntactic congruences at a later stage. (Besides, this is a more general approach.)

**Plan of the article:** We start by recalling basic notions and facts from tree-automata theory (§ 1) before we introduce our definition for the PA process algebra (§ 2). After we explain how sets of PA processes can be seen as tree languages (§ 3), we give a simple proof showing how $Post^*(t)$ and $Pre^*(t)$ are regular tree languages and start listing applications to verification problems. We then move on to $Post^*(L)$ and $Pre^*(L)$ for $L$ a regular language (§ 5). These are our main technical results and we devote § 6 to the important applications in model-checking. We end up with an extension to reachability and model-checking *under constraints* (§ 7) and some simple but important techniques allowing us to deal with PA processes modulo structural equivalence (§ 8).

**Related work:** Tree-automata have been successfully used in the field of branching-time temporal logics [BVW94]. There, automata recognize the computation trees of finite-state systems, not the reachability sets of infinite-state systems.

The set of all reachable configurations of a pushdown automaton is a regular (word) language. This folk theorem is often attributed to [Büc64]. It was extended in [Cau92] and applications to the model-checking of pushdown automata have been proposed in [FWW97,BEM97]. A "parallel" variant exists: the reachable configurations of a BPP process form a semilinear set [Esp97].

The transitive closure $\xrightarrow{*}$ of the rewrite relation induced by a ground term rewrite system is recognizable by *ground tree transducers* [DT90]. Note that PA is defined by a *conditional* ground rewrite system, and in fact the induced reachability relation is a *rational tree relation* in the sense of Raoult [Rao97] (see § 5.3).

Among the applications we develop for our regularity theorems, several have been suggested by Mayr's work on PA [May97c,May97b] and/or our earlier work on RPPS [KS97a,KS97b]. Our results have been used in [EK99].

## 1 Regular tree languages and tree automata

We recall some basic definitions and results on tree automata and regular tree languages. For more details, the reader is referred to any classical source (e.g. [CDG+99,GS97]).

A *ranked alphabet* is a finite set of symbols $\mathcal{F}$ together with an arity function $\eta : \mathcal{F} \to \mathbb{N}$. This partitions $\mathcal{F}$ according to arities: $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots$. We write $\mathcal{T}(\mathcal{F})$ the set of terms over $\mathcal{F}$ and call them *finite trees* or just *trees*.

A *tree language* over $\mathcal{F}$ is any subset of $\mathcal{T}(\mathcal{F})$.

**Tree automata.** A (finite, bottom-up) *tree automaton* $\mathcal{A}$ is a tuple $\langle \mathcal{F}, Q, F, R \rangle$ where $\mathcal{F}$ is a ranked alphabet, $Q = \{q_1, \dots\}$ is a finite set of *states*, $F \subseteq Q$ is the subset of *final states*, and $R$ is a finite set of *transition rules* of the form $f(q_1, \dots, q_n) \longmapsto q$ where $n \geq 0$ is the arity $\eta(f)$ of symbol $f \in \mathcal{F}$. *Tree automata with $\varepsilon$-rules* also allow some transition rules of the form $q \longmapsto q'$.

The transition rules define a rewrite relation on terms built on $\mathcal{F} \cup Q$ (seeing states from $Q$ as nullary symbols). This works bottom-up. At first the nullary symbols at the leaves are replaced by states from $Q$, and then the quasi-leaf symbols immediately on top of leaves from $Q$ are replaced by states from $Q$. We write $t \xmapsto{\mathcal{A}} q$ when, using rules from $\mathcal{A}$, $t \in \mathcal{T}(\mathcal{F})$ can be rewritten (in some number of steps) to $q \in Q$ and say $t$ is accepted by $\mathcal{A}$ if it can be rewritten into a final state of $\mathcal{A}$. We write $L(\mathcal{A})$ for the set of all terms accepted by $\mathcal{A}$. Any tree language which coincides with $L(\mathcal{A})$ for some $\mathcal{A}$ is a *regular tree language*. Regular tree languages are closed under complementation, union, etc.

A tree automaton is *completely specified* (also *complete*) if for each $f \in \mathcal{F}_n$ and $q_1, \dots, q_n \in Q$, there is a rule $f(q_1, \dots, q_n) \longmapsto q$. By adding a sink state and the obvious rules, any $\mathcal{A}$ can be extended into a complete one accepting the same language.

**An example.** Let $\mathcal{F}$ be given by $\mathcal{F}_0 = \{a, b\}$, $\mathcal{F}_1 = \{g\}$ and $\mathcal{F}_2 = \{f\}$. There is an automaton accepting the set of all $t \in \mathcal{T}(\mathcal{F})$ where $g$ occurs an even number of times in $t$. $\mathcal{A}$ is given by $Q \stackrel{\text{def}}{=} \{q_0, q_1\}$, $R \stackrel{\text{def}}{=} \{a \longmapsto q_0, b \longmapsto q_0, g(q_0) \longmapsto q_1, g(q_1) \longmapsto q_0, f(q_0, q_0) \longmapsto q_0, f(q_0, q_1) \longmapsto q_1, f(q_1, q_0) \longmapsto q_1, f(q_1, q_1) \longmapsto q_0\}$ and $F \stackrel{\text{def}}{=} \{q_0\}$.

Let $t$ be $g(f(g(a), b))$. $\mathcal{A}$ rewrites $t$ as follows:

$$g(f(g(a), b)) \longmapsto g(f(g(q_0), q_0)) \longmapsto g(f(q_1, q_0)) \longmapsto g(q_1) \longmapsto q_0.$$

Hence $t \xrightarrow{\mathcal{A}} q_0$ and $q_0 \in F$, so that $t \in L(\mathcal{A})$.

If we replace $R$ by $R' \stackrel{\text{def}}{=} \{a \longmapsto q_0, b \longmapsto q_0, g(q_0) \longmapsto q_1, g(q_1) \longmapsto q_0, f(q_0, q_0) \longmapsto q_0, f(q_1, q_1) \longmapsto q_1\}$ we have an automaton accepting the set of all $t$ where there is an even number of $g$'s along every path from the root to a leaf.

**Complexity.** The *size* of a tree automaton, denoted by $|\mathcal{A}|$, is the number of states of $\mathcal{A}$ plus the size of the rules of $\mathcal{A}$, where a rule $f(q_1, \ldots, q_n) \longmapsto q$ has size $n + 2$. In this article, we shall never be more precise than counting $|Q|$, the number of states of our automata. Note that, for a fixed $\mathcal{F}$ where the largest arity is $m \geq 2$, $|\mathcal{A}|$ is in $O(|Q|^m)$.

A tree automaton is *deterministic* if all transition rules have distinct left-hand sides (and there are no $\varepsilon$-rules). Otherwise it is non-deterministic. Given a non-deterministic tree automaton, one can use the classical "subset construction" and, at the cost of a potential exponential blow-up in size, build a deterministic tree automaton accepting the same language. Telling whether $L(\mathcal{A})$ is empty for $\mathcal{A}$ a (non-necessarily deterministic) tree automaton can be done in time $O(|\mathcal{A}|)$. Telling whether a given tree $t$ is accepted by a given (non-necessarily deterministic) $\mathcal{A}$ can be done in time polynomial in $|\mathcal{A}| + |t|$.

**Regular equations.** Given a set of variables $L_1, \ldots, L_n$, a *regular equation* has the form $L = exp_1 \cup \ldots \cup exp_k$ where the $exp_i$ have the form $f(L_{n_{i,1}}, \ldots, L_{n_{i,m}})$ (for $f \in \mathcal{F}_m$). Here the $L$'s denote tree-languages, and $f(L_1, \ldots, L_m)$ is $\{f(t_1, \ldots, t_m) \mid t_i \in L_i\}$. The least solution (w.r.t. inclusion) of a finite set of regular equations always exists and is a (tuple of) regular tree-language(s). As with word languages, it is easy to translate a tree-automaton into an equivalent system of regular equations, and vice versa.

## 2 The PA process algebra

For our presentation of PA, we explicitly refrain from writing terms modulo some simplification laws (e.g. the neutral laws for 0). Hence our use of the *IsNil* predicate (see below), inspired by [Chr93].

This viewpoint is in agreement with the earliest works on (general) process algebras like CCS, ACP, etc. It is a key condition for the results of the next section, and it clearly does not prevent considering terms modulo some structural congruence at a later stage, as we demonstrate in section 8.

### 2.1 Syntax

$Act = \{a, b, c, \dots\}$ is a set of *action names*.
$Var = \{X, Y, Z, \dots\}$ is a set of *process variables*.
$E_{\mathrm{PA}} = \{t, u, \dots\}$ is the set of PA-terms, given by the following abstract syntax

$$t, u ::= 0 \mid X \mid t.u \mid t \parallel u$$

Given $t \in E_{\mathrm{PA}}$, we write $Var(t)$ the set of process variables occurring in $t$ and $Subterms(t)$ the set of all subterms of $t$ (includes $t$).

A guarded PA *declaration* is a finite set $\Delta = \{X_i \xrightarrow{a_i} t_i \mid i = 1, \dots, n\}$ of *process rewrite rules*. Note that the $X_i$'s need not be distinct.

We write $Var(\Delta)$ for the set of process variables occurring in $\Delta$, and $Subterms(\Delta)$ the union of all $Subterms(t)$ for $t$ a right- or a left-hand side of a rule in $\Delta$.

$\Delta_a(X)$ denotes $\{t \mid \text{there is a rule "}X \xrightarrow{a} t\text{" in } \Delta\}$ and $\Delta(X)$ is $\bigcup_{a \in Act} \Delta_a(X)$. $Var_\varnothing \stackrel{\text{def}}{=} \{X \in Var \mid \Delta(X) = \varnothing\}$ is the set of variables for which $\Delta$ provides no rewrite.

In the following, we assume a fixed $Var$ and $\Delta$.

### 2.2 Semantics

A PA declaration $\Delta$ defines a labeled transition relation $\rightarrow_\Delta \subseteq E_{\mathrm{PA}} \times Act \times E_{\mathrm{PA}}$. We always omit the $\Delta$ subscript when no confusion is possible, and use the standard notations and abbreviations: $t \xrightarrow{w} t'$ with $w \in Act^*$, $t \xrightarrow{k} t'$ with $k \in \mathbb{N}$, $t \xrightarrow{*} t'$, $t \rightarrow$, ... $\rightarrow_\Delta$ is inductively defined via the following SOS rules:

$$\frac{t_1 \xrightarrow{a} t'_1}{t_1 \parallel t_2 \xrightarrow{a} t'_1 \parallel t_2} \qquad \frac{t_1 \xrightarrow{a} t'_1}{t_1.t_2 \xrightarrow{a} t'_1.t_2} \qquad \frac{}{X \xrightarrow{a} t}(X \xrightarrow{a} t) \in \Delta$$

$$\frac{t_2 \xrightarrow{a} t'_2}{t_1 \parallel t_2 \xrightarrow{a} t_1 \parallel t'_2} \qquad \frac{t_2 \xrightarrow{a} t'_2}{t_1.t_2 \xrightarrow{a} t_1.t'_2} IsNil(t_1)$$

where the $IsNil(\dots)$ predicate is inductively defined by

$$IsNil(t_1 \parallel t_2) \stackrel{\text{def}}{=} IsNil(t_1) \wedge IsNil(t_2), \qquad IsNil(0) \stackrel{\text{def}}{=} true,$$

$$IsNil(t_1.t_2) \stackrel{\text{def}}{=} IsNil(t_1) \wedge IsNil(t_2), \quad IsNil(X) \stackrel{\text{def}}{=} \begin{cases} true \text{ if } \Delta(X) = \varnothing, \\ false \text{ otherwise.} \end{cases}$$

The *IsNil* predicate is a syntactic test for termination, and indeed

**Lemma 2.1.** *The following three properties are equivalent:*
*1.* $IsNil(t) = true$*,*
*2.* $t \not\rightarrow$ *(i.e. t is terminated),*
*3.* $Var(t) \subseteq Var_{\varnothing}$*.*

*Proof.* ($3 \Rightarrow 2$) Assume, by way of contradiction, that $t \to t'$. This derivation used some process rewrite rule $X_i \xrightarrow{a_i} t_i$ with $X_i \in Var(t)$.
($2 \Rightarrow 1$) Use induction over $t$ to prove that $IsNil(t) = false$ implies that $t \to t'$ for some $t'$.
($1 \Rightarrow 3$) is obvious from the definition. □

## 3 $E_{\mathrm{PA}}$ as a tree language

We shall use tree automata to recognize sets of terms from $E_{\mathrm{PA}}$. This is possible because $E_{\mathrm{PA}}$ is just a $\mathcal{T}(\mathcal{F})$ for $\mathcal{F}$ given by $\mathcal{F}_0 = \{0, X, Y, \dots\}$ ($= \{0\} \cup Var$) and $F_2 = \{., \parallel\}$. Of course, we shall keep using the usual infix notation for terms built with "." or "$\parallel$".

We begin with one of the simplest languages in $E_{\mathrm{PA}}$:

**Proposition 3.1.** *For any $t$, the singleton tree language $\{t\}$ is regular, and an automaton for $\{t\}$ needs only have $|t|$ states.*

Similarly, an immediate consequence of Lemma 2.1 is

**Corollary 3.2.** *$L^{\varnothing}$, the set of terminated processes, is a regular tree language, and an automaton for $L^{\varnothing}$ needs only have one state.*

## 4 Regularity of the reachability set

For $t \in E_{\mathrm{PA}}$, we let $Pre^*(t) \stackrel{\text{def}}{=} \{t' \mid t' \xrightarrow{*} t\}$ (resp. $Post^*(t) \stackrel{\text{def}}{=} \{t' \mid t \xrightarrow{*} t'\}$) denote the set of iterated predecessors (resp. the set of iterated successors, also called the *reachability set*) of $t$.

These notions do not take into account the sequences $w \in Act^*$ of action names allowing to move from some $t$ to some $t'$ in $Post^*(t)$. Indeed, we will forget about action names until section 7 which is devoted to $Pre^*[C](t)$ and $Post^*[C](t)$ for $C \subseteq Act^*$.

Given two tree languages $L, L' \subseteq E_{\mathrm{PA}}$, we let

$$L.L' \stackrel{\text{def}}{=} \{t.t' \mid t \in L, t' \in L'\}, \qquad\qquad L \parallel L' \stackrel{\text{def}}{=} \{t \parallel t' \mid t \in L, t' \in L'\}.$$

### *4.1 Regularity of $Post^*(t)$*

We define $(L'_t)_{t \in E_{\mathrm{PA}}}$, $(L''_t)_{t \in E_{\mathrm{PA}}}$, two infinite families of tree languages, as the least solution of the following set of recursive equations:

$$\begin{aligned}
L'_0 &= \{0\}, & L''_0 &= \{0\}, \\
L'_X &= \{X\} \cup \bigcup_{X \xrightarrow{a} t \in \Delta} L'_t, & L''_X &= \begin{cases} \{X\} \text{ if } X \in Var_\varnothing, \\ \bigcup_{X \xrightarrow{a} t \in \Delta} L''_t, \text{ otherwise,} \end{cases} \\
L'_{t \parallel t'} &= L'_t \parallel L'_{t'}, & L''_{t \parallel t'} &= L''_t \parallel L''_{t'}, \\
L'_{t.t'} &= L'_t.\{t'\} \cup L''_t.L'_{t'}, & L''_{t.t'} &= L''_t.L''_{t'}.
\end{aligned} \tag{1}$$

**Lemma 4.1.** *For any $t \in E_{\mathrm{PA}}$, $L'_t = Post^*(t)$ and $L''_t = Post^*(t) \cap L^\varnothing$.*

*Proof (sketch).* We have $L'_t \subseteq Post^*(t)$ and $L''_t \subseteq Post^*(t) \cap L^\varnothing$ because $Post^*(t)$ and $Post^*(t) \cap L^\varnothing$ satisfy the equations in (1).

For the other direction, we show that $t \xrightarrow{k} u$ imply $u \in L'_t$ by induction over $k$ and then by induction over the structure of $t$. E.g., when $t$ is some $t_1.t_2$ we use the fact that $t \xrightarrow{k} u$ entails that $u$ is some $u_1.u_2$ s.t. $t_i \xrightarrow{k_i} u_i$ and $k_i \leq k$ $(i = 1, 2)$, and $t_2 = u_2$ if $u_1 \notin L^\varnothing$. $\square$

The equations in (1) can easily be turned into regular equations with standard unfolding and replacement techniques. The important observation is that, eventually, a given $L'_t$ or $L''_t$ only depends on a finite number of $L'_u$'s, $L''_u$'s and

$\{u\}$'s (the $u$'s are subterms of $t$ and of $\Delta$), so that a *finite set* of regular equations defining $L'_t$ can be extracted.
**Corollary 4.2.** *For any $t \in E_{\text{PA}}$, the sets $L'_t$ and $L''_t$ are regular tree languages.*

and the corresponding tree automata have $O(|\Delta| + |t|)$ states. Hence
**Theorem 4.3.** *For any $t \in E_{\text{PA}}$, $Post^*(t)$, $Post(t)$ and $Post^+(t)$ are regular tree languages that can be constructed effectively.*

### *4.2 Regularity of $Pre^*(t)$*

We define $(L_t)_{t \in E_{\text{PA}}}$, an infinite family of tree languages, as the least solution of the following set of recursive equations.

$$L_0 = \{0\} \cup \bigcup_{Y \xrightarrow{a} 0 \in \Delta} L_Y, \qquad L_{t \| t'} = L_t \parallel L_{t'} \cup \bigcup_{t \| t' \in Post^*(Y)} L_Y,$$

$$L_X = \{X\} \cup \bigcup_{Y \xrightarrow{a} X \in \Delta} L_Y, \qquad L_{t.t'} = \begin{cases} L_t.L_{t'} \cup \bigcup_{t.t' \in Post^*(Y)} L_Y, & \text{if } t \in L^{\varnothing}, \\ \\ L_t.\{t'\} \cup \bigcup_{t.t' \in Post^*(Y)} L_Y, & \text{otherwise.} \end{cases} \tag{2}$$

**Lemma 4.4.** *For any $t \in E_{\text{PA}}$, $L_t = Pre^*(t)$.*

*Proof.* Similar to the proof of Lemma 4.1, and omitted.

□

Again, a given $L_t$ only depends eventually on a finite number of $L_u$'s [2] and the equations from (2) can easily be transformed into regular equations. Observe that the conditions $t \parallel t' \in Post^*(Y)$ (resp. $t.t' \in Post^*(Y)$) are effective since $Post^*(Y)$ is a regular tree language and we can build an automaton accepting this language. Moreover we can test in one pass whether a term $s$ or any of its subterm belongs to $Post^*(Y)$ in time $O(|s| \times |\Delta|)$. Therefore the set of equations defining $L_t$ can be computed in time $O(\max(|t|, |\Delta|) \times |\Delta|^2)$.
**Corollary 4.5.** *For any $t \in E_{\text{PA}}$, the set $L_t$ is a regular tree language.*

and the corresponding tree automaton has $O(|\Delta| + |t|)$ states. This entails

[2] In section 5.1, we shall see that Corollary 4.5 holds even when $\Delta$ is infinite (but $Var(\Delta)$ must be finite).

**Theorem 4.6.** *For any $t \in E_{\mathrm{PA}}$, $Pre^*(t)$, $Pre(t)$ and $Pre^+(t)$ are regular tree languages.*

Theorems 4.3 and 4.6 will be generalized in sections 5 and 7. However, we found it enlightening to give simple proofs of the simplest variants of our regularity results.

### *4.3 Some applications*

Theorems 4.3 and 4.6 and the effective constructibility of the associated automata already have many applications.

**Theorem 4.7.** *The reachability problem "is $t$ reachable from $t'$?" is in P.*

*Proof.* Combine the cost of membership testing for non-deterministic tree automata and the regularity of $Pre^*(t')$ or the regularity of $Post^*(t)$. $\square$

For a different presentation of PA and $\rightarrow_\Delta$, [May97c] shows that the reachability problem is NP-complete. In section 8, we describe how to get his result as a byproduct of our approach.

Many other problems are solved by simple application of Theorems 4.6 and 4.3:

**boundedness.** Is $Post^*(t)$ infinite?
**covering.** (a.k.a. control-state reachability). Can we reach a $t'$ in which $Y_1, \ldots, Y_m$ occur (resp. do not occur).
**inclusion.** Are all states reachable from $t_1$ also reachable from $t_2$? Same question modulo a regularity preserving operation (e.g. projection).
**liveness.** where a given $\Delta' \subseteq \Delta$ is live if, in all reachable states, at least one transition from $\Delta'$ can be fired.

## 5 Regularity of $Post^*(L)$ and $Pre^*(L)$ for a regular language $L$

In this section we prove the regularity of $Pre^*(L)$ and $Post^*(L)$ for a regular language $L$.

For notational simplicity, given two states $q, q'$ of an automaton $\mathcal{A}$, we denote by $q \parallel q'$ (resp. $q.q'$) any state $q''$ such that $q \parallel q' \xmapsto{\mathcal{A}} q''$ (resp. $q.q' \xmapsto{\mathcal{A}} q''$), possibly using $\varepsilon$-rules.

**Ingredients for $\mathcal{A}_{Pre^*}$:** Assume $\mathcal{A}_L$ is an automaton recognizing $L \subseteq E_{\mathrm{PA}}$. $\mathcal{A}_{Pre^*}$ is a new automaton combining several ingredients:

- $\mathcal{A}_{\varnothing}$ is a *completely specified* automaton accepting terminated processes (see Corollary 3.2).
- $\mathcal{A}_L$ is the automaton accepting $L$.
- We also use a boolean to record whether some rewriting steps have been done.

**States of $\mathcal{A}_{Pre^*}$:** A state of $\mathcal{A}_{Pre^*}$ is a 3-tuple $(q_{\varnothing} \in Q_{\mathcal{A}_{\varnothing}}, q_L \in Q_{\mathcal{A}_L}, b \in \{true, false\})$ where $Q_{\dots}$ denotes the set of states of the relevant automaton.

The underlying idea is that a state $(q_{\varnothing}, q_L, b)$ recognize any predecessor of a term recognized by $q_L$. The other two components record additional information (is a subterm terminated, is it a strict predecessor or not) we need to make Lemma 5.1 work.

**Transition rules of $\mathcal{A}_{Pre^*}$:** The transition rules of $\mathcal{A}_{Pre^*}$ are easy to understand once one knows we have Lemma 5.1 in mind. Formally, they are defined as follows:

**type** $0$**:** all rules of the form $0 \longmapsto (q_{\varnothing}, q_L, false)$ s.t. $0 \overset{\mathcal{A}_{\varnothing}}{\longmapsto} q_{\varnothing}$ and $0 \overset{\mathcal{A}_L}{\longmapsto} q_L$.
**type** $1a$**:** all rules of the form $X \longmapsto (q_{\varnothing}, q_L, true)$ s.t. there exists some $u \in Post^+(X)$ with $u \overset{\mathcal{A}_{\varnothing}}{\longmapsto} q_{\varnothing}$ and $u \overset{\mathcal{A}_L}{\longmapsto} q_L$.
**type** $1b$**:** all rules of the form $X \longmapsto (q_{\varnothing}, q_L, false)$ s.t. $X \overset{\mathcal{A}_{\varnothing}}{\longmapsto} q_{\varnothing}$ and $X \overset{\mathcal{A}_L}{\longmapsto} q_L$.
**type** $2$**:** all rules of the form $(q_{\varnothing}, q_L, b) \parallel (q'_{\varnothing}, q'_L, b') \longmapsto (q_{\varnothing} \parallel q'_{\varnothing}, q_L \parallel q'_L, b \vee b')$.
**type** $3a$**:** all rules of the form $(q_{\varnothing}, q_L, b).(q'_{\varnothing}, q'_L, b') \longmapsto (q_{\varnothing}.q'_{\varnothing}, q_L.q'_L, b \vee b')$ s.t. $q_{\varnothing}$ is a final state of $\mathcal{A}_{\varnothing}$.
**type** $3b$**:** all rules of the form $(q_{\varnothing}, q_L, b).(q'_{\varnothing}, q'_L, false) \longmapsto (q_{\varnothing}.q'_{\varnothing}, q_L.q'_L, b)$.

**Lemma 5.1.** *For any $t \in E_{\mathrm{PA}}$, $t \overset{\mathcal{A}_{Pre^*}}{\longmapsto} (q_{\varnothing}, q_L, b)$ iff there is some $u \in E_{\mathrm{PA}}$ and some $p \in \mathbb{N}$ such that $t \xrightarrow{p} u$, $u \overset{\mathcal{A}_{\varnothing}}{\longmapsto} q_{\varnothing}$, $u \overset{\mathcal{A}_L}{\longmapsto} q_L$ and ($b = false$ iff $p = 0$).*

*Proof.* By structural induction over $t$. There are three cases:

**1.** $t = 0$ **or** $t = X$**:** Because $\mathcal{A}_{Pre^*}$ has no $\varepsilon$-rules, we only have to observe that its rules of type $0$, $1a$ and $1b$ exactly correspond to what the lemma requires.

**2.** $t = t_1.t_2$**:** $(\Rightarrow)$: the rewrite $t \xmapsto{\mathcal{A}_{Pre^*}} (q_\varnothing, q_L, b)$ required that, for $i = 1, 2$, we have $t_i \xmapsto{\mathcal{A}_{Pre^*}} (q^i_\varnothing, q^i_L, b^i)$ and there is a type 3 rule $(q^1_\varnothing, q^1_L, b^1).(q^2_\varnothing, q^2_L, b^2) \longmapsto (q_\varnothing, q_L, b)$.

The induction hypothesis entails there are $t_1 \xrightarrow{p_1} u_1$ and $t_2 \xrightarrow{p_2} u_2$ corresponding to the rewrite of $t_1$ and $t_2$ by $\mathcal{A}_{Pre^*}$. Now if $\mathcal{A}_{Pre^*}$ used a type $3b$ rule, then $b_2 = \mathit{false}$ hence $p_2 = 0$, $u_2 = t_2$, $p_1 = p$ and $t_1.t_2 \xrightarrow{p_1} u_1.t_2 = u_1.u_2$. If we used a type $3a$ rule, then $q^1_\varnothing$ is a final state, therefore $u_1 \in L_\varnothing$ is a terminated process, hence $t_1.t_2 \xrightarrow{p_1} u_1.t_2 \xrightarrow{p_2} u_1.u_2$ and ($b = b_1 \vee b_2 = \mathit{false}$ iff $p_1 + p_2 = 0$).

$(\Leftarrow)$: Conversely, assume $t = t_1.t_2 \xrightarrow{p} u$ with $u \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$ and $u \xmapsto{\mathcal{A}_L} q_L$. Then $u$ is some $u_1.u_2$ and either (1) $u_2 = t_2$ and $t_1 \xrightarrow{p} u_1$, or (2) $u_1 \in L_\varnothing$ and $t_1.t_2 \xrightarrow{p_1} u_1.t_2 \xrightarrow{p_2} u_1.u_2$ for $p_1 + p_2 = p$.

In the first case the ind. hyp. entails $t_1 \xmapsto{\mathcal{A}_{Pre^*}} (q^1_\varnothing, q^1_L, b^1)$ with $u_1 \xmapsto{\mathcal{A}_L} q^1_L$, and $t_2 = u_2 \xmapsto{\mathcal{A}_{Pre^*}} (q^2_\varnothing, q^2_L, \mathit{false})$. Now we can use a type $3b$ rule to show $t \xmapsto{\mathcal{A}_{Pre^*}} (q^1_\varnothing.q^2_\varnothing, q^1_L.q^2_L, b_1)$ with $u \xmapsto{\mathcal{A}_L} q^1_L.q^2_L$.

In the second case, $u_1 \in L_\varnothing$ entails $t_1 \xmapsto{\mathcal{A}_L} (q^1_\varnothing, q^1_L, b_1)$ with $q^1_\varnothing$ a final state of $\mathcal{A}_\varnothing$. We can use a type $3a$ rule to show $t \xmapsto{\mathcal{A}_{Pre^*}} (q^1_\varnothing.q^2_\varnothing, q^1_L.q^2_L, b_1 \vee b_2)$.

**3.** $t = t_1 \parallel t_2$**:** This case is similar to the previous one (actually it is simpler).

□

If we now let the final states of $\mathcal{A}_{Pre^*}$ be all states $(q_\varnothing, q_L, b)$ s.t. $q_L$ is a final state of $\mathcal{A}_L$, then $t \xrightarrow{*} u$ for some $u$ accepted by $\mathcal{A}_L$ iff $\mathcal{A}_{Pre^*}$ accepts $t$ (this is where we use the assumption that $\mathcal{A}_\varnothing$ is completely specified.)

**Theorem 5.2. (Regularity)**
*(1) If $L$ is a regular subset of $E_{\mathrm{PA}}$, then $Pre^*(L)$ is regular.*
*(2) Furthermore, from an automaton $\mathcal{A}_L$ recognizing $L$, it is possible to construct (in polynomial time) an automaton $\mathcal{A}_{Pre^*}$ recognizing $Pre^*(L)$. If $\mathcal{A}_L$ has $k$ states, then $\mathcal{A}_{Pre^*}$ needs only have $4k$ states.*

*Proof.* (1) is an immediate consequence of Lemma 5.1. Observe that the result does not need the finiteness of $\Delta$ (but $Var(\Delta)$ must be finite).

(2) Building $\mathcal{A}_{Pre^*}$ effectively requires an effective way of listing the type $1a$ rules. This can be done by computing a product of $\mathcal{A}_X$, an automaton for $Post^+(X)$, with $\mathcal{A}_\varnothing$ and $\mathcal{A}_L$. Then there exists some $u \in Post^+(X)$ with $u \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$ and $u \xmapsto{\mathcal{A}_L} q_L$ iff the language accepted by the final states $\{(q_X, q_\varnothing, q_L) \mid q_X$ a final state of $\mathcal{A}_X\}$ is not-empty. This gives us the pairs $q_\varnothing, q_L$ we need for type $1a$ rules. Observe that we need the finiteness of $\Delta$ to build the $\mathcal{A}_X$'s. □

**Ingredients for $\mathcal{A}_{Post^*}$:** Assume $\mathcal{A}_L$ is an automaton recognizing $L \subseteq E_{\mathrm{PA}}$. $\mathcal{A}_{Post^*}$ is a new automaton combining several ingredients:

- Automata $\mathcal{A}_\varnothing$ and $\mathcal{A}_L$ as in the previous construction, but this time *we need to assume each of them is a completely specified automata.*
- $\mathcal{A}_\Delta$ is a completely specified automaton recognizing the subterms of $\Delta$. More precisely, it has all states $q_s$ for $s \in \mathit{Subterms}(\Delta)$. We ensure "$t \overset{\mathcal{A}_\Delta}{\longmapsto} q_s$ iff $s = t$" by taking as transition rules $0 \longmapsto q_0$ if $0 \in \mathit{Subterms}(\Delta)$, $X \longmapsto q_X$ if $X \in \mathit{Subterms}(\Delta)$, $q_s \| q_{s'} \longmapsto q_{s\|s'}$ (resp. $q_s.q_{s'} \longmapsto q_{s.s'}$) if $s\|s'$ (resp. $s.s'$) belongs to $\mathit{Subterms}(\Delta)$. In addition, the automaton has a sink state $q_\perp$ and the obvious transitions so that it is a completely specified automaton.
- Again, we use a boolean $b$ to record whether rewrite steps have occurred.

**States of $\mathcal{A}_{Post^*}$:** The states of $\mathcal{A}_{Post^*}$ are 4-uples $(q_\varnothing \in Q_{\mathcal{A}_\varnothing}, q_L \in Q_{\mathcal{A}_L}, q_\Delta \in Q_{\mathcal{A}_\Delta}, b \in \{\mathit{true}, \mathit{false}\})$.

Here the underlying idea is that a state $(q_\varnothing, q_L, q_\Delta, b)$ recognize any successor of a term recognized by $q_L$, as formally stated in Lemma 5.3. If $q_\Delta$ is some $q_u$ for a subterm $u$ of $\Delta$, this means that additionally the current term happens to be $u$.

**Transition rules of $\mathcal{A}_{Post^*}$:** The transition rules of $\mathcal{A}_{Post^*}$ are easy to understand once one knows we have Lemma 5.3 in mind. Formally, they are defined as follows:

**type** $0$**:** all rules of the form $0 \longmapsto (q_\varnothing, q_L, q_\Delta, \mathit{false})$ s.t. $0 \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$, $0 \overset{\mathcal{A}_L}{\longmapsto} q_L$ and $0 \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$.

**type** $1$**:** all rules of the form $X \longmapsto (q_\varnothing, q_L, q_\Delta, \mathit{false})$ s.t. $X \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$, $X \overset{\mathcal{A}_L}{\longmapsto} q_L$, and $X \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$.

**type** $2$**:** all $\varepsilon$-rules of the form $(q_\varnothing, q'_L, q_s, b') \longmapsto (q_\varnothing, q_L, q_X, \mathit{true})$ s.t. $X \to s$ is a rule in $\Delta$ with $X \overset{\mathcal{A}_L}{\longmapsto} q_L$.

**type** $3$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b) \| (q'_\varnothing, q'_L, q'_\Delta, b') \longmapsto (q_\varnothing \| q'_\varnothing, q_L \| q'_L, q_\Delta \| q'_\Delta, b \vee b')$

**type** $4a$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b).(q'_\varnothing, q'_L, q'_\Delta, \mathit{false}) \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, q_\Delta.q'_\Delta, b)$.

**type** $4b$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b).(q'_\varnothing, q'_L, q'_\Delta, b') \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, q_\Delta.q'_\Delta, b \vee b')$ s.t. $q_\varnothing$ is a final state of $\mathcal{A}_\varnothing$.

**Lemma 5.3.** *For any $t \in E_{\mathrm{PA}}$, $t \overset{\mathcal{A}_{Post^*}}{\longmapsto} (q_\varnothing, q_L, q_\Delta, b)$ iff there is some $u \in E_{\mathrm{PA}}$ and some $p \in \mathbb{N}$ such that $u \xrightarrow{p} t$, $u \overset{\mathcal{A}_L}{\longmapsto} q_L$, $u \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$, ($b = \mathit{false}$ iff $p = 0$) and $t \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$.*

*Proof.* We first prove the ($\Rightarrow$) direction by induction over the length $k$ of the rewrite $t \overset{\mathcal{A}_{Post^*}}{\longmapsto} (q_\varnothing, q_L, q_\Delta, b)$. We distinguish four cases:

**1.** $k = 1$**:** Then $t = 0$ or $t = X$ and we used a type 0 or type 1 rule. Taking $u = t$ and $p = 0$ satifies the requirements.

**2.** $k > 1$ **and the last rewrite step used a type** 2 $\varepsilon$**-rule:** Then the rewrite has the form $\overbrace{t \longmapsto (q'_\varnothing, q'_L, q_s, b')}^{k-1 \text{ steps}} \longmapsto (q_\varnothing, q_L, q_X, \mathit{true})$. By ind. hyp., there is a $u'$ and a $p'$ s.t. $u' \xrightarrow{p'} t$. Now $u' \overset{\mathcal{A}_\Delta}{\longmapsto} q_s$ entails $u' = s$. The existence of the type 2 rule entails $X \to s \in \Delta$. Hence $X \xrightarrow{p'+1} t$. Taking $u = X$ and $p = p' + 1$ satifies the requirements.

**3.** $k > 1$ **and the last rewrite step used a type** 4 **rule:** Then $t$ is some $t_1.t_2$ and the type 4 rule applied on top of two rewrite sequences $t_i \longmapsto (q^i_\varnothing, q^i_L, q^i_\Delta, b^i)$ for $i = 1, 2$. The ind. hyp. gives us, for $i = 1, 2$, some $u_i$ and $p_i$ s.t. $u_i \xrightarrow{p_i} t_i$.

If the last rule was a type $4a$ rule, then $b^2 = \mathit{false}$ so that $p_2 = 0$ and $u_2 = t_2$. Then $u_1.u_2 \xrightarrow{p_1} t_1.u_2 = t$. Taking $u = u_1.u_2$ and $p = p_1$ satifies the requirements.

Otherwise the last rule was a type $4b$ rule. Then $q^1_\varnothing$ is a final state and $t_1 \overset{\mathcal{A}_\varnothing}{\longmapsto} q^1_\varnothing$ entails that $t_1$ is a terminated process. Hence $u_1.u_2 \xrightarrow{p_1} t_1.u_2 \xrightarrow{p_2} t_1.t_2 = t$. Again, taking $u = u_1.u_2$ (with $p = p_1 + p_2$) satifies the requirements.

**4.** $k > 1$ **and the last rewrite step used a type** 3 **rule:** This case is similar (actually simpler) to the previous one.

For the ($\Leftarrow$) direction, we assume $u \xrightarrow{p} t$ with the accompanying conditions (a.c.), and proceed by induction over the length of the transition sequence (i.e. over $p$), followed by structural induction over $u$. There are five cases:

**1.** $u = 0$**:** Then $t = u$ and the a.c.'s ensure there is a type 0 rule for $t \overset{\mathcal{A}_{Post^*}}{\longmapsto} (q_\varnothing, q_L, q_\Delta, \mathit{false})$.

**2.** $u = X$ **and** $p = 0$**:** Like the previous case but with a type 1 rule.

**3.** $u = X$ **and** $p > 0$**:** Then the sequence has the form $X \xrightarrow{1} u' \xrightarrow{p-1} t$. Here the a.c.'s read $q_\Delta = q_X$ and $b = \mathit{true}$. $X \xrightarrow{1} u' \in \Delta$ entails $u' \in \mathit{Subterms}(\Delta)$. If we now take a $q'_L$ s.t. $u' \overset{\mathcal{A}_L}{\longmapsto} q'_L$ (one such $q'_L$ must exist) and let $b'$ be $\mathit{false}$ iff $p - 1 = 0$, the ind. hyp. gives us $t \overset{\mathcal{A}_{Post^*}}{\longmapsto} (q_\varnothing, q'_L, q_{u'}, b')$. Now, there must be a type 2 $\varepsilon$-rule $(q_\varnothing, q'_L, q_{u'}, b') \longmapsto (q_\varnothing, q_L, q_X, \mathit{true})$. We use it to show

$t \xrightarrow{\mathcal{A}_{Post^*}} (q_\varnothing, q_L, q_X, true)$.

**4.** $u = u_1.u_2$**:** Then $t$ is some $t_1.t_2$ and $u \xrightarrow{p} t$ is a combination of some $u_1 \xrightarrow{p_1} t_1$ and $u_2 \xrightarrow{p_2} t_2$ with $p = p_1 + p_2$. Additionally, if $p_2 > 0$ then $t_1 \in L_\varnothing$.

For $i = 1, 2$, The rewrites $t \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$, $u \xmapsto{\mathcal{A}_L} q_L$ and $u \xmapsto{\mathcal{A}_\Delta} q_\Delta$ used some $t_i \xmapsto{\mathcal{A}_\varnothing} q^i_\varnothing$, $u_i \xmapsto{\mathcal{A}_L} q^i_L$ and $u_i \xmapsto{\mathcal{A}_\Delta} q^i_\Delta$ (for $i = 1, 2$). If we now define $b^i$ according to $p_i$, the ind. hyp. entails that, for $i = 1, 2$, $t_i \xmapsto{\mathcal{A}_{Post^*}} (q^i_\varnothing, q^i_L, q^i_\Delta, b^i)$.

There are two cases. If $t_1 \in L_\varnothing$ then $q^1_\varnothing$ is a final state of $\mathcal{A}_\varnothing$ and $\mathcal{A}_{Post^*}$ has a type $4b$ rule $(q^1_\varnothing, q^1_L, q^1_\Delta, b^1).(q^2_\varnothing, q^2_L, q^2_\Delta, b^2) \longmapsto (q_\varnothing, q_L, q_\Delta, b)$ that we can use. If $t_1 \notin L_\varnothing$, then $p_2 = 0$ and $b^2 = false$. There is a type $4a$ rule that we can use.

**5.** $u = u_1 \parallel u_2$**:** Similar to the previous case (actually it is simpler).

□

If we now let the final states of $\mathcal{A}_{Post^*}$ be all states $(q_\varnothing, q_L, q_\Delta, b)$ s.t. $q_L$ is a final state of $\mathcal{A}_L$, then $\mathcal{A}_{Post^*}$ accepts a term $t$ iff $u \xrightarrow{*} t$ for a $u$ accepted by $\mathcal{A}_L$ iff $t$ belongs to $Post^*(L)$. Hence the

**Theorem 5.4. (Regularity)**
*(1) If $L$ is a regular subset of $E_{\text{PA}}$, then $Post^*(L)$ is regular.*
*(2) Furthermore, from an automaton $\mathcal{A}_L$ recognizing $L$, it is possible to construct (in polynomial time) an automaton $\mathcal{A}_{Post^*}$ recognizing $Post^*(L)$. If $\mathcal{A}_L$ has $k$ states, then $\mathcal{A}_{Pre^*}$ needs only have $O(k.|\Delta|)$ states.*

*Proof.* Obvious from the previous construction. □

### *5.3 Recognizability of $\xrightarrow{*}$*

Our two regularity theorems relate $L$ and $Pre^*(L)$ (resp. $Post^*(L)$). A natural question is to ask whether the relation "$\xrightarrow{*}$" (i.e. $\{(t, u) \mid t \xrightarrow{*} u\}$, a subset of $E_{\text{PA}} \times E_{\text{PA}}$) is recognizable in some sense.

**Ground tree transducers.** For this question, the most relevant notion of recognizability is based on *ground tree transducers*, GTT's for short, see [DT90,CDG⁺99] for details. It can be shown that the $\xrightarrow{*}$ relation induced by a ground rewrite system is recognizable by a GTT. In the case of PA processes, the rules are ground rewrite rules with simple left-hand sides, but with a contextual restriction on when a rule may be applied (reflecting the semantics of the sequential composition operator). This restriction has the unfortunate consequence that $\xrightarrow{*}$ for PA is not stable under contexts. And the natural extensions of GTT that could handle such conditional rules are immediately able to recognize any recursively enumerable relation.

**Rational tree relations.** Actually, there exists a notion of relation on trees which allows us to say $\xrightarrow{*}$ is recognizable (with, as a corollary, an alternative proof of our two regularity theorems). These relations are Raoult's *rational tree relations* [Rao97]. Compared to tree automata, the rational tree relations allow a more general result (i.e. recognizability of $\xrightarrow{*}$) but they require more involved notions and techniques. Here we only give the main ideas on how this alternative technique works.

Top-down tree automata can be easily understood as bottom-up tree automata working in the reverse direction. Final states are now initial states and a rule $f(q_1, \ldots, q_n) \to q$ yields a rule $q \to f(q_1, \ldots, q_n)$. The expressive power of top-down and bottom-up tree automata is the same. The regular equations we used are a formalism very close to the tree automata formalism. In fact, these regular equations are just *regular tree grammars.*

Rational tree relations are similar to tree grammars. The main difference is that rules are defined for tuples of non-terminals, i.e. a rule has the form $[X_1, \ldots, X_n] \to [t_1, \ldots, t_n]$ where the $t_i$'s are terms built from function symbols in $\mathcal{F}$ and non-terminals of the grammar (seen as nullary symbols). Several copies of the same tuple of non-terminals may appear in $[t_1, \ldots, t_n]$ and we add indexes to distinguish between copies. For instance the right-hand side of $[X, Y] \to [f(X^1, X^2), f(Y^1, Y^2)]$ involves two copies of the pair $[X, Y]$ i.e. $[X^1, Y^1]$ and $[X^2, Y^2]$. If we assume a second rule $[X, Y] \to [a, b]$, a possible derivation is

$$[X, Y] \to [f(X^1, X^2), f(Y^1, Y^2)] \to [f(a, X^2), f(b, Y^2)] \to [f(a, a), f(b, b)].$$

If furthermore the axiom of the grammar is the pair $[X, Y]$, then we say that $[f(a, a), f(b, b)]$ belong to the relation generated by the grammar. Here the relation is the set of pairs $[s, t]$ which can be derived from the axiom. In general, when the axiom is a $n$-tuple, the defined relation is a $n$-ary relation. Such relations are called *rational tree relations.* For a complete description of rational tree grammars, we refer to [Rao97].

**Proposition 5.5.** *The relation $\xrightarrow{*} \subseteq E_{\mathrm{PA}} \times E_{\mathrm{PA}}$ is a rational tree relation and a grammar generating $\xrightarrow{*}$ has size $O(|\Delta|)$.*

*Proof.* Using results from section 4, we easily get a grammar (with axiom $N_X$) generating $Post^*(X)$, and this for any $X$. In the same way we get grammars (with axiom $N'_X$) generating $Post^*(X) \cap L^{\varnothing}$.

$s \xrightarrow{*} t$ holds if $s = C[X_1, \ldots, X_n]$ and $t = C[t_1, \ldots, t_n]$ for some context $C$, and $X_i \xrightarrow{*} t_i$ for $i = 1, \ldots, n$. Moreover a rewrite may occur to the right of a sequential composition only if no rewrite may occur to its left. The idea of the next construction is to give a grammar for pairs such that each component of the pair derive the context $C$ until the left part derives some $X_i$ and the right part some $t_i \in Post^*(X_i)$. Moreover, the condition on the sequential composition complicates the grammar a little bit. For simplicity, we assume that there is only one variable and we use the classical presentation of grammar rules.

The pair $[I, R]$ ($I$ for *identity*, $R$ for *rewrite*) generates pairs $[s, t]$ such that $s \xrightarrow{*} t$.

$$
\begin{aligned}
[I, R] \to\ & [.(I, I_l), .(R, I_r)] \\
\mid\ & [.(I', I), .(RT, R)] \\
\mid\ & [\| (I^1, I^2), \| (R^1, R^2))] \\
\mid\ & [X, N_X]
\end{aligned}
$$

The pair $[I_l, I_r]$ ($I$ for *identity* and subscripts $r, l$ for *right* and *left*) generates pairs $[s, s]$ for any ground term $s$.

$$
\begin{aligned}
[I_l, I_r] \to\ & [.(I_l^1, I_l^2), .(I_r^1, I_r^2)] \\
\mid\ & [\| (I_l^1, I_l^2), \| (I_r^1, I_r^2)] \\
\mid\ & [X, X]
\end{aligned}
$$

The pair $[I', RT]$ generates pairs $[s, t]$ such that $s \xrightarrow{*} t$ and $t$ is a terminated process.

$$
\begin{aligned}
[I', RT] \to\ & [X, N'_X] \\
\mid\ & [\| ({I'}^1, {I'}^2), \| (RT^1, RT^2)] \\
\mid\ & [.({I'}^1, {I'}^2), .(RT^1, RT^2)]
\end{aligned}
$$

□

Rational tree relations are closed under composition when the grammars are transduction grammmars. This means roughly that we can decompose each non-terminal into two parts, the first one for the first projection, the second one for the second projection. The grammar for $\xrightarrow{*}$ has this property. Rational tree relations generated by transduction grammars are closed under composition and inverse. Moreover the image and the range of a rational tree relation is a regular tree language.

It is then possible to derive our two regularity theorems from Proposition 5.5. For $L$ a regular tree language such that there exists an automaton with $k$ states accepting $L$, there exists a transduction grammar of size $O(k)$ generating the relation $Id_L = \{(t, t) \mid t \in L\}$. By composing $Id_L$ and $\xrightarrow{*}$ we get that

$Post^*(L)$ and $Pre^*(L)$ are regular tree languages. The composition between $Id_L$ and $\xrightarrow{*}$ requires a synchronization between the rules of $Id_L$ and $\xrightarrow{*}$. This yields an $O(k|\Delta|)$ bound on the resulting grammar, hence on the size of the tree automaton accepting $Post^*(L)$ (resp. $Pre^*(L)$).

If we compare this to the direct automata-theoretic proof we gave, this technique yields the regularity of both $Post^*(L)$ and $Pre^*(L)$ at the same time. But the direct approach is simpler and can be adapted e.g. for reachability under constraints (section 7) where actions are taken into account. Dealing with constraints inside the rational tree relation framework would require adding the constraints inside Raoult's framework, and adapting the complex proofs of [Rao97].

## 6 Model-checking PA processes

In this section we show a simple approach to the model-checking problem solved in [May97b]. We see this as one more immediate application of our main regularity theorems.

We consider a set $Prop = \{P_1, P_2, \dots\}$ of *atomic propositions*. For $P \in Prop$, Let $Mod(P)$ denotes the set of PA processes for which $P$ holds. We only consider propositions $P$ such that $Mod(P)$ is a regular tree-language. Thus $P$ could be "$t$ can make an $a$-labeled step right now", "there is at least two occurences of $X$ inside $t$", "there is exactly one occurence of $X$ in a non-frozen position", ...

The logic EF has the following syntax:

$$\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{EX}\varphi \mid \mathsf{EF}\varphi$$

and semantics

$$t \models P \overset{\text{def}}{\Leftrightarrow} t \in Mod(P), \qquad t \models \neg\varphi \overset{\text{def}}{\Leftrightarrow} t \not\models \varphi, \qquad t \models \varphi \wedge \varphi' \overset{\text{def}}{\Leftrightarrow} t \models \varphi \text{ and } t \models \varphi',$$

$$t \models \mathsf{EX}\varphi \overset{\text{def}}{\Leftrightarrow} t' \models \varphi \text{ for some } t \to t', \qquad t \models \mathsf{EF}\varphi \overset{\text{def}}{\Leftrightarrow} t' \models \varphi \text{ for some } t \xrightarrow{*} t'.$$

Thus $\mathsf{EX}\varphi$ reads "it is possible to reach in one step a state s.t. $\varphi$" and $\mathsf{EF}\varphi$ reads "it is possible to reach (via some sequence of steps) a state s.t. $\varphi$".
**Definition 6.1.** *The* model-checking problem *for EF over PA has as inputs a given $t$ in $E_{\text{PA}}$ and a given $\varphi$ in EF. The answer is yes iff $t \models \varphi$.*

R. Mayr [May97b] gives a quite involved procedure for this problem. Here we give a simple more general solution.

Let us define $Mod(\varphi) \stackrel{\text{def}}{=} \{t \in E_{\text{PA}} \mid t \models \varphi\}$. It is well known that the following holds

$$\begin{aligned} Mod(\neg\varphi) &= E_{\text{PA}} - Mod(\varphi) & Mod(\mathsf{EX}\varphi) &= Pre(Mod(\varphi)) \\ Mod(\varphi \wedge \varphi') &= Mod(\varphi) \cap Mod(\varphi') & Mod(\mathsf{EF}\varphi) &= Pre^*(Mod(\varphi)) \end{aligned} \tag{3}$$

**Theorem 6.2.** *(1) For any EF formula $\varphi$, $Mod(\varphi)$ is a regular tree language. (2) If we are given tree-automata $\mathcal{A}_P$'s recognizing the regular sets $Mod(P)$, then a tree-automaton $\mathcal{A}_\varphi$ recognizing $Mod(\varphi)$ can be built effectively.*

*Proof.* A corollary of (3) and the regularity theorems. □

This gives us a decision procedure for the model-checking problem: build an automaton for $Mod(\varphi)$ and check whether it accepts $t$. We can estimate the complexity of this approach in term of $|\varphi|$ and $n_{\text{alt}}(\varphi)$.

We define $n_{\text{alt}}(\varphi)$ the number of alternation of negations and temporal connectives in $\varphi$ as

$$\begin{aligned} n_{\text{alt}}(P) &= 0 & n_{\text{alt}}(\neg P) &= 1 \\ n_{\text{alt}}(\varphi \wedge \psi) &= \max(n_{\text{alt}}(\varphi), n_{\text{alt}}(\psi)) & n_{\text{alt}}(\neg(\varphi \wedge \psi)) &= \max(n_{\text{alt}}(\neg\varphi), n_{\text{alt}}(\neg\psi)) \\ n_{\text{alt}}(\mathsf{EF}\varphi) &= n_{\text{alt}}(\varphi) & n_{\text{alt}}(\neg\mathsf{EF}\varphi) &= 1 + n_{\text{alt}}(\varphi) \\ n_{\text{alt}}(\mathsf{EX}\varphi) &= n_{\text{alt}}(\varphi) & n_{\text{alt}}(\neg\mathsf{EX}\varphi) &= 1 + n_{\text{alt}}(\varphi) \\ & & n_{\text{alt}}(\neg\neg\varphi) &= n_{\text{alt}}(\varphi) \end{aligned}$$

**Theorem 6.3. (Model-checking)** *An automaton for $Mod(\varphi)$ can be computed in time*

$$\left.2^{|\varphi||\Delta|2^{\cdot^{\cdot^{\cdot^{2^{|\varphi||\Delta|2^{O(|\varphi||\Delta|)}}}}}}}\right\}n_{\text{alt}}(\varphi) .$$

*Proof.* We assume all automata for the $Mod(P)$'s have size bounded by $M$ (a constant). We construct an automaton for $Mod(\varphi)$ by applying the usual automata-theoretic constructions for intersection, union, complementation of regular tree languages, and by invoking our regularity theorems for *Pre* and

$Pre^*$. All constructions are polynomial except for complementation. With only polynomial constructions, we would have a $2^{O(|\varphi|)}$ size for the resulting automaton. The negations involving complementation are the cause of the non-elementary blowup.

Negations can be pushed inward except that they cannot cross the temporal connectives $\mathsf{EF}$ and $\mathsf{EX}$. Here we have one exponential blowup for determinization at each level of alternation. This is repeated $n_{\text{alt}}(\varphi)$ times, yielding the given bound on the number of states hence the overall complexity. $\square$

The procedure described in [May97b] is non-elementary (and the known lower-bound is PSPACE-hard).

Observe that computing a representation of $Mod(\varphi)$ is more general than just telling whether a given $t$ belongs to it. Observe also that our results just translate the EF logic into a combination of boolean and $Pre^*$ operations on sets, hence assuming a "backward" method. More generally, our two regularity theorems allow symbolic model-checking approaches based on combinations of forward and backward methods.

## 7 Reachability under constraints

In this section, we consider *reachability under constraints*. Let $C \subseteq Act^*$ be a (word) language over action names. We write $t \xrightarrow{C} t'$ when $t \xrightarrow{w} t'$ for some $w \in C$, and we say that $t'$ can be reached from $t$ under the constraint $C$. We extend our notations and write $Pre^*[C](L)$, $Post^*[C](L)$, ... with the obvious meaning.

Observe that, even if we assume $C$ is regular, the problem of telling whether $t \xrightarrow{C}$, i.e. whether $Post^*[C](t)$ is not empty, is undecidable for the PA algebra. This can be proved by a reduction from the intersection problem for context-free languages as follows: Let $\Sigma$ be an alphabet and $\#$ some distinguished symbol. We use two copies $\underline{a}, \overline{a}$ of every letter $a$ in $\Sigma \cup \{\#\}$. Context-free languages can be defined in BPA (PA without $\|$), that is, for any context-free language $L_1$ (resp. $L_2$) on $\Sigma$, we can define PA rules such that $X_1 \xrightarrow{\underline{w}.\underline{\#}}$ iff $w \in L_1$ (resp.$X_2 \xrightarrow{\overline{w}.\overline{\#}}$ iff $w \in L_2$). These rules don't overlap. We now introduce the regular constraint $C \stackrel{\text{def}}{=} (\underline{a_1}.\overline{a_1} + \cdots + \underline{a_n}.\overline{a_n})^* \underline{\#}.\overline{\#}$. Then $(X_1 \| X_2) \xrightarrow{C}$ holds iff $L_1 \cap L_2 \neq \emptyset$, which is undecidable.

In this section we give sufficient conditions over $C$ so that the problem becomes decidable (and so that we can compute the $C$-constrained $Pre^*$ of a regular tree language).

### 7.1 Decomposable languages

Recall that the *shuffle* $w \sqcup\!\sqcup\, w'$ of two finite words is the set of all words one can obtain by interleaving $w$ and $w'$ in an arbitary way.

**Definition 7.1.** • $\{(C_1, C'_1), \dots, (C_m, C'_m)\}$ *is a (finite)* seq-decomposition *of $C$ iff for all $w, w' \in Act^*$ we have*

$$w.w' \in C \quad \textit{iff} \quad (w \in C_i, w' \in C'_i \textit{ for some } 1 \leq i \leq m).$$

• $\{(C_1, C'_2), \dots, (C_m, C'_m)\}$ *is a (finite)* paral-decomposition *of $C$ iff for all $w, w' \in Act^*$ we have*

$$C \cap (w \sqcup\!\sqcup\, w') \neq \varnothing \quad \textit{iff} \quad (w \in C_i, w' \in C'_i \textit{ for some } 1 \leq i \leq m).$$

Observe that a seq-decomposition of $C$ must apply to all possible ways of splitting any word in $C$. It even applies to a decomposition $w.w'$ with $w = \varepsilon$ (or $w' = \varepsilon$) so that one of the $C_i$'s (and one of the $C''_i$'s) contains $\varepsilon$.

Actually, there are close links between seq-decompositions and the classical notion of residuals of a language, but paral-decompositions have no equivalent.

Seq- and paral-decompositions look similar, but $w \| w'$ usually contains several elements: when $w \in C$ can be decomposed as a shuffle of some $u$ and some $v$, there must be a $(C_i, C'_i)$ for $(u, v)$. Reciprocally, when $u \in C_i$ and $v \in C'_i$, there must be some way of shuffling them into some $w \in C$. Hence, while we have $C_i.C'_i \subseteq C$ in seq-decompositions, we don't ask for $(C_i \parallel C'_i) \subseteq C$ in paral-decompositions, and in general it does not hold.

**Definition 7.2.** *A family $\mathbb{C} = \{C_1, \dots, C_n\}$ of languages over $Act$ is a* finite decomposition system *iff every $C \in \mathbb{C}$ admits a seq-decomposition and a paral-decomposition only using $C_i$'s from $\mathbb{C}$.*
*A $C$ is decomposable if it belongs to a finite decomposition system.*

**Lemma 7.3.** *Any decomposable $C \subseteq Act^*$ is regular.*

since it has a finite number of residuals.

However not all regular $C$ admit finite decompositions, even in the regular case. Consider $C = (ab)^*$ and assume $\{(C_1, C'_1), \dots\}$ is a finite paral-decomposition. Then for every $k$, there is a shuffle of $a^k$ and $b^k$ in $C$. Hence there must be a $i_k$ s.t. $a^k \in C_{i_k}$ and $b^k \in C'_{i_k}$. Now if $i_k = i_{k'}$ then there there must exist a shuffle $w''$ of $a^k$ and $b^{k'}$ with $w'' \in C$. This is only possible if $k = k'$. Hence all $i_k$'s are distinct, contradicting finiteness.

[Sch99] summarizes what is currently known about decomposable languages. They form a class of regular languages closed by union and concatenation. They include all commutative regular language, hence all UPC's (*union-product of commutative languages*) and it is conjectured that they are exactly the UPC's. They are closed by shuffle, contain all finite and co-finite languages.

### *7.2 Extended regularity theorems*

**Theorem 7.4. (Regularity)**
*For any regular $L \subseteq E_{\mathrm{PA}}$ and any decomposable $C_0 \in \mathbb{C}$, $Pre^*[C_0](L)$ and $Post^*[C_0](L)$ are regular tree languages.*

**Ingredients for $\mathcal{A}_{Post^*[\mathbb{C}]}$:** We build $\mathcal{A}_{Post^*[\mathbb{C}]}$ in the same way as $\mathcal{A}_{Post^*}$ but states contain a new $C \in \mathbb{C}$ component.

**States of $\mathcal{A}_{Post^*[\mathbb{C}]}$:** The states of $\mathcal{A}_{Post^*[\mathbb{C}]}$ are 5-uples $(q_\varnothing \in Q_{\mathcal{A}_\varnothing}, q_L \in Q_{\mathcal{A}_L}, q_\Delta \in Q_{\mathcal{A}_\Delta}, b \in \{\mathit{true}, \mathit{false}\}, C \in \mathbb{C})$.

**Transition rules of $\mathcal{A}_{Post^*[\mathbb{C}]}$:** The transition rules are:

**type** $0$**:** all rules of the form $0 \longmapsto (q_\varnothing, q_L, q_\Delta, \mathit{false}, C)$ s.t. $0 \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$, $0 \overset{\mathcal{A}_L}{\longmapsto} q_L$, $0 \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$ and $\varepsilon \in C$.
**type** $1$**:** all rules of the form $X \longmapsto (q_\varnothing, q_L, q_\Delta, \mathit{false}, C)$ s.t. $X \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$, $X \overset{\mathcal{A}_L}{\longmapsto} q_L$, $X \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$ and $\varepsilon \in C$.
**type** $2$**:** all $\varepsilon$-rules of the form $(q_\varnothing, q'_L, q_s, b', C'') \longmapsto (q_\varnothing, q_L, q_X, \mathit{true}, C)$ s.t. $X \overset{a}{\to} s$ is a rule in $\Delta$ with $X \overset{\mathcal{A}_L}{\longmapsto} q_L$, and $a \in C'$ for some $C'$ s.t. $(C', C'')$ appears in the seq-decomposition of $C$.
**type** $3$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b, C) \parallel (q'_\varnothing, q'_L, q'_\Delta, b', C') \longmapsto (q_\varnothing \parallel q'_\varnothing, q_L \parallel q'_L, q_\Delta \parallel q'_\Delta, b \vee b', C'')$ s.t. $(C, C')$ appears in the paral-decomposition of $C''$.
**type** $4a$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b, C).(q'_\varnothing, q'_L, q'_\Delta, \mathit{false}, C') \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, q_\Delta.q'_\Delta, b, C)$.
**type** $4b$**:** all rules of the form
$(q_\varnothing, q_L, q_\Delta, b, C).(q'_\varnothing, q'_L, q'_\Delta, b', C') \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, q_\Delta.q'_\Delta, b \vee b', C'')$ s.t. $q_\varnothing$ is a final state of $\mathcal{A}_\varnothing$ s.t. $(C, C')$ appears in the seq-decomposition of $C''$.

**Lemma 7.5.** *For any $t \in E_{\mathrm{PA}}$, $t \overset{\mathcal{A}_{Post^*[\mathbb{C}]}}{\longmapsto} (q_\varnothing, q_L, q_\Delta, b, C)$ iff there is some $u \in E_{\mathrm{PA}}$ and some $w \in C$ such that $u \overset{w}{\to} t$, $u \overset{\mathcal{A}_L}{\longmapsto} q_L$, $u \overset{\mathcal{A}_\Delta}{\longmapsto} q_\Delta$, ($b = \mathit{false}$ iff $|w| = 0$) and $t \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$.*

*Proof.* $\mathcal{A}_{Post^*[\mathbb{C}]}$ is $\mathcal{A}_{Post^*}$ equipped with a new component and the proof follows exactly the lines of the proof of Lemma 5.3. We refer to this earlier proof and only explain how we deal with the new $C$ components.

The ($\Rightarrow$) direction is as in lemma 5.3. The new observations in the 4 cases are:

**1.** $k = 1$**:** The type 0 and type 1 rules entail $\varepsilon \in C$, so that we can take $w = \varepsilon$.
**2.** $k > 1$ **and the last rewrite step used a type** $2$ $\varepsilon$**-rule:** Use the fact that $w' \in C''$ entail $a.w' \in C$.
**3.** $k > 1$ **and the last rewrite step used a type** $4$ **rule:** Use the fact that $C.C' \subseteq C''$.
**4.** $k > 1$ **and the last rewrite step used a type** $3$ **rule:** Use the fact that $w_1 \in C$ and $w_2 \in C'$ entail that there exists at least one shuffling $w$ of $w_1$ and $w_2$ s.t. $w \in C''$.

The ($\Leftarrow$) direction is as in lemma 5.3. The new observations in the 5 cases are:

**1.** $u = 0$**:** The type 0 rules allow all $C$'s containing $\varepsilon$.
**2.** $u = X$ **and** $p = 0$**:** *Idem.*
**3.** $u = X$ **and** $p > 0$**:** Then the sequence has the form $X \xrightarrow{a} u' \xrightarrow{w'} t$. Now if $w = a.w' \in C$, there must be a $(C', C'')$ in the seq-decomposition of $C$ s.t. $a \in C'$ and $w' \in C''$. So that there is a type 2 rule $(\dots, q_{u'}, b', C'') \longmapsto (\dots, q_X, \mathit{true}, C)$ we can use.
**4.** $u = u_1.u_2$**:** Here $u_1 \xrightarrow{w_1} t_1$, $u_2 \xrightarrow{w_2} t_2$ and $w_1.w_2 w \in C$. If $t_1 \notin L_\varnothing$ then $w_2 = \varepsilon$, $w_1 \in C$ and we have the type 4a rule we need. Otherwise there is a pair $(C_1, C_2)$ in the seq-decomposition of $C$ s.t. $w_i \in C_i$ $(i = 1, 2)$. This pair gives us the type 4b rule we need.
**5.** $u = u_1 \parallel u_2$**:** Here $u_1 \xrightarrow{w_1} t_1$, $u_2 \xrightarrow{w_2} t_2$ and $w \in C$ is some shuffle of $w_1$ and $w_2$. Therefore there is a $(C_1, C_2)$ in the paral-decomposition of $C$ s.t. $w_i \in C_i$ $(i = 1, 2)$. This pair gives us the type 3 rule we need.

□

If we now let the final states of $\mathcal{A}_{Post^*[\mathbb{C}]}$ be all $(q_\varnothing, q_L, q_\Delta, b, C_0)$ s.t. $q_L$ is a final state of $\mathcal{A}_L$, then $\mathcal{A}_{Post^*[\mathbb{C}]}$ accepts a term $t$ iff $t \in Post^*[C_0](t)$. (The set of final states can easily be adapted so that we recognize $Post^+[C_0](L)$.)

**Ingredients for** $\mathcal{A}_{Pre^*[\mathbb{C}]}$**:** Same as in the construction of $\mathcal{A}_{Pre^*}$, with an additional $C \in \mathbb{C}$ component.

**States of** $\mathcal{A}_{Pre^*}$**:** A state of $\mathcal{A}_{Pre^*}$ is a 4-tuple $(q_\varnothing \in Q_{\mathcal{A}_\varnothing}, q_L \in Q_{\mathcal{A}_L}, b \in \{\mathit{true}, \mathit{false}\}, C \in \mathbb{C})$.

The final states are all $(q_\varnothing, q_L, b, C_i)$ s.t. $q_L$ is a final state of $\mathcal{A}_L$ and $C_i$ the constraint to satisfy.

**Transition rules of $\mathcal{A}_{Pre^*}$:** The transition rules of $\mathcal{A}_{Pre^*}$ are defined as follows:

**type** $0$**:** all rules of the form $0 \longmapsto (q_\varnothing, q_L, \mathit{false}, C)$ s.t. $0 \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$, $0 \xmapsto{\mathcal{A}_L} q_L$ and $\varepsilon \in C$.
**type** $1a$**:** all rules of the form $X \longmapsto (q_\varnothing, q_L, \mathit{true}, C)$ s.t. there exists some $u \in \mathit{Post}^+[C](X)$ with $u \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$ and $u \xmapsto{\mathcal{A}_L} q_L$.
**type** $1b$**:** all rules of the form $X \longmapsto (q_\varnothing, q_L, \mathit{false}, C)$ s.t. $X \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$, $X \xmapsto{\mathcal{A}_L} q_L$ and $\varepsilon \in C$.
**type** $2$**:** all rules of the form $(q_\varnothing, q_L, b, C) \| (q'_\varnothing, q'_L, b', C') \longmapsto (q_\varnothing \| q'_\varnothing, q_L \| q'_L, b \vee b', C'')$ s.t. $(C, C')$ appears in the paral-decomposition of $C''$.
**type** $3a$**:** all rules of the form $(q_\varnothing, q_L, b, C).(q'_\varnothing, q'_L, b', C') \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, b \vee b', C'')$ s.t. $q_\varnothing$ is a final state of $\mathcal{A}_\varnothing$ and $(C, C')$ appears in the seq-decomposition of $C''$
**type** $3b$**:** all rules of the form $(q_\varnothing, q_L, b, C).(q'_\varnothing, q'_L, \mathit{false}, C') \longmapsto (q_\varnothing.q'_\varnothing, q_L.q'_L, b, C)$.

**Lemma 7.6.** *For any $t \in E_{\mathrm{PA}}$, $t \xmapsto{\mathcal{A}_{Pre^*[\mathbb{C}]}} (q_\varnothing, q_L, b, C)$ iff there is some $u \in E_{\mathrm{PA}}$ and some $w \in C$ such that $t \xrightarrow{w} u$, $u \xmapsto{\mathcal{A}_\varnothing} q_\varnothing$, $u \xmapsto{\mathcal{A}_L} q_L$ and ($b = \mathit{false}$ iff $|w| = 0$).*

*Proof.* $\mathcal{A}_{Pre^*[\mathbb{C}]}$ is $\mathcal{A}_{Pre^*}$ equipped with a new component and the proof follows exactly the lines of the proof of Lemma 5.3. We refer to this earlier proof and only explain how we deal with the new $C$ components.

1. **$t = 0$ or $t = X$:** The conditions on the $C$ component for the existence of rules of type 0, $1a$ and $1b$ agree with the statement of the lemma.
2. **$t = t_1.t_2$:** ($\Rightarrow$): Now, for $i = 1, 2$, we have $t_i \xmapsto{\mathcal{A}_{Pre^*}} (q^i_\varnothing, q^i_L, b^i, C^i)$ and there is a type 3 rule $(\dots, C^1).(\dots, C^2) \longmapsto (\dots, C)$. Also, the ind. hyp. gives $t_i \xrightarrow{w_i} u_i$ $(i = 1, 2)$ with $w_1 \in C^i$. In the type $3b$ case, $w_1 \in C$. In the type $3a$ case, we use $C^1.C^2 \subseteq C$.

   ($\Leftarrow$): Here we have either (1) $u_2 = t_2$ and $t_1 \xrightarrow{w} u_1$, or (2) $u_1 \in L_\varnothing$ and $t_1.t_2 \xrightarrow{w_1} u_1.t_2 \xrightarrow{w_2} u_1.u_2$ with $w = w_1.w_2$.

   In the first case we apply the induction hypothesis with $C$ itself on $t_1$ and some $C'$ containing $\varepsilon$ on $t_2$, then we can use a type $3b$ rule. In the second case, there must be a pair $(C^1, C^2)$ in the seq-decomposition of $C$, with $w_i \in C^i$ and we just have to use the ind. hyp. and a type $3a$ rule.

**3.** $t = t_1 \parallel t_2$**:** This case is similar to the previous one. The $(\Leftarrow)$ direction uses the pair accouting for $w_1, w_2$ in the paral-decomposition of $C$. The $(\Rightarrow)$ direction uses the crucial fact that whenever $t_i \xrightarrow{w_i} u_i$ for $i = 1, 2$, we have $t_1 \parallel t_2 \xrightarrow{w} u_1 \parallel u_2$ *for any* $w$ *in* $w_1 \sqcup w_2$, in particular for the $w$ that $C$ must contain.

$\square$

### *7.3 Applications to model-checking*

The above results let us apply the model-checking method from section 6 to an extended EF logic where we now allow all $\langle C\rangle\varphi$ formulas for decomposable $C$. The semantics is given by

$$t \models \langle C\rangle\varphi \overset{\text{def}}{\Leftrightarrow} t' \models \varphi \text{ for some } t \xrightarrow{C} t'.$$

so that $Mod(\langle C\rangle\varphi) = Pre^*[C](Mod(\varphi))$. Then a corollary of Theorem 7.4 is that $Mod(\varphi)$ is regular for any $\varphi$ in the extended EF.

Decomposability of $C$ is a quite general condition. It excludes the undecidable situations that would exist in the general regular case and immediately includes the extensions proposed in [May97b].

Observe that it is possible to combine decomposable constraints already in the model-checking algorithm: when $C \in \mathbb{C}$ and $C' \in \mathbb{C}'$ are decomposable, we can deal with $\langle C \cap C'\rangle\varphi$ directly (i.e. without constructing a finite decomposition system containing $C$ and $C'$) because it is obvious how to extend the construction for $\mathcal{A}_{Pre^*[\mathbb{C}]}$ to some $\mathcal{A}_{Pre^*[\mathbb{C},\mathbb{C}']}$ where several $C$ components are dealt with simultaneously.

One can also deal with $\langle C \cup C'\rangle\varphi$ and $\langle C.C'\rangle\varphi$ directly since $Pre^*[C \cup C'](L)$ and $Pre^*[C.C'](L)$ are $Pre^*[C](L) \cup Pre^*[C'](L)$ and $Pre^*[C](Pre^*[C'](L))$ for any $C, C'$ and $L$.

## 8 Structural equivalence of PA terms

In this section we show how our tree-automata techniques can quite directly solve problems about *PA modulo structural equivalence*. Here we consider the congruence $\equiv$ induced on PA terms by the following equations:

$$
\begin{array}{lc}
(C_{\|}) & t \parallel t' \equiv t' \parallel t \\
(A_{\|}) & (t \parallel t') \parallel t'' \equiv t \parallel (t' \parallel t'') \\
(A_{.}) & (t.t').t'' \equiv t.(t'.t'')
\end{array}
\qquad
\begin{array}{llll}
(N_1) & t.0 \equiv t & (N_3) & t \parallel 0 \equiv t \\
(N_2) & 0.t \equiv t & (N_4) & 0 \parallel t \equiv t
\end{array}
$$

This choice of equations is motivated by the fact that several recent works on PA (and extensions) only consider processes up-to this same congruence. Our tree-automata techniques could deal with variants.

It is useful to explain how our definition of PA compares with the definition used in [May97c,May97b]. We consider a transition system between terms from $E_{\mathrm{PA}}$. The terms Mayr considers for his transition system can be seen as equivalence classes, modulo $\equiv$, of our $E_{\mathrm{PA}}$ terms. Write $[t]_{\equiv}$ for the set $\{t' \mid t \equiv t'\}$. The transition relation used by Mayr coincides with a transition relation defined by

$$[t]_{\equiv} \xrightarrow{a} [u]_{\equiv} \quad \overset{\text{def}}{\Leftrightarrow} \quad \exists t' \in [t]_{\equiv}, u' \in [u]_{\equiv} \text{ s.t. } t' \xrightarrow{a} u'. \tag{4}$$

In the following, we speak of "$\mathrm{PA}_{\equiv}$" when we mean the transition system one obtains with $\equiv$-classes of terms as states, and transitions given by (4).

Our approach to PA is more general in the sense that we can define $\mathrm{PA}_{\equiv}$ in our framework. By contrast, if one reasons modulo $\equiv$ right from the start, one loses the information required to revert to the other approach.

For example, the reachability problem "do we have $t \xrightarrow{*} u$?" from Theorem 4.7 asks for a very precise form for $u$. The reachability problem solved in [May97c] asks for $u$ modulo $\equiv$. In our framework, this can be stated as "given $t$ and $u$, do we have $t' \xrightarrow{*} u'$ for some $t' \equiv t$ and $u' \equiv u$?" (see below). In the other framework, it is impossible to state our problem. (But of course, the first motivation for our framework is that it allows the two regularity theorems.)

The rest of this section is devoted to some applications of our tree-automata approach to problems for $\mathrm{PA}_{\equiv}$. The emphasis is on simplicity, not on exhaustivity.

### *8.1 Structural equivalence and regularity*

$(A_{.})$, $(C_{\|})$ and $(A_{\|})$ are the associativity-commutativity axioms satisfied by $.$ and $\|$. We call them the *permutative axioms* and write $t =_P u$ when $t$ and $t'$ are permutatively equivalent.

$(N_1)$ to $(N_4)$ are the axioms defining 0 as the neutral element of . and $\|$. We call them the *simplification axioms* and write $t \searrow u$ when $u$ is a simplification of $t$, i.e. $u$ can be obtained by applying the simplification axioms *from left to right* at some positions in $t$. Note that $\searrow$ is a (well-founded) partial ordering. We write $\swarrow$ for $(\searrow)^{-1}$. The *simplification normal form* of $t$, written $t{\downarrow}$, is the unique $u$ one obtains by simplifying $t$ as much as possible (no permutation allowed).

Such axioms are classical in rewriting and have been extensively studied [BN98]. $\equiv$ coincide with $(=_P \cup \searrow \cup \swarrow)^*$. Now, because the permutative axioms commute with the simplification axioms (i.e., $\searrow \circ =_P \subseteq =_P \circ \searrow$), we have

$$t \equiv t' \quad \text{iff} \quad t \searrow u =_P u' \swarrow t' \text{ for some } u, u' \quad \text{iff} \quad t{\downarrow} =_P t'{\downarrow}. \tag{5}$$

This lets us decompose questions about $\equiv$ into questions about $=_P$ and questions about $\searrow$. We start with $=_P$.

**Lemma 8.1.** *For any $t$, the set $[t]_{=_P} \stackrel{def}{=} \{u \mid t =_P u\}$ is a regular tree language, and an automaton for $[t]_{=_P}$ needs only have $m.(m/2)!$ states if $|t| = m$.*

*Proof (sketch).* This is because $[t]_{=_P}$ is a finite set with at most $(m/2)!$ elements. (The exponential blowup cannot be avoided.) $\square$

The simplification axioms do not have the nice property that they only allow finitely many combinations, but they behave better w.r.t. regularity. Write $[L]_\searrow$ for $\{u \mid t \searrow u \text{ for some } t \in L\}$, $[L]_\swarrow$ for $\{u \mid u \searrow t \text{ for some } t \in L\}$, and $[L]{\downarrow}$ for $\{t{\downarrow} \mid t \in L\}$.

**Lemma 8.2.** *For any regular $L$, the sets $[L]_\swarrow$, $[L]_\searrow$, and $[L]{\downarrow}$ are regular tree languages. From an automaton $\mathcal{A}_L$ recognizing $L$, we can build automata of size $O(|\mathcal{A}|)$ for these three languages in polynomial time.*

*Proof.* **1.** $[L]_\swarrow$**:** $u$ is in $[L]_\swarrow$ iff $u$ is some $t \in L$ with additional 0's that can be simplified out. Hence an automaton accepting $[L]_\swarrow$ is obtained from $\mathcal{A}_L$ by adding a new state $q_0$ for the subterms that will be simplified. We also add rules $0 \longmapsto q_0$, $q_0 \| q_0 \longmapsto q_0$, and $q_0.q_0 \longmapsto q_0$ for accepting these subterms, and, for any $q$ in $\mathcal{A}_L$, rules $q.q_0 \longmapsto q$, $q_0.q \longmapsto q$, $q \| q_0 \longmapsto q$ and $q_0 \| q \longmapsto q$ for simulating simplification.

**2.** $[L]_\searrow$**:** $u$ is in $[L]_\searrow$ iff $u$ is some $t \in L$ where some 0's have been simplified. A simple way to obtain an automaton for $[L]_\searrow$ is to synchronize the automaton $\mathcal{A}_L$ accepting $L$ with the complete automaton $\mathcal{A}_0$ recognizing terms built with 0, . and $\|$ only. $\mathcal{A}_0$ has only two states: $q^0$ and $q^{\neq 0}$.

Once the two automata are synchronized, we have $t \longmapsto (q, q')$ iff $t \overset{\mathcal{A}_L}{\longmapsto} q$ and $t \overset{\mathcal{A}_0}{\longmapsto} q'$. We simulate simplification of nullable terms with additional $\varepsilon$-rules. Namely, whenever there is a rule $(q_1, q_1') \parallel (q_2.q_2') \longmapsto (q_3.q_3')$ with $q_2' = q^0$, we add an $\varepsilon$-rule $(q_1, q_1') \longmapsto (q_3.q_3')$. We add a symmetric rule if $q_1' = q^0$ and do the same for . instead of $\parallel$.

Now a routine induction on the length of derivations shows that $s \longmapsto (q, q')$ iff $\exists t \in L$ s.t. $t \searrow s$ and $t \overset{\mathcal{A}_L}{\longmapsto} q$.

**3.** $[L]\downarrow$**:** The simplest way to see regularity is to note that $[L]\downarrow$ is $[L]_{\searrow} \cap [E_{\mathrm{PA}}]\downarrow$. □

Note that for a regular $L$, $[L]_{=_P}$ and $[L]_{\equiv}$ are not necessarily regular [GD89]. However we have

**Proposition 8.3.** *For any $t$, the set $[t]_{\equiv}$ is a regular tree language, and an automaton for $[t]_{\equiv}$ needs only have $m.(m/2)!$ states if $|t| = m$.*

*Proof.* Combine (5) with lemmas 8.1 and 8.2. □

### *8.2 Structural equivalence and behaviour*

Seeing terms modulo $\equiv$ does not modify the observable behaviour because of the following

**Proposition 8.4.** *$\equiv$ is a bisimulation relation, i.e. for all $t \equiv t'$ and $t \xrightarrow{a} u$ there is a $t' \xrightarrow{a} u'$ with $u \equiv u'$ (and vice versa).*

The proof is standard but tedious. We shall only give a proof sketch.

*Proof.* For any single equation $l = r$ in the definition of $\equiv$, we show that the set $R = \{(l\sigma, r\sigma)\}$ of all instances of the equation is a bisimulation relation. A complete proof of this for $(A_{\parallel})$ takes the better part of p. 95 of the book [Mil89] and the other six equations can be dealt with similarly, noting that *IsNil*() is compatible with $\equiv$. Then there only remains to prove that the generated congruence is a bisimulation. This too is standard: the SOS rules for PA obey a format ensuring that the behaviour of a term depends on the behaviour of its subterms, not their syntax. □

We may now define a new transition relation between terms: $t \overset{a}{\Rightarrow} t'$ iff $t \equiv u \xrightarrow{a} u' \equiv t'$ for some $u, u'$. This amounts to the "$[t]_{\equiv} \xrightarrow{a} [u]_{\equiv}$" from (4) and is the simplest way to translate problems for $\mathrm{PA}_{\equiv}$ into problems for our set of terms.

We adopt the usual abbreviations $\stackrel{*}{\Rightarrow}$, $\stackrel{w}{\Rightarrow}$, $\stackrel{k}{\Rightarrow}$ for $w \in Act^*$, $k \in \mathbb{N}$, etc.
**Proposition 8.5.** *For any $w \in Act^*$, $t \stackrel{w}{\Rightarrow} u$ iff $t \xrightarrow{w} u'$ for some $u' \equiv u$.*

*Proof.* By induction on the length of $w$, and using Proposition 8.4. □

### *8.3 Reachability modulo ≡*

Now it is easy to prove decidability of the reachability problem modulo $\equiv$: $t \stackrel{*}{\Rightarrow} u$ iff $Post^*(t) \cap [u]_\equiv \neq \varnothing$. Recall that $[u]_\equiv$ and $Post^*(t)$ are regular tree-languages one can build effectively. Hence it is decidable whether they have a non-empty intersection.

This gives us a simple algorithm using exponential time (because of the size of $[u]_\equiv$). Actually we can have a better result [3]:
**Theorem 8.6.** *The reachability problem in* $\mathrm{PA}_\equiv$, *"given $t$ and $u$, do we have $t \stackrel{*}{\Rightarrow} u$?", is in NP.*

*Proof.* NP-easiness is straightforward in the automata framework: We have $t \stackrel{*}{\Rightarrow} u$ iff $t \xrightarrow{*} u'$ for some $u'$ s.t. $u'{\downarrow} =_P u{\downarrow}$. Write $u''$ for $u'{\downarrow}$ and note that $|u''| \leq |u|$. A simple algorithm is to compute $u{\downarrow}$, then *guess non-deterministically* a permutation $u''$, then build automata $\mathcal{A}_1$ for $[u'']_\searrow$ and $\mathcal{A}_2$ for $Post^*(t)$. These automata have polynomial-size. There remains to checks whether $\mathcal{A}_1$ and $\mathcal{A}_2$ have a non-empty intersection to know whether the required $u'$ exists. □

**Corollary 8.7.** *The reachability problem in* $\mathrm{PA}_\equiv$ *is NP-complete.*

*Proof.* NP-hardness of reachability for BPP's is proved in [Esp97] and the proof idea can be reused in our framework. We reduce 3SAT to reachability in $\mathrm{PA}_\equiv$. Consider an instance $P$ of 3SAT. $P$ has $m$ variables and $n$ clauses, so that it is some $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \varepsilon_{i,j} x_{r_{i,j}}$ where, for every $i, j$, $1 \leq r_{i,j} \leq m$ and $\varepsilon_{i,j} \in \{+, -\}$. We define the following $\Delta_P$:

$$\Delta_P \stackrel{\text{def}}{=} \begin{cases} (\mathrm{R1}) \quad X_r \to X_r^\varepsilon & \text{for } 1 \leq r \leq m \text{ and } \varepsilon \in \{+,-\}, \\ (\mathrm{R2}) \quad X_r^\varepsilon \to 0 & \text{for } 1 \leq r \leq m \text{ and } \varepsilon \in \{+,-\}, \\ (\mathrm{R3}) \quad X_{r_{i,j}}^{\varepsilon_{i,j}} \to C_j \parallel X_{r_{i,j}}^{\varepsilon_{i,j}} & \text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq 3. \end{cases}$$

(Note that $|\Delta_P| = O(|P|)$.) The (R1) rules pick a valuation $v$ for the $X_r$'s, the (R3) rules use $v$ to list satisfied clauses, the (R2) rules discard unnecessary

[3] First proved in [May97c]

elements. Finally

$$(X_1 \parallel (X_2 \parallel (\cdots \parallel X_m)\dots)) \overset{*}{\Rightarrow} (C_1 \parallel (C_2 \parallel (\cdots \parallel C_n)\dots)) \text{ iff } P \text{ is satisfiable.}$$

□

Other applications are possible, e.g.:
**Proposition 8.8.** *The boundedness problem in* $\mathrm{PA}_\equiv$ *is decidable in polynomial-time.*

*Proof.* $[t]_\equiv$ can only reach a finite number of states in $\mathrm{PA}_\equiv$ iff $t$ can only reach a finite number of non-$\equiv$ terms in PA. Now because the permutative axioms only allow finitely many variants of any given term, $Post^*(L)$ contains a finite number of non-$\equiv$ processes iff $[Post^*(L)]\!\downarrow$ is finite. □

### *8.4 Model-checking modulo* $\equiv$

The model-checking problem solved in [May97b] considers the EF logic over $\mathrm{PA}_\equiv$. Translated into our framework, this amounts to interpret the temporal connectives in terms of $\Rightarrow$ instead of $\rightarrow$: if we write $Mod_\equiv(\varphi)$ for the interpretation modulo $\equiv$, we have

$$Mod_\equiv(\langle C\rangle\varphi) \overset{\text{def}}{=} \{t \mid t \overset{w}{\Rightarrow} u \text{ for some } u \in Mod_\equiv(\varphi) \text{ and some } w \in C\}.$$

Additionally, we only consider atomic propositions $P$ compatible with $\equiv$, i.e. where $t \models P$ and $t \equiv u$ imply $u \models P$.

Model-checking in $\mathrm{PA}_\equiv$ is as simple as model-checking in PA:
**Lemma 8.9.** *For any EF-formula* $\varphi$ *we have* $Mod_\equiv(\varphi) = Mod(\varphi) = [Mod(\varphi)]_\equiv$.

*Proof.* By structural induction over $\varphi$, using Prop. 8.5 and closure w.r.t. $\equiv$ for the $\langle C\rangle\varphi$ case. □

The immediate corollary is that we can use exactly the same approach for model-checking in PA with or without $\equiv$.

## Conclusion

In this article we showed how tree-automata techniques are a powerful tool for the analysis of the PA process algebra. Our main results are two general

Regularity Theorems with numerous immediate applications, including model-checking of PA with an extended EF logic.

The tree-automata viewpoint has many advantages. It gives simpler and more general proofs. It helps one to understand why some problems can be solved in P-time, some others in NP-time, etc. It is quite versatile and many variants of PA can be attacked with the same approach.

## References


[BBK87] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proc. Parallel Architectures and Languages Europe (PARLE'87), Eindhoven, NL, June 1987, vol. II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 1987.

[BE97] O. Burkart and J. Esparza. More infinite results. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96), Pisa, Italy, Aug. 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science Publishers, 1997.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[Büc64] J. R. Büchi. Regular canonical systems. *Arch. Math. Logik Grundlag.*, 6:91–111, 1964.

[BVW94] O. Bernholtz, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94), Stanford, CA, USA, June 1994*, volume 818 of *Lecture Notes in Computer Science*, pages 142–152. Springer, 1994.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge Univ. Press, 1990.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, 1992.

[CDG+99] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 1997–99. A preliminary version of this electronic book is available at `http://www.grappa.univ-lille3.fr/tata`.

[CHM94] S. Christensen, Y. Hirshfeld, and F. Moller. Decidable subsets of CCS. *The Computer Journal*, 37(4):233–242, 1994.

[Chr93] S. Christensen. Decidability and decomposition in process algebras. PhD thesis CST-105-93, Dept. of Computer Science, University of Edinburgh, UK, 1993.

[CKSV97] H. Comon, D. Kozen, H. Seidl, and M. Y. Vardi, editors. *Applications of Tree Automata in Rewriting, Logic and Programming*, Dagstuhl-Seminar-Report number 193. Schloß Dagstuhl, Germany, 1997.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symp. Logic in Computer Science (LICS'90), Philadelphia, PA, USA, June 1990*, pages 242–248, 1990.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. Conf. Foundations of Software Science and Computation Structures (FoSSaCS'99), Amsterdam, The Netherlands, Mar. 1999*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 1999.

[Esp97] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *Proc. 2nd Int. Workshop on Verification of Infinite State Systems (INFINITY'97), Bologna, Italy, July 1997*, volume 9 of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science Publishers, 1997.

[GD89] R. Gilleron and A. Deruyver. The reachability problem for ground TRS and some extensions. In *Proc. Int. Joint Conf. Theory and Practice of Software Development (TAPSOFT'89), Barcelona, Spain, March 1989, Vol. 1*, volume 351 of *Lecture Notes in Computer Science*, pages 227–243. Springer, 1989.

[GS97] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.

[HJ99] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.

[JKM98] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98), Aalborg, Denmark, July 1998*, volume 1443 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 1998.

[KS97a] O. Kouchnarenko and Ph. Schnoebelen. A model for recursive-parallel programs. In *Proc. 1st Int. Workshop on Verification of Infinite State Systems (INFINITY'96), Pisa, Italy, Aug. 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier Science Publishers, 1997.

[KS97b] O. Kushnarenko and Ph. Schnoebelen. A formal framework for the analysis of recursive-parallel programs. In *Proc. 4th Int. Conf. Parallel Computing Technologies (PaCT'97), Yaroslavl, Russia, Sep. 1997*, volume 1277 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 1997.

[Kuč96] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proc. 16th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS'96), Hyderabad, India, Dec. 1996*, volume 1180 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 1996.

[Kuč97] A. Kučera. How to parallelize sequential processes. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 1997.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In *Proc. Int. Symp. Theoretical Aspects of Computer Software (TACS'97), Sendai, Japan, Sep. 1997*, volume 1281 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 1997.

[May97b] R. Mayr. Model checking PA-processes. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 332–346. Springer, 1997.

[May97c] R. Mayr. Tableaux methods for PA-processes. In *Proc. Int. Conf. Automated Reasoning with Analytical Tableaux and Related Methods (TABLEAUX'97), Pont-à-Mousson, France, May 1997*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 276–290. Springer, 1997.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall Int., 1989.

[Mol96] F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR'96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996.

[Rao97] J.-C. Raoult. Rational tree relations. *Bull. Belg. Math. Soc.*, 4:149–176, 1997.

[Sch99] Ph. Schnoebelen. Decomposable regular languages and the shuffle operator. *EATCS Bull.*, 67:283–289, 1999.