The Regular Viewpoint on PA-Processes

D. Lugiez[1] and Ph. Schnoebelen[2]

[1] Lab. d'Informatique de Marseille, Univ. Aix-Marseille & CNRS URA 1787, 39, r. Joliot-Curie, 13453 Marseille Cedex 13 France
email: lugiez@lim.univ-mrs.fr

[2] Lab. Spécification & Vérification, ENS de Cachan & CNRS URA 2236, 61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: phs@lsv.ens-cachan.fr

Abstract. PA is the process algebra allowing non-determinism, sequential and parallel compositions, and recursion. We suggest a view of PA-processes as *tree languages.*
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.

Introduction

Verification of Infinite State Processes is a very active field of research today in the concurrency-theory community. Of course, there has always been an active Petri-nets community, but researchers involved in process algebra and model-checking really became interested into infinite state processes after the proof that bisimulation was decidable for normed BPA-processes [BBK87]. This prompted several researchers to investigate decidability issues for BPP and BPA (with or without the normedness condition) (see [CHM94,Mol96,BE97] for a partial survey).

From BPA and BPP to PA: BPA is the "non-determinism + sequential composition + recursion" fragment of process algebra. BPP is the "non-determinism + parallel composition + recursion" fragment. PA (from [BEH95]) 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,HJ98]).

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 paper, 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 set of configurations reachable from (resp. allowing to reach) a configuration in 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 have been proved very powerful in several fields (see [CKSV97]) are useful for the process-algebraic community as well. After all, PA is just a simple term-rewrite system with a special context-sensitive rewriting strategy, not unlike head-rewriting, in presence of the sequential composition operator.
2. the syntactic congruences used to simplify notations in simple process algebras help one get 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 paper: We start with our definition of the PA algebra (§ 1). Then we recall what are tree automata and how sets of PA processes can be seen as tree languages (§ 2). This allows proving that $Post^*(L)$ and $Pre^*(L)$ are regular when L is a regular set of PA terms (§ 3). We then extend these results by taking labels of transitions into account (§ 4) and showing how transitions "modulo structural congruence" are handled (§ 5). Finally we consider the important applications in model-checking (§ 6). Several proofs are omited for lack of space. They can be found in the longer version of this paper at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV`.

Related work: The set of all reachable configurations of a pushdown automaton form a regular (word) language. This was proven in [Büc64] and extended in [Cau92]. Applications to the model-checking of pushdown automata have been proposed in [FWW97,BEM97].

$\xrightarrow{*}$ over PA terms is similar to the transitive closure of relations defined by ground rewrite systems. Because the sequential composition operator in PA implies a certain form of prefix rewriting, the *ground tree transducers* of Dauchet and Tison [DT90] cannot recognize $\xrightarrow{*}$. It turns out that $\xrightarrow{*}$ can be seen as a *rational tree relation* as defined by Raoult [Rao97].

Regarding the applications we develop for our regularity theorems, most have been suggested by Mayr's work on PA [May97c,May97b] and/or our earlier work on RPPS [KS97a,KS97b].

1 The PA process algebra

1.1 Syntax

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

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

where X is any process variable from Var. 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 (t included).

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

We write $Subterms(\Delta)$ for the union of all $Subterms(t)$ for t a right- or a left-hand side of a rule in Δ, and let $Var(\Delta)$ denotes $Var \cap Subterms(\Delta)$, the set of process variables occurring in Δ. $\Delta_a(X)$ denotes $\{t \mid$ there is a rule "$X \xrightarrow{a} t$" 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 Δ provides no rewrite.

In the following, we assume a fixed Var and Δ.

1.2 Semantics

A PA declaration Δ defines a labeled transition relation $\rightarrow_\Delta \subseteq E_{\mathrm{PA}} \times Act \times E_{\mathrm{PA}}$. We always omit the Δ 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 $k \in \mathbb{N}$, $t \xrightarrow{*} t'$, $t \rightarrow, \ldots$ $\rightarrow_\Delta$ is inductively defined via the following SOS rules:

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

$$\frac{t_2 \xrightarrow{a} t_2'}{t_1 \parallel t_2 \xrightarrow{a} t_1 \parallel t_2'} \qquad \frac{t_2 \xrightarrow{a} t_2'}{t_1.t_2 \xrightarrow{a} t_1.t_2'} IsNil(t_1)$$

The second SOS rule for sequential composition is peculiar: it uses a syntactic predicate, "$IsNil(t_1)$", as a side condition checking that t_1 cannot evolve anymore, i.e. that t_1 is terminated. Indeed, our intention is that the t_2 part in $t_1.t_2$ only evolves once t_1 is terminated.

The *IsNil*(...) predicate is inductively defined by

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

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

It is indeed a syntactic test for termination, and we have

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

1.3 Structural equivalence of PA terms

Several works on PA and related algebras only consider processes up-to some structural congruence. PA itself usually assumes an equivalence $\equiv$ defined by the following equations:

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

$\equiv$ respects the behaviour of process terms. However, *we do not want to identify PA terms related by $\equiv$!*

Our approach clearly separates the behavior of E_{PA} (the $\rightarrow$ relation) and structural equivalence between terms (the $\equiv$ relation). We get simple proofs of results which are hard to get in the other approach because the transition relation and the equivalence relation interact at each step.

In the following, we study first the $\rightarrow$ relation. Later (§ 5) we combine $\rightarrow$ and structural equivalence and show how it is possible to reason about "PA-terms modulo $\equiv$". In effect, this shows that our approach is also more general since we can define the "modulo $\equiv$" approach in our framework.

2 Tree languages and PA

We shall use tree automata to recognize sets of terms from E_{PA}.

2.1 Regular tree languages and tree automata

We recall some basic facts on tree automata and regular tree languages. For more details, the reader is referred to any classical source (e.g. [CDG+97,GS97]).

A *ranked alphabet* is a finite set of symbols $\mathcal{F}$ together with an arity function $\eta : \mathcal{F} \rightarrow \mathbb{N}$. 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 $\mathcal{T}(\mathcal{F})$.

A (finite, bottom-up) *tree automaton* (a "TA") is a tuple $\mathcal{A} = \langle \mathcal{F}, Q, F, R\rangle$ where $\mathcal{F}$ is a ranked alphabet, $Q = \{q, q', \dots\}$ 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 $f \in \mathcal{F}$. *TA's with ε-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. We write $t \overset{\mathcal{A}}{\longmapsto} q$ when $t \in \mathcal{T}(\mathcal{F})$ can be rewritten (using any 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 coincide with $L(\mathcal{A})$ for some $\mathcal{A}$ is a *regular tree language*. Regular tree languages are closed under complementation, union, etc.

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

The *size* of a TA $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the number of states of $\mathcal{A}$ augmented by the size of the rules of $\mathcal{A}$ where a rule $f(q_1, \dots, q_n) \longmapsto q$ has size $n + 2$. Notice that, for a fixed $\mathcal{F}$ where the largest arity is m, $|\mathcal{A}|$ is in $O(|Q|^{m+1})$.

A TA is *deterministic* if all transition rules have distinct left-hand sides (and there are no ε-rule). Our earlier $\mathcal{A}_{\text{even}\,g}$ example was deterministic. Given a non-deterministic TA, the classical subset construction yields a deterministic TA accepting the same language (this construction involves a potential exponential blow-up in size).

Telling whether $L(\mathcal{A})$ is empty for some TA $\mathcal{A}$ can be done in time $O(|\mathcal{A}|)$. Telling whether a given tree t is accepted by a given $\mathcal{A}$ can be done in time polynomial in $|\mathcal{A}| + |t|$.

A TA 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 $f(q_1, \dots, q_n) \to q$. By adding a sink state and the obvious rules, any $\mathcal{A}$ can be extended into a complete TA accepting the same language.

2.2 Some regular subsets of E_{PA}

E_{PA}, the set of PA-terms, can be seen as a set of trees, i.e. as $\mathcal{T}(\mathcal{F})$ for $\mathcal{F}$ given by $\mathcal{F}_0 = \{0, X, Y, \dots\}$ $(= \{0\} \cup \mathit{Var})$ and $\mathcal{F}_2 = \{., \|\}$.

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

Proposition 2. *For any t, the singleton tree language $\{t\}$ is regular, and a TA for $\{t\}$ needs only have $|t|$ states.*

The set of terminated processes is also a tree language. Write $L^{\varnothing}$ for $\{t \in E_{\mathrm{PA}} \mid IsNil(t)\}$. An immediate consequence of Lemma 1 is

Proposition 3. *$L^{\varnothing}$ is a regular tree language, and a TA for $L^{\varnothing}$ needs only have one state.*

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

Given a set $L \subseteq E_{\mathrm{PA}}$ of PA-terms, we let $Pre(L) \stackrel{\text{def}}{=} \{t \mid \exists t' \in L, t \rightarrow t'\}$ and $Post(L) \stackrel{\text{def}}{=} \{t \mid \exists t' \in L, t' \rightarrow t\}$ denotes the set of (immediate) *predecessors* (resp. *successors*) of terms in L. $Pre^+(L) \stackrel{\text{def}}{=} Pre(L) \cup Pre(Pre(L)) \cup \cdots$ and $Post^+(L) \stackrel{\text{def}}{=} Post(L) \cup Post(Post(L)) \cup \cdots$ contain the *iterated* predecessors (resp. successors). Similarly, $Pre^*(L)$ denotes $L \cup Pre^+(L)$ and $Post^*(L)$ is $L \cup Post^+(L)$, also called the *reachability set.*

In this section we prove the regularity of $Pre^*(L)$ and $Post^*(L)$ for a regular language L. $Pre^*(L)$ and $Post^*(L)$ do not take into account the labels accompanying PA transitions, but these will be considered in section 4.

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

3.1 Regularity of $Post^*(L)$

First, we give some intuition which helps understanding the construction of a TA $\mathcal{A}_{Post^*}$ accepting $Post^*(L)$.

Let us assume Δ contains $X \rightarrow r_1$ and $Y \rightarrow r_2$, and that r_1 is terminated. Starting from $t_1 = X.Y$, there exists the transition sequence $t_1 \rightarrow t_2 \rightarrow t_3$ illustrated in figure 1.

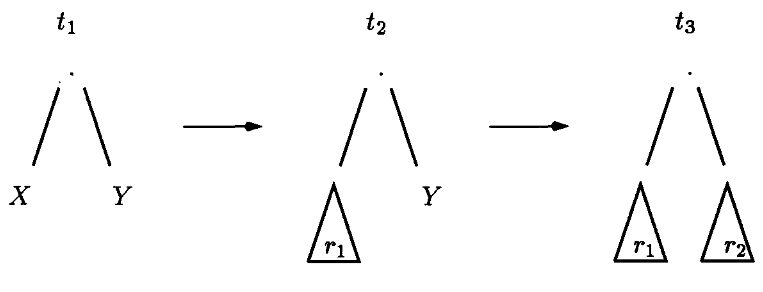

Fig. 1. An example sequence: $X.Y \rightarrow r_1.Y \rightarrow r_1.r_2$

We want to build $\mathcal{A}_{Post^*}$, a TA that reads t_3 (i.e. $r_1.r_2$) bottom-up and sees that it belongs to $Post^*(L)$. For this, the TA has to recognize that t_3 comes from t_1 (i.e. $X.Y$) and check that t_1 is in L.

1. $\mathcal{A}_{Post^*}$ must recognize that r_1 (resp. r_2) is the right-hand side of a rule $X \to r_1$ (resp. $Y \to r_2$). Therefore we need an automaton $\mathcal{A}_\Delta$ which recognizes such right-hand sides.
2. The automaton $\mathcal{A}_{Post^*}$ works on t_3 but must check that t_1 is in L. Therefore we need an automaton $\mathcal{A}_L$ accepting L. $\mathcal{A}_{Post^*}$ mimicks $\mathcal{A}_L$ but it has additional rules simulating rewrite steps: once r_1 has been recognized (by the $\mathcal{A}_\Delta$ part), the computation may continue as if X were in place of r_1. The same holds for r_2 and Y.
3. The transition between t_2 and t_3 is allowed only if r_1 is terminated. Therefore we need an automaton $\mathcal{A}_\varnothing$ to check whether a term is terminated.
4. A non-terminated term is allowed to the left of a "." when no transition has been performed to the right. Therefore we use a boolean value to indicate whether rewrite steps have been done or not.

These remarks lead to the following construction.

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:

- $\mathcal{A}_\varnothing$ is a *completely specified* automaton accepting terminated processes (see Proposition 3).
- $\mathcal{A}_L$ is a *completely specified* automaton accepting L.
- $\mathcal{A}_\Delta$ is a completely specified automaton recognizing the subterms of Δ. It has all states q_s for $s \in Subterms(\Delta)$. We ensure "$t \xrightarrow{\mathcal{A}_\Delta} q_s$ iff $s = t$" by taking as transition rules $0 \longmapsto q_0$ if $0 \in Subterms(\Delta)$, $X \longmapsto q_X$ if $X \in Subterms(\Delta)$, $q_s \parallel q_{s'} \longmapsto q_{s\parallel s'}$ (resp. $q_s.q_{s'} \longmapsto q_{s.s'}$) if $s \parallel s'$ (resp. $s.s'$) belongs to $Subterms(\Delta)$. In addition, the automaton has a sink state $q_\perp$ and the obvious transitions so that it is a completely specified automaton.
- The boolean b records whether rewrite steps have occurred.

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

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

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

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

type 2: all ε-rules of the form $(q_\varnothing, q'_L, q_s, b') \longmapsto (q_\varnothing, q_L, q_X, true)$ s.t. $X \to s$ is a rule in Δ and $X \xrightarrow{\mathcal{A}_L} q_L$.

type 3: all rules of the form
$(q_\varnothing, q_L, q_\Delta, b) \parallel (q'_\varnothing, q'_L, q'_\Delta, b') \longmapsto (\delta_\parallel(q_\varnothing, q'_\varnothing), \delta_\parallel(q_L, q'_L), \delta_\parallel(q_\Delta, q'_\Delta), b \vee b')$
type 4*a*: all rules of the form
$(q_\varnothing, q_L, q_\Delta, b).(q'_\varnothing, q'_L, q'_\Delta, \mathit{false}) \longmapsto (\delta_.(q_\varnothing, q'_\varnothing), \delta_.(q_L, q'_L), \delta_.(q_\Delta, q'_\Delta), b)$.
type 4*b*: all rules of the form
$(q_\varnothing, q_L, q_\Delta, b).(q'_\varnothing, q'_L, q'_\Delta, b') \longmapsto (\delta_.(q_\varnothing, q'_\varnothing), \delta_.(q_L, q'_L), \delta_.(q_\Delta, q'_\Delta), b \vee b')$ s.t. $q_\varnothing$ is a final state of $\mathcal{A}_\varnothing$.

This construction ensures the following lemma, whose complete proof is given in the full version of this paper.

Lemma 4. *For any* $t \in E_{\text{PA}}$, $t \overset{\mathcal{A}_{Post^*}}{\longmapsto} (q_\varnothing, q_L, q_\Delta, b)$ *iff there is some* $u \in E_{\text{PA}}$ *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* $t \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$.

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 some u accepted by $\mathcal{A}_L$ iff t belongs to $Post^*(L)$. We get our first main result:

Theorem 5. (Regularity of $Post^*(L)$**)**
(1) If L *is a regular subset of* E_{PA}, *then* $Post^*(L)$ *is regular.*
(2) Furthermore, from a TA $\mathcal{A}_L$ *recognizing* L, *is it possible to construct (in polynomial time) a TA* $\mathcal{A}_{Post^*}$ *recognizing* $Post^*(L)$. *If* $\mathcal{A}_L$ *has* k *states, then* $\mathcal{A}_{Post^*}$ *needs only have* $O(k.|\Delta|)$ *states.*

Notice that a TA for $Post^+(L)$ can be obtained just by requiring that the final states have $b = \mathit{true}$ as their fourth component.

3.2 Regularity of *Pre*(L)*

Assume we have a TA $\mathcal{A}_{Pre^*}$ recognizing $Pre^*(L)$. If we consider the same sequence $t_1 \to t_2 \to t_3$ from Fig. 1, we want $\mathcal{A}_{Pre^*}$ to accept t_1 if t_3 is in L. The TA must then read t_1, imitating the behaviour of $\mathcal{A}_L$. When $\mathcal{A}_{Pre^*}$ sees a variable (say, X), it may move to any state q of $\mathcal{A}_L$ that could be reached by some $t \in Post^*(X)$. This accounts for transitions from X, and of course we must keep track of the actual occurences of transitions so that they do not occur in the right-hand side of a "." when the left-hand side is not terminated.

This leads to the following construction:

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

- $\mathcal{A}_\varnothing$ is a *completely specified* automaton accepting terminated processes (see Proposition 3).
- $\mathcal{A}_L$ is the automaton accepting L.
- The boolean b records 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 $Q_{\ldots}$ denotes the set of states of the relevant automaton.

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

type 0: all rules of the form $0 \longmapsto (q_\varnothing, q_L, false)$ 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 $(q_\varnothing, q_L, b) \parallel (q'_\varnothing, q'_L, b') \longmapsto (\delta_\parallel(q_\varnothing, q'_\varnothing), \delta_\parallel(q_L, q'_L), b \vee b')$.
type 3a: all rules of the form $(q_\varnothing, q_L, b).(q'_\varnothing, q'_L, b') \longmapsto (\delta_.(q_\varnothing, q'_\varnothing), \delta_.(q_L, q'_L), b \vee b')$ s.t. $q_\varnothing$ is a final state of $\mathcal{A}_\varnothing$.
type 3b: all rules of the form $(q_\varnothing, q_L, b).(q'_\varnothing, q'_L, false) \longmapsto (\delta_.(q_\varnothing, q'_\varnothing), \delta_.(q_L, q'_L), b)$.

This construction allows the following lemma, whose complete proof is given in the full version of this paper.

Lemma 6. *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 $p = 0$).*

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). This is summarized by the next theorem.

Theorem 7. (Regularity of $Pre^*(L)$)
(1) If L is a regular subset of E_{PA}, then $Pre^*(L)$ is regular.
(2) Furthermore, from an automaton $\mathcal{A}_L$ recognizing L, is it 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 6. Observe that the regularity result does not need the finiteness of Δ (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 $u \overset{\mathcal{A}_\varnothing}{\longmapsto} q_\varnothing$ and $u \overset{\mathcal{A}_L}{\longmapsto} q_L$ iff the the language accepted by the final states $\{(q_X, q_\varnothing, q_L) \mid q_X \text{ a final state of } \mathcal{A}_X\}$ 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 Δ to build the $\mathcal{A}_X$'s. □

Actually, the $\xrightarrow{*}$ relation between PA-terms is a *rational tree relation* in the sense of [Rao97]. This entails that $Pre^*(L)$ and $Post^*(L)$ are regular tree languages when L is. Raoult's approach is more powerful than our elementary constructions but it relies on complex new tools (much more powerful than usual

TA's) and does not provide the straightforward complexity analysis we offer. Moreover, the extensions we discuss in section 4 would be more difficult to obtain in his framework.

3.3 Applications

Theorems 5 and 7 already give us simple solutions to verification problems over PA: the *reachability problem* asks, given t, u (and Δ), whether $t \xrightarrow{*} u$. The *boundedness problem* asks whether $Post^*(t)$ is finite. They can be solved in polynomial time just by looking at the TA for $Post^*(t)$. Variant problems such as "*can we reach terms with arbitrarily many occurences of X in parallel ?*" can be solved equally easily.

4 Reachability under constraints

In this section, we consider *reachability under constraints*, that is, reachability where the labels of transitions must respect some criterion. 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, in general, the problem of telling whether $t \xrightarrow{C}$ (i.e. whether $Post^*[C](t)$ is not empty) is undecidable for the PA algebra even if we assume regularity of C [1]. 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^* and $Post^*$ of a regular tree language).

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

Definition 8. $\{(C_1, C_1'), \dots, (C_m, C_m')\}$ is a (finite) *seq-decomposition* of C iff for all $w, w' \in Act^*$ we have

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

$\{(C_1, C_2'), \dots, (C_m, C_m')\}$ is a (finite) *paral-decomposition* of C iff for all $w, w' \in Act^*$ we have

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

[1] E.g. by using two copies $\underline{a}, \overline{a}$ of every letter a in some Σ, and by using the regular constraint $C \overset{\text{def}}{=} (\underline{a_1}.\overline{a_1} + \cdots + \underline{a_n}.\overline{a_n})^* \underline{\#}.\overline{\#}$, we can state with "$(\underline{t_1} \parallel \overline{t_2}) \xrightarrow{C}$?" that t_1 and t_2 share a common trace ending with $\#$. This can be used to encode the (undecidable) empty-intersection problem for context-free grammars.

The crucial point of the definition is 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 ε. Observe that the formal difference between seq-decomposition and paral-decomposition comes from the fact that $w \parallel w'$, the set of all shuffles of w and w' may contain several elements.

Definition 9. 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 language C is *decomposable* if it appears in a finite decomposition system.

Not all $C \subseteq Act^*$ are decomposable, e.g. $(ab)^*$ is not. It is known that decomposable languages are regular and that all commutative regular languages are decomposable. (Write $w \sim w'$ when w' is a permutation of w. A commutative language is a language C closed w.r.t. $\sim$). Simple examples of commutative languages are obtained by considering the number of occurrences (rather than the positions) of given letters: for any positive weight function θ given by $\theta(w) \overset{\text{def}}{=} \sum_i n_i |w|_{a_i}$ with $n_i \in \mathbb{N}$, the set C of all w s.t. $\theta(w) = k$ (or $\theta(w) < k$, or $\theta(w) > k$, or $\theta(w) = k \mod k'$) is a commutative regular language, hence is decomposable.

However, a decomposable language needs not be commutative: finite languages are decomposable, and decomposable languages are closed by union, concatenation and shuffle.

Theorem 10. (Regularity)
For any regular $L \subseteq E_{\text{PA}}$ and any decomposable C, $Pre^*[C](L)$ and $Post^*[C](L)$ are regular tree languages.

Proof. The construction is similar to the constructions for $Pre^*(L)$ and $Post^*(L)$. See the full version of the paper. □

5 Handling structural equivalence of PA-terms

In this section we show how to take into account the axioms $(A_.)$, $(C_\parallel)$, $(A_\parallel)$ and (N_1) to (N_4) (from section 1.3) defining the structural equivalence on E_{PA} terms.

Some definitions of PA consider PA-terms modulo $\equiv$. This viewpoint assumes that a PA-term t really denotes an equivalence class $[t]_\equiv$, and that transitions are defined between such equivalence classes, coinciding with a transition relation we would define by

$$[t]_\equiv \xrightarrow{a} [u]_\equiv \overset{\text{def}}{\Leftrightarrow} \exists t' \in [t]_\equiv, u' \in [u]_\equiv \text{ s.t. } t' \xrightarrow{a} u'. \tag{1}$$

This yields a new process algebra: $\text{PA}_\equiv$.

In our framework, we can *define* a new transition relation between PA-terms: $t \overset{a}{\Rightarrow} t'$ iff $t \equiv u \xrightarrow{a} u' \equiv t'$ for some u, u', i.e. $[t]_\equiv \xrightarrow{a} [u]_\equiv$. We adopt the usual abbreviations $\overset{*}{\Rightarrow}$, $\overset{k}{\Rightarrow}$ for $k \in \mathbb{N}$, etc.

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

Proposition 11. *$\equiv$ has the transfer property, i.e. it 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).*

Proof. Check this for each equation, then deal with the general case by using congruence property of $\equiv$ and structural induction over terms, transitivity of $\equiv$ and induction over the number of equational replacements needed to relate t and t'. Observe that *IsNil* is compatible with $\equiv$. □

Proposition 12. *$t \overset{k}{\Rightarrow} u$ iff $t \xrightarrow{k} u'$ for some $u' \equiv u$.*

The reachability problem solved by Mayr actually coincides with "reachability modulo $\equiv$" or "reachability through $\overset{*}{\Rightarrow}$". Our tree automata method can deal with this, as we now show.

5.1 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 u 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, we have

$$t \equiv t' \quad \text{iff} \quad t \searrow u =_P u' \swarrow t' \text{ for some } u, u' \quad \text{iff} \quad t\downarrow =_P t'\downarrow. \tag{2}$$

Lemma 13. *For any t, the set $[t]_{=_P} \overset{\text{def}}{=} \{u \mid t =_P u\}$ is a regular tree language, and an automaton for $[t]_{=_P}$ needs only have $m.(m/2)!$ states if $|t| = m$.*

Note that for a regular L, $[L]_{=_P}$ (and $[L]_\equiv$) are not necessarily regular.

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 14. *For any regular L, the sets $[L]_{\searrow}$, $[L]_{\swarrow}$, and $[L]\downarrow$ are regular tree languages. From an automaton $\mathcal{A}$ recognizing L, we can build automata for these three languages in polynomial time.*

Corollary 15. *"Boundedness modulo $\equiv$" of the reachability set is decidable in polynomial-time.*

Proof. 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. □

We can also combine (2) and lemmas 13 and 14 and have

Proposition 16. *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 $|t| = m$.*

Now it is easy to prove decidability of the reachability problem modulo $\equiv$: $t \overset{*}{\Rightarrow} u$ iff $Post^*(t) \cap [u]_{\equiv} \neq \emptyset$. 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 [2]:

Theorem 17. *The reachability problem in* $\text{PA}_{\equiv}$, *"given t and u, do we have $t \overset{*}{\Rightarrow} u$?", 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 (see long version).
NP-easiness is straightforward in the automata framework. We have $t \overset{*}{\Rightarrow} u$ 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 NP 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. □

6 Model-checking PA processes

In this section we show a simple approach to the model-checking problem which is an immediate application of our main regularity theorems. We do not consider the structural equivalence $\equiv$ until section 6.3, where we show that the decidability results are a simple consequence of our previous results.

[2] First proved in [May97c]

6.1 Model-checking in E_{PA}

We consider a set $\mathit{Prop} = \{P_1, P_2, \ldots\}$ of *atomic propositions*. For $P \in \mathit{Prop}$, Let $\mathit{Mod}(P)$ denotes the set of PA processes for which P holds. We only consider propositions P such that $\mathit{Mod}(P)$ 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

$$\begin{aligned} t \models P &\overset{\text{def}}{\Leftrightarrow} t \in \mathit{Mod}(P), \\ t \models \neg\varphi &\overset{\text{def}}{\Leftrightarrow} t \not\models \varphi, \\ t \models \varphi \wedge \varphi' &\overset{\text{def}}{\Leftrightarrow} t \models \varphi \text{ and } t \models \varphi', \end{aligned} \qquad \begin{aligned} t \models \mathsf{EX}\varphi &\overset{\text{def}}{\Leftrightarrow} t' \models \varphi \text{ for some } t \rightarrow t', \\ t \models \mathsf{EF}\varphi &\overset{\text{def}}{\Leftrightarrow} t' \models \varphi \text{ for some } t \xrightarrow{*} t'. \end{aligned}$$

Thus $\mathsf{EX}\varphi$ reads "it is possible to reach in one step a state s.t. φ" and $\mathsf{EF}\varphi$ reads "it is possible to reach (via some sequence of steps) a state s.t. φ".

Definition 18. The *model-checking problem* for EF over PA has as inputs: a given Δ, a given t in E_{PA}, a given φ in EF. The answer is yes iff $t \models \varphi$.

We now extend the definition of *Mod* to the whole of EF: $\mathit{Mod}(\varphi) \overset{\text{def}}{=} \{t \in E_{\mathrm{PA}} \mid t \models \varphi\}$, we have

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

Theorem 19. *(1) For any EF formula φ, $\mathit{Mod}(\varphi)$ is a regular tree language. (2) If we are given tree-automata $\mathcal{A}_P$'s recognizing the regular sets $\mathit{Mod}(P)$, then a tree-automaton $\mathcal{A}_\varphi$ recognizing $\mathit{Mod}(\varphi)$ can be built effectively.*

This gives us a decision procedure for the model-checking problem: build an automaton for $\mathit{Mod}(\varphi)$ and check whether it accepts t. Observe that computing a representation of $\mathit{Mod}(\varphi)$ is more general than just telling whether a given t belongs to it. Observe also that our results allow model-checking approches based on combinations of forward and backward methods (while Theorem 19 only relies on the standard backward approach.)

The above procedure is non-elementary since every nesting level of negations potentially induces an exponential blowup. Actually, negations in φ can be pushed towards the leaves and only stop at the EF's, so that really the tower of exponentials depend on the maximal number of alternations between negations and EF's in φ. The procedure described in [May97b] is non-elementary and today the known lower bound is PSPACE-hard.

6.2 Model-checking with constraints

We can also use the constraints introduced in section 4 to define an extended EF logic where we now allow all $\langle C\rangle\varphi$ formulas for decomposable C. The meaning is given by $Mod(\langle C\rangle\varphi) \stackrel{\text{def}}{=} Pre^*[C](Mod(\varphi))$. This is quite general and immediately include the extensions proposed in [May97b].

6.3 Model-checking modulo $\equiv$

The model-checking problem solved in [May97b] considers the EF logic over $\mathrm{PA}_{\equiv}$.

In this framework, the semantics of EF-formulas is defined over equivalence classes, or equivalently, using the $\overset{a}{\Rightarrow}$ relation and only considering atomic propositions P s.t. $Mod(P)$ is closed under $\equiv$.

But if the $Mod(P)$'s are closed under $\equiv$, then $t \models \varphi$ in PA iff $t \models \varphi$ in $\mathrm{PA}_{\equiv}$ (a consequence of Proposition refprop-equiv-transfer), so that our earlier tree-automata algorithm can be used to solve the model-checking problem for $\mathrm{PA}_{\equiv}$. We can also easily allow constraints like in the previous section.

Conclusion

In this paper 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 understand why some problems can be solved in P-time, some others in NP-time, etc. It is quite versatile and we believe that many variants of PA can be attacked with the same approach.

We certainly did not list all possible applications of the tree-automata approach for verification problems in PA. Future work should aim at better understanding which problems can benefit from our TA viewpoint and techniques.

Acknowledgments We thank H. Comon and R. Mayr for their numerous suggestions, remarks and questions about this work.

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-Verlag, 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. 30–31, 1996*, volume 5 of *Electronic Notes in Theor. Comp. Sci.* Elsevier, 1997.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proc. 22nd ACM Symp. Principles of Programming Languages (POPL'95), San Francisco, CA, USA, Jan. 1995*, pages 95–106, 1995.

[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-Verlag, 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.

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

[CDG$^+$97] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata and their application, 1997. A preliminary version of this (yet unpublished) book is available at `http://l3ux02.univ-lille3.fr/tata`.

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

[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.

[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 11–12, 1997*, volume 9 of *Electronic Notes in Theor. Comp. Sci.* Elsevier, 1997.

[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-Verlag, 1997.

[HJ98] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed Process Algebra. Research Report ECS-LFCS-98-386, Lab. for Foundations of Computer Science, Edinburgh, May 1998.

[JKM98] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. Tech. Report TUM-I9805, Institut für Informatik, TUM, Munich, Germany, February 1998. To appear in Proc. ICALP'98, Aalborg, DK, July 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, 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-Verlag, 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-Verlag, 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-Verlag, 1997.

[May97a] R. Mayr. Combining Petri nets and PA-processes. In *Proc. 4th Int. Symp. Theoretical Aspects Computer Software (TACS'97), Sendai, Japan, Sep. 1997*, volume 1281 of *Lecture Notes in Computer Science*, pages 547–561. Springer-Verlag, 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-Verlag, 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-Verlag, 1997.

[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-Verlag, 1996.

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