FI MU

Faculty of Informatics
Masaryk University

Regularity is Decidable for Normed PA Processes in Polynomial Time

by

Antonín Kučera

FI MU Report Series **FIMU-RS-96-01**

 February 1996

Regularity is Decidable for Normed PA Processes in Polynomial Time

Antonín Kučera
e-mail: tony@fi.muni.cz

Faculty of Informatics, Masaryk University
Botanická 68a, 60200 Brno
Czech Republic

Abstract

A process Δ is regular if it is bisimilar to a process Δ' with finitely many states. We prove that regularity of normed PA processes is decidable and we present a practically usable polynomial-time algorithm. Moreover, if the tested normed PA process Δ is regular then the process Δ' can be effectively constructed. It implies decidability of bisimulation equivalence for any pair of processes such that one process of this pair is a normed PA process and the other process has finitely many states.

1 Introduction

We consider the problem of deciding regularity of normed PA processes. A process Δ is regular if there is a process Δ' with finitely many states such that $\Delta \sim \Delta'$. Finite-state processes have been intensively studied in the last decades (see e.g. [Mil89]). Almost all interesting properties are decidable for finite-state processes. Moreover, designed algorithms are practically usable.

This is no more true if one moves to process classes which contain also processes with infinitely many states (up to bisimilarity). Some problems can remain decidable—for example, bisimilarity is known to be decidable for BPA (see [BBK87, Cau88, Gro91, HS91, CHS92]) and BPP (see [CHM93])

processes. The same problem becomes undecidable for labelled Petri nets (see [Jan94]). But even if a given property is decidable, the algorithm is usually not interesting from the practical point of view due to its complexity. Before running a complex algorithm, it is a good idea to ask whether the process we are dealing with can be replaced with some equivalent (bisimilar) process with finitely many states. If so, we can usually run a much more efficient algorithm. Natural questions are, whether the regularity is decidable for a given class of processes and whether the equivalent finite-state process can be effectively constructed.

Mauw and Mulder showed in [MM94] that "regularity" of BPA processes is decidable. The quotes are important here because Mauw and Mulder used the word regularity in a different sense—a BPA process is "regular" if each of its variables denotes a regular process. This notion is thus strongly dependent on BPA syntax. It is not clear how to define "regularity" e.g. for Petri nets. However, with a help of this result one can easily conclude that regularity is decidable for normed BPA processes (see [Kuč95]). A similar result holds for normed BPP processes (see [Kuč95]). Both algorithms are polynomial and easy to implement.

A recent result of Esparza and Jančar [EJ96] says that regularity is decidable for labelled Petri nets. The algorithm is obtained by a combination of two semi-decidability results and hence there are no complexity estimations. Furthermore, Burkart, Caucal and Steffen showed in [BCS96] that regularity is decidable for all BPA processes.

An interesting related problem is decidability of various behavioural equivalences and preorders for pairs of processes such that one process of this pair is regular. For example, Jančar and Moller proved in [JM95] that bisimilarity is decidable for a pair of labelled Petri nets provided one net of this pair is bounded (regular). The same result holds for trace equivalence and simulation equivalence.

In this paper we prove that regularity is decidable for normed PA processes. PA processes appeared as a natural subclass of ACP processes (see [BW90]). It is strictly greater than the union of normed BPP and normed BPA processes and it is incomparable with the class of labelled Petri nets. Our regularity test for normed PA processes is of polynomial time complexity. Moreover, if the tested normed PA process is regular then we can also construct a bisimilar finite-state process—and therefore we can also decide bisimilarity for pairs of processes such that one process of this pair is a normed PA process and the other has finitely many states. The problem of

decidability of bisimulation equivalence for (normed) PA processes is open, hence this result can be seen as the first small step towards the solution.

2 Basic definitions

2.1 PA processes

Let $Act = \{a, b, c, \ldots\}$ be a countably infinite set of *atomic actions*. Let $Var = \{X, Y, Z, \ldots\}$ be a countably infinite set of *variables* such that $Var \cap Act = \emptyset$. The class of recursive PA expressions is defined by the following abstract syntax equations:

$$E_{PA} \ ::= \ a \mid X \mid E_{PA}.E_{PA} \mid E_{PA} \| E_{PA} \mid E_{PA} \between E_{PA} \mid E_{PA} + E_{PA}$$

Here a ranges over Act and X ranges over Var. The symbol Act^* denotes the set of all finite strings over Act.

As usual, we restrict our attention to guarded expressions. A PA expression E is *guarded* if every variable occurence in E is within the scope of an atomic action.

A *guarded PA process* is defined by a finite family Δ of recursive process equations

$$\Delta = \{X_i \stackrel{def}{=} E_i \quad | \quad 1 \leq i \leq n\}$$

where X_i are distinct elements of Var and E_i are guarded PA expressions, containing variables from $\{X_1, \ldots, X_n\}$. The set of variables which appear in Δ is denoted by $Var(\Delta)$.

The variable X_1 plays a special role (X_1 is sometimes called "the leading variable")—it is a root of a labelled transition system, defined by the process Δ and following rules:

$$\frac{}{a \stackrel{a}{\to} \epsilon} \qquad \frac{E \stackrel{a}{\to} E'}{E.F \stackrel{a}{\to} E'.F} \qquad \frac{E \stackrel{a}{\to} E'}{E + F \stackrel{a}{\to} E'} \qquad \frac{F \stackrel{a}{\to} F'}{E + F \stackrel{a}{\to} F'}$$

$$\frac{E \stackrel{a}{\to} E'}{E \| F \stackrel{a}{\to} E' \| F} \qquad \frac{F \stackrel{a}{\to} F'}{E \| F \stackrel{a}{\to} E \| F'} \qquad \frac{E \stackrel{a}{\to} E'}{E \between F \stackrel{a}{\to} E' \| F} \qquad \frac{E \stackrel{a}{\to} E'}{X \stackrel{a}{\to} E'} \ (X \stackrel{def}{=} E \in \Delta)$$

The symbol ϵ denotes the empty expression with usual conventions: $\epsilon \| E = E$, $E \| \epsilon = E$, $\epsilon.E = E$, $\epsilon \between E = E$ and $E \between \epsilon = E$. Nodes of the transition system generated by Δ are PA expressions, which are often called *states of* Δ, or just "states" when Δ is understood from the context. We also define the

relation $\stackrel{w}{\rightarrow}^*$ where $w \in Act^*$ as the reflexive and transitive closure of $\stackrel{a}{\rightarrow}$ (we often write $E \rightarrow^* F$ instead of $E \stackrel{w}{\rightarrow}^* F$ if w is irrelevant). Given two states E, F, we say that F is *reachable from* E, if $E \rightarrow^* F$. States of Δ which are reachable from X_1 are said to be *reachable*.

2.1.1 Bisimulation

The equivalence between process expressions (states) we are interested in here is *bisimilarity* [Par81], defined as follows:

Definition 1. *A binary relation R over process expressions is a* bisimulation *if whenever $(E, F) \in R$ then for each $a \in Act$*

- *if $E \stackrel{a}{\rightarrow} E'$, then $F \stackrel{a}{\rightarrow} F'$ for some F' such that $(E', F') \in R$*
- *if $F \stackrel{a}{\rightarrow} F'$, then $E \stackrel{a}{\rightarrow} E'$ for some E' such that $(E', F') \in R$*

Processes Δ and Δ' are bisimilar*, written $\Delta \sim \Delta'$, if their leading variables are related by some bisimulation.*

2.1.2 Normed processes

An important subclass of PA processes can be obtained by an extra restriction of *normedness*. A variable $X \in Var(\Delta)$ is *normed* if there is $w \in Act^*$ such that $X \stackrel{w}{\rightarrow}^* \epsilon$. In that case we define the *norm* of X, written $[X]$, to be the length of the shortest such w. Thus $[X] = \min\{\text{length}(w) \mid X \stackrel{w}{\rightarrow}^* \epsilon\}$. A process Δ is *normed*, if all variables of $Var(\Delta)$ are normed. The norm of Δ is then defined to be the norm of X_1.

2.1.3 A normal form for PA processes

Before we present a normal form for PA processes, we need to introduce the set of *VPA expressions* defined inductively as follows:

1. The empty expression ϵ is a *VPA* expression.
2. Each variable $X \in Var$ is a *VPA* expression.
3. If α, β are nonempty *VPA* expressions, then $\alpha.\beta$, $\alpha \| \beta$ and $\alpha \between \beta$ are *VPA* expressions.

4. Each *VPA* expression can be constructed using the rules 1, 2 and 3 in a finite number of steps.

We use Greek letters $\alpha, \beta, \ldots$ to range over *VPA* expressions. The set of *VPA* expressions which contain only variables from $Var(\Delta)$, where Δ is a PA process, is denoted $VPA(\Delta)$. Finally, the set of variables which appear in a *VPA* expression α is denoted $Var(\alpha)$.

Definition 2. *A PA process Δ is said to be in* normal form *if all its equations are of the form*

$$X_i \stackrel{def}{=} \sum_{j=1}^{n_i} a_{ij}\alpha_{ij}$$

where $1 \leq i \leq n$, $n_i \in N$, $a_{ij} \in Act$ and $\alpha_{ij} \in VPA(\Delta)$. Moreover, we also require that for each X_i, $1 \leq i \leq n$ there is a reachable state $\alpha \in VPA(\Delta)$ such that $X_i \in Var(\alpha)$.

Any PA process can be effectively presented in normal form (see [BEH95]). From now on we assume that all PA processes we are working with are presented in normal form. This justifies also the assumption that all reachable states of a PA process Δ are elements of $VPA(\Delta)$.

2.2 Regular processes

The main question considered in this paper is whether regularity of normed PA processes is decidable. The next definition explains what is meant by the notion of regularity.

Definition 3. *A process Δ is* regular *if there is a process Δ' with finitely many states such that $\Delta \sim \Delta'$.*

It is easy to see that a process is regular iff it can reach only finitely many states up to bisimilarity. In [Mil89] it is shown that regular processes can be represented in the following normal form:

Definition 4. *A regular process Δ is said to be in normal form if all its equations are of the form*

$$X_i \stackrel{def}{=} \sum_{j=1}^{n_i} a_{ij}X_{ij}$$

where $1 \leq i \leq n$, $n_i \in N$, $a_{ij} \in Act$ and $X_{ij} \in Var(\Delta)$.

Thus a process Δ is regular iff there is a regular process Δ' in normal form such that $\Delta \sim \Delta'$. In the next section we show that regularity of normed PA processes is decidable. Moreover, if a given normed PA process Δ is regular then the process Δ' can be effectively constructed.

Lemma 1. *A process Δ is not regular iff there is an infinite path $X_1 = \alpha_0 \stackrel{a_0}{\to} \alpha_1 \stackrel{a_1}{\to} \alpha_2 \stackrel{a_2}{\to} \cdots$ such that $\alpha_i \not\sim \alpha_j$ for $i \neq j$.*

Proof: It can be found e.g. in [Kuč95]. □

3 Decidability of regularity for normed PA processes

3.1 The inheritance tree

Let Δ be a normed PA process. The aim of the following definition is to describe all variables in a state $\alpha \in VPA(\Delta)$ which can potentially emit an action:

Definition 5. *Let Δ be a normed PA process. For each $\alpha \in VPA(\Delta)$ we define the set $FIRE(\alpha)$ in the following way:*

$$FIRE(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \epsilon \\ \{X\} & \text{if } \alpha = X \\ FIRE(\beta_1) & \text{if } \alpha = \beta_1.\beta_2 \text{ or } \alpha = \beta_1 \lfloor\!\lfloor \beta_2 \\ FIRE(\beta_1) \cup FIRE(\beta_2) & \text{if } \alpha = \beta_1 \| \beta_2 \end{cases}$$

The following function is needed in some proofs of this section:

Definition 6. *The function Length : $VPA \to N \cup \{0\}$ returns for each $\alpha \in VPA$ the number of variables contained in α, distinguishing multiple occurence of the same variable.*

Lemma 2. *Let Δ be a normed PA process, $\alpha \in VPA(\Delta)$. Then for each $X \in Var(\alpha)$ there is $\beta \in VPA(\Delta)$ such that $\alpha \to^* \beta$ and $X \in FIRE(\beta)$.*

The following concept stands behind many constructions of this paper:

Definition 7. *For each $\alpha \in VPA$ we define the set $Tail(\alpha) \subseteq Var$ in the following way:*

$$Tail(\alpha) = \begin{cases} \{X\} & \text{if } \alpha = X \\ \emptyset & \text{if } \alpha = \epsilon \text{ or } \alpha = \beta \| \gamma \text{ where } \beta \neq \epsilon \neq \gamma \\ Tail(\gamma) - Var(\beta) & \text{if } \alpha = \beta.\gamma \text{ or } \alpha = \beta \between \gamma \text{ where } \beta \neq \epsilon \neq \gamma \end{cases}$$

Remark 1. *The set $Tail(\alpha)$ provides two important pieces of information:*

1. *If $X \in Var(\alpha)$ such that $X \notin Tail(\alpha)$, then there is α' such that $\alpha \rightarrow^* \alpha'$, $X \in FIRE(\alpha')$ and $Length(\alpha') \geq 2$.*

2. *If $X \in Tail(\alpha)$, then the only occurence of X in α can become active (i.e. X can emit an action) after all other variables disappear.*

Definition 8. *Let Δ be a normed PA process. A variable $X \in Var(\Delta)$ is* growing *if there is $\alpha \in VPA(\Delta)$ such that $X \rightarrow^* \alpha$, $X \in FIRE(\alpha)$ and $Length(\alpha) \geq 2$.*

Lemma 3. *Let Δ be a normed PA process. The problem whether $Var(\Delta)$ contains a growing variable is decidable in polynomial time.*

Proof: We define the binary relation $GROW$ on $Var(\Delta)$ in the following way:

$$(X, Y) \in GROW \stackrel{def}{\Longleftrightarrow} \exists \beta \in VPA(\Delta) \text{ such that } X \rightarrow^* \beta \text{ where } Length(\beta) \geq 2 \text{ and } Y \in FIRE(\beta).$$

Clearly $Var(\Delta)$ contains a growing variable iff there is $X \in Var(\Delta)$ such that $(X, X) \in GROW$. We show that the relation $GROW$ can be effectively constructed in polynomial time. We need two auxiliary binary relations on $Var(\Delta)$:

$X \rightsquigarrow Y \quad \stackrel{def}{\Longleftrightarrow}$ there is a summand $a\alpha$ in the defining equation for X in Δ such that $Length(\alpha) \geq 2$, $Y \in Var(\Delta)$ and $Y \notin Tail(\alpha)$

$X \hookrightarrow Y \quad \stackrel{def}{\Longleftrightarrow}$ there is a summand $a\alpha$ in the defining equation for X in Δ such that $Y \in Var(\alpha)$.

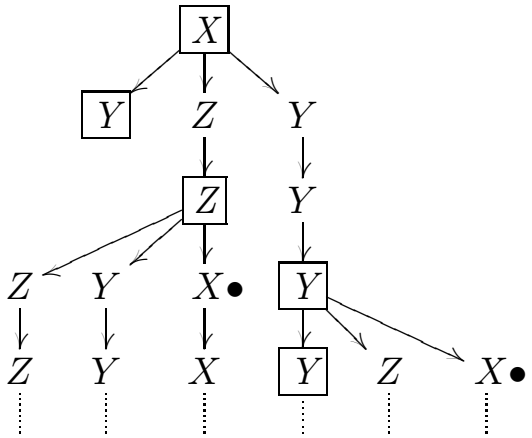
It is easy to prove that $GROW = \hookrightarrow^* . \rightsquigarrow . \hookrightarrow^*$ where $\hookrightarrow^*$ denotes the reflexive and transitive closure of $\hookrightarrow$. Moreover, the composition $\hookrightarrow^* . \rightsquigarrow . \hookrightarrow^*$ can be constructed in polynomial time. $\square$

Let Δ be a normed PA process. If Δ is not regular then there is (due to Lemma 1) an infinite path $\mathcal{P}$ of the form $X_1 = \alpha_0 \stackrel{a_0}{\to} \alpha_1 \stackrel{a_1}{\to} \alpha_2 \stackrel{a_2}{\to} \cdots$ such that $\alpha_i \not\sim \alpha_j$ for $i \neq j$. To be able to examine properties of $\mathcal{P}$ in a detail, we define for $\mathcal{P}$ the corresponding *inheritance tree*, denoted $IT_{\mathcal{P}}$. The aim of this construction is to describe the relationship between variables which are located in successive states of $\mathcal{P}$. The way how $IT_{\mathcal{P}}$ is constructed is similar to the construction of a derivation tree for a word $w \in L(G)$ where $L(G)$ is a language generated by a context-free grammar G. We start with an example which shows how $IT_{\mathcal{P}}$ looks for a given prefix of $\mathcal{P}$.

Example 1. *Let Δ be a normed PA process given by the following set of equations:*

$$\{\ X \stackrel{def}{=} b\ +\ a(Y.(Z\|Y)),\ Y \stackrel{def}{=} c\ +\ b(Y.Z.X),\ Z \stackrel{def}{=} a\ +\ a((Z\|Y).X)\ \}$$

Let $\mathcal{P}\ =\ X \stackrel{a}{\to} Y.(Z\|Y) \stackrel{c}{\to} Z\|Y \stackrel{a}{\to} ((Z\|Y).X)\|Y \stackrel{b}{\to} ((Z\|Y).X)\|(Y.Z.X)\cdots$. *If we draw a fragment of $IT_{\mathcal{P}}$, we get the following picture:*



Nodes of $IT_{\mathcal{P}}$ are labelled with variables of $Var(\Delta)$. The state $\alpha_i, i \in N \cup \{0\}$ of $\mathcal{P}$ corresponds to the set of nodes in $IT_{\mathcal{P}}$ which have the distance i from the root of $IT_{\mathcal{P}}$ (the root itself has the distance 0). This set of nodes is called the i^{th} *Level* of $IT_{\mathcal{P}}$. Each transition $\alpha_i \stackrel{a_i}{\to} \alpha_{i+1}$ is due to a single variable $A \in Var(\alpha_i)$ and a transition $A \stackrel{a_i}{\to} \gamma$ where the expression $a_i\gamma$ is a summand in the defining equation for A in Δ (see Definition 2). Moreover, α_{i+1} can be obtained from α_i by replacing one occurence of A with γ (here we must distinguish between multiple occurence of the variable A within the state α_i). We call the variable A the *active variable* of α_i and the transition $A \stackrel{a_i}{\to} \gamma$ the *step* of α_i. The nodes of $IT_{\mathcal{P}}$ which correspond to active variables are called *active*. Each active node is placed within a box in the previous example.

Nodes and edges of $IT_{\mathcal{P}}$ are defined inductively—we define all nodes in the *Level* $i+1$ together with their labels, using the nodes from the *Level* i. Moreover, we also define all edges between nodes in these two levels.

1. **i=0:** There is just one node N in the *Level* 0 — the root, labelled X_1.
2. **induction step:** Let us suppose that nodes of *Level* i have been already defined. For each node U from *Level* i we define its immediate successors. There are two possibilities:
   - **U is not active:** Then U has just one immediate successor whose label is the same as the label of U.
   - **U is active:** Let $A \xrightarrow{a_i} \gamma$ be the step of α_i and let $n = Length(\gamma)$. The node U (whose label is A) has n immediate successors (if $n = 0$ then U is a leaf). The label of the l^{th} immediate successor of U is the l^{th} variable from γ, reading γ from left to right. Here l ranges from 1 to n. As we cannot afford to lose the information about the structure of γ completely, we distinguish the case when $Tail(\gamma) = \{B\}$ where $B \in Var(\Delta)$. Then we say, that the last successor of U is a *tail* of U. In the example above, tails are marked with a black dot.

A node of $IT_{\mathcal{P}}$ which has at least two immediate successors is called a *branching node*. Branching nodes are especially important because their labels are potential candidates to be growing. This is the basic idea which stands behind the notion of the *Allow set*.

Definition 9. *For each node U of $IT_{\mathcal{P}}$ we define the set $Allow(U) \subseteq Var(\Delta)$ in the following way:*

- *If U is the root of $IT_{\mathcal{P}}$, then $Allow(U) = Var(\Delta)$.*
- *If U is an immediate successor of a node V, then*
  - *If V is not branching, then $Allow(U) = Allow(V)$.*
  - *If V is branching and U is not a tail of V, then $Allow(U) = Allow(V) - \{Label(V)\}$.*
  - *If V is branching and U is a tail of V, then $Allow(U) = Allow(V)$.*

The next lemma explains what is the relationship between a node U and the set $Allow(U)$:

Lemma 4. *Let U be a node of $IT_{\mathcal{P}}$. If $Label(U) \notin Allow(U)$ then $Label(U)$ is a growing variable.*

Now we prove the first main theorem of this paper:

Theorem 1. *A normed PA process Δ is regular iff $Var(\Delta)$ does not contain any growing variable.*

Proof:
($\Rightarrow$) : Let $X \in Var(\Delta)$ be a growing variable. We show that Δ can reach infinitely many pairwise non-bisimilar states. To do this, it suffices to show that for any $k \in N$ there is a reachable state $\alpha \in VPA(\Delta)$ such that $[\alpha] \geq k$ (bisimilar processes must have the same norm). As X is growing, there is $\gamma \in VPA(\Delta)$ such that $X \rightarrow^* \gamma$, $X \in FIRE(\gamma)$ and $Length(\gamma) \geq 2$. Moreover, there is a reachable state $\beta_1 \in VPA(\Delta)$ such that $X \in FIRE(\beta_1)$ (it follows from the Definition 2 and Lemma 2). Thus $\beta_1 \rightarrow^* \beta_2$ where β_2 is obtained from β_1 by replacing one occurence of X with γ. As $X \in FIRE(\beta_1)$, each variable from $FIRE(\gamma)$ belongs to $FIRE(\beta_2)$—hence $X \in FIRE(\beta_2)$. Moreover, $Length(\beta_2) > Length(\beta_1)$ because $Length(\gamma) \geq 2$. As $X \in FIRE(\beta_2)$, we can repeat this construction producing β_3 and so on. As $Length(\beta_i) > Length(\beta_j)$ for each $i > j$, the state β_k has the property $Length(\beta_k) \geq k$, thus $[\beta_k] \geq k$.

($\Leftarrow$) : This part of the proof is more complicated. The basic scheme is similar to the method which was used by Mauw and Mulder in [MM94] and can be described in the following way: We need to show that if Δ is not regular then there is a growing variable $X \in Var(\Delta)$. As Δ is not regular, there is (due to Lemma 1) an infinite path $\mathcal{P}$ of the form $X_1 = \alpha_0 \stackrel{a_0}{\rightarrow} \alpha_1 \stackrel{a_1}{\rightarrow} \alpha_2 \stackrel{a_2}{\rightarrow} \cdots$ such that $\alpha_i \not\sim \alpha_j$ for $i \neq j$. We show that if $Var(\Delta)$ does not contain any growing variable, then there are $i \neq j$ such that $\alpha_i \sim \alpha_j$. It contradicts the assumption above—hence $Var(\Delta)$ contains at least one growing variable.

Let $IT_{\mathcal{P}}$ be the inheritance tree for the path $\mathcal{P}$. To complete the proof we need to divide $IT_{\mathcal{P}}$ into more manageable units called *blocks*.

Levels of $IT_{\mathcal{P}}$ which contain just one node are called *delimiters* of $IT_{\mathcal{P}}$. A *block* of $IT_{\mathcal{P}}$ is a subgraph S of $IT_{\mathcal{P}}$ composed of:

1. all nodes and edges between two successive delimiters i and j where $i < j$. The only node of *Level* i is called the *opening* node of S and the

only node of *Level* j is called the *closing* node of S. Out-going edges of the closing node and in-going edges of the opening node are not a part of S.

2. all nodes below the delimiter i (including *Level* i), if there is no delimiter j with $j > i$. The only node of *Level* i is called the *opening* node of S. In-going edges of the opening node are not a part of S.

As *Level* 0 is a delimiter of $IT_{\mathcal{P}}$, we can view $IT_{\mathcal{P}}$ as a vertical sequence of blocks.

The *width* of $IT_{\mathcal{P}}$ is defined to be the least $n \in N$ such that the cardinality of i^{th} *Level* of $IT_{\mathcal{P}}$ is less or equal n for each $i \in N \cup \{0\}$. If there is no such n, we define the width of $IT_{\mathcal{P}}$ to be ∞.

Similarly, if S is a block of $IT_{\mathcal{P}}$, the *width* of S is the least $n \in N$ such that the cardinality of each *Level* which is a part of S is less or equal n. If there is no such n, we define the width of S to be ∞.

Furthermore, we define the *branching degree* of $IT_{\mathcal{P}}$ to be the least $n \in N$ such that each node U of $IT_{\mathcal{P}}$ has at most n immediate successors. The branching degree of $IT_{\mathcal{P}}$ is always finite (it actually depends only on Δ—let $\mathcal{M}$ be the set of all *VPA* expressions, which appear in defining equations of Δ (see Definition 2). The branching degree of $IT_{\mathcal{P}}$ is then at most $\max\{Length(\beta) \mid \beta \in \mathcal{M}\})$. We denote the branching degree of $IT_{\mathcal{P}}$ by $\mathcal{D}$ in the rest of this proof.

Each node U of $IT_{\mathcal{P}}$ defines its associated subtree, rooted by U. This subtree is denoted $Subtree(U)$. Although the notions of block, width, branching node, tail, etc. were originally defined for $IT_{\mathcal{P}}$, they can be used also for any $Subtree(U)$ of $IT_{\mathcal{P}}$ in an obvious way.

We prove that if $Var(\Delta)$ does not contain any growing variable, then for each node U of $IT_{\mathcal{P}}$ the $Subtree(U)$ has the width at most $\mathcal{D}^{n-1}$, where $n = card(Allow(U))$.

We proceed by induction on $n = card(Allow(U))$: First, if $Var(\Delta)$ does not contain any growing variable, then $Subtree(U)$ does not contain any node U with $Allow(U) = \emptyset$. This is due to Lemma 4—clearly $Label(U) \notin \emptyset$, thus $Label(U)$ would be a growing variable. Hence n is at least 1.

1. **n=1:** Let $Allow(U) = \{X\}$. We show that $Subtree(U)$ does not contain any branching node. Let us assume the opposite. Then there is a branching node V in $Subtree(U)$ with $Allow(V) = \{X\}$, thus

$Label(V) = X$. As V is branching, at least one immediate successor V' of V has the property $Allow(V') = Allow(V) - \{Label(V)\} = \emptyset$. Hence $Label(V')$ is a growing variable and we have a contradiction. As $Subtree(U)$ does not contain any branching node, the width of $Subtree(U)$ is $1 = \mathcal{D}^{n-1}$.

2. **induction step:** Let $card(Allow(U)) = n$. We prove that each block of $Subtree(U)$ has the width at most $\mathcal{D}^{n-1}$. Let S be a block of $Subtree(U)$ and let V be its opening node. Clearly $card(Allow(V)) \leq n$. If V has no successors then the width of S is 1. If V is not branching then the only immediate succesor of V is a closing node of S, thus the width of S equals 1. If V is branching, there are two possibilities:

   - V does not have a tail. Then each immediate successor V' of V has the property $card(Allow(V')) \leq n - 1$. By induction hypothesis, the width of $Subtree(V')$ is at most $\mathcal{D}^{n-2}$. As V can have at most $\mathcal{D}$ immediate successors, the width of $Subtree(V)$ is at most $\mathcal{D}.\mathcal{D}^{n-2} = \mathcal{D}^{n-1}$. Thus the width of S is also at most $\mathcal{D}^{n-1}$.
   - V has a tail T. Each immediate successor V' of V which is different from T has the property $card(Allow(V')) \leq n - 1$. Hence we can use the induction hypothesis for each such V'. The only problem is the node T. We show, that if T has a branching successor T' then the node T' is either the closing node of the block S or it is a successor of the closing node of the block S—hence the block S can have the width at most $(\mathcal{D} - 1).\mathcal{D}^{n-2} + 1$.

     Suppose that T has a branching successor T'. Branching nodes are always active—thus T has at least one active successor. Let W be the active successor of T which has the least distance from T. The node T' is clearly either the node W (if W is branching), or a successor of W. We show, that the node W is the closing node of the block S. But it follows directly from the definition of the tail (see Remark 1)—as W is active, there are no successors of V in the level of W except the node W itself.

We have just proved that if $Var(\Delta)$ does not contain any growing variable then the width of $IT_{\mathcal{P}}$ is at most $\mathcal{D}^{card(\,Var(\Delta))-1}$. Hence each element α_i of $\mathcal{P}$ has the property $Length(\alpha_i) \leq \mathcal{D}^{card(\,Var(\Delta))-1}$. As $Var(\Delta)$ is finite, there are only finitely many $VPA(\Delta)$ expressions whose *Length* is at most

$\mathcal{D}^{card(Var(\Delta))-1}$. Therefore there are $i, j \in N \cup \{0\}$, $i \neq j$, such that $\alpha_i = \alpha_j$ and thus $\alpha_i \sim \alpha_j$. □

3.2 A construction of the process Δ' in normal form

In this section we show that if a given normed PA process Δ is regular, then Δ can be effectively transformed into a regular process Δ' in normal form such that $\Delta \sim \Delta'$. In order to simplify the construction, we identify several *VPA* expressions:

Definition 10. *Let $\equiv$ be the smallest congruence relation over VPA expressions such that the following laws hold:*

- *associativity for sequential composition (the '.' operator).*
- *associativity and commutativity for parallel composition (the '$\|$' operator).*

The algorithm is based on the following fact:

Lemma 5. *A normed PA process Δ is regular iff Δ can reach only finitely many states up to $\equiv$.*

The algorithm finds all reachable states $\alpha \in VPA(\Delta)$ of Δ up to $\equiv$. For each such α a new variable and a new defining equation is added to Δ'.

The relationship between variables of Δ' and reachable states of Δ is described by the set $MEM \subseteq Var \times VPA(\Delta)$. This set is initialised to $MEM = \{[Y_1, X_1]\}$ where X_1 is the leading variable of Δ and Y_1 is the leading variable of Δ'.

An element $[Y, \alpha]$ of *MEM* is said to be *undefined* if there is no defining equation for Y in Δ'. The algorithm chooses any undefined element of *MEM* and adds a new defining equation for Y to Δ', possibly producing new undefined elements of *MEM*. The algorithm stops when *MEM* does not contain any undefined elements.

Let $[Y, \alpha]$ be an undefined element of *MEM*. The defining equation for Y in Δ' is obtained by *unfolding* α. The function *Unfold* is defined as follows:

$$Unfold(\alpha) = \begin{cases} \sum a_{ij}\alpha_{ij} & \text{if } \alpha = X_j \text{ and } X_j \stackrel{def}{=} \sum a_{ij}\alpha_{ij} \in \Delta \\ Distr(Unfold(\beta_1), \beta_2) & \text{if } \alpha = \beta_1.\beta_2 \\ Expand1(\beta_1, \beta_2) & \text{if } \alpha = \beta_1 \| \beta_2 \\ Expand2((\beta_1, \beta_2) & \text{if } \alpha = \beta_1 \between \beta_2 \end{cases}$$

where *Expand1*, *Expand2* and *Distr* are defined as follows (functions *Expand1* and *Expand2* are instances of the CCS expansion law (see [Mil89]) and the function *Distr* is a variant of the right distributivity law (see [BW90])):

$$\begin{array}{lcl} \mathit{Expand1}(\beta_1,\beta_2) & = & \sum\{\ a(\beta_1'\|\beta_2)\ :\ \beta_1 \stackrel{a}{\to} \beta_1', a \in \mathit{Act}\} \\ & + & \sum\{\ a(\beta_1\|\beta_2')\ :\ \beta_2 \stackrel{a}{\to} \beta_2', a \in \mathit{Act}\} \\ \\ \mathit{Expand2}(\beta_1,\beta_2) & = & \sum\{\ a(\beta_1'\|\beta_2)\ :\ \beta_1 \stackrel{a}{\to} \beta_1', a \in \mathit{Act}\} \\ \\ \mathit{Distr}(\sum a_{ij}\alpha_{ij},\beta) & = & \sum a_{ij}(\alpha_{ij}.\beta) \end{array}$$

The function *Unfold* returns an expression of the form

$$\sum_{i=1}^{n} a_i\alpha_i$$

where $n \in N$, $a_i \in \mathit{Act}$ and $\alpha_i \in \mathit{VPA}(\Delta)$. Now the algorithm replaces each α_i with a single variable. There are two possibilities: if the set *MEM* contains an element $[Z,\beta]$ such that $\alpha_i \equiv \beta$, then the expression α_i is replaced with Z. Otherwise, the expression α_i is replaced with a new variable W and the pair $[W,\alpha_i]$ is added to *MEM*. After the replacement of each α_i the defining equation for Y is added to Δ'.

It is easy to see that each variable of Δ' corresponds to a reachable state of the process Δ'. Hence the algorithm has to stop (due to Lemma 5).

Example 2 Let Δ be a normed PA process given by the following set of equations:

$$\begin{array}{lcl} X & \stackrel{def}{=} & b\ +\ a(Y\|Z).X \\ Y & \stackrel{def}{=} & c\ +\ a(Z\|(Z.Z)) \\ Z & \stackrel{def}{=} & c \end{array}$$

The process Δ' is constructed in the following way (the first two elements of each line constitute a member of *MEM*, the third element is a result of

Unfold and the last element is the defining equation):

$$\begin{array}{lcll}
A & = & X & = & b \;+\; a(Y\|Z).X & = & b \;+\; aB \\
B & = & (Y\|Z).X & = & a(Z\|(Z.Z)\|Z).X \;+\; c(Z.X) \;+\; c(Y.X) & = & aC \;+\; cD \\
 & & & & & & \;+\; cE \\
C & = & (Z\|(Z.Z)\|Z).X & = & c((Z\|Z\|Z).X) \;+\; c((Z\|(Z.Z)).X) & = & cF \;+\; cG \\
D & = & Z.X & = & cX & = & cA \\
E & = & Y.X & = & cX \;+\; a((Z\|(Z.Z)).X) & = & cA \;+\; aG \\
F & = & (Z\|Z\|Z).X & = & c((Z\|Z).X) & = & cH \\
G & = & (Z\|(Z.Z)).X & = & c(Z.Z.X) \;+\; c((Z\|Z).X) & = & cI \;+\; cH \\
H & = & (Z\|Z).X & = & c(Z.X) & = & cD \\
I & = & (Z.Z.X) & = & c(Z.X) & = & cD
\end{array}$$

Using this algorithm it is possible to decide bisimilarity for any pair of processes $[\Delta_1, \Delta_2]$, where Δ_1 is a normed PA process and Δ_2 is a regular process. First, we check whether Δ_1 is regular. If not, then $\Delta_1 \not\sim \Delta_2$. Otherwise, we construct the regular process Δ_1' in normal form such that $\Delta_1 \sim \Delta_1'$ and check whether $\Delta_1' \sim \Delta_2$.

Theorem 2. *Bisimulation equivalence is decidable for any pair of processes such that one process of this pair is a normed PA process and the other process is regular.*

4 Conclusions

We proved that regularity of normed PA processes is decidable in polynomial time. As our result is constructive, we obtained also decidability of bisimulation equivalence for any pair of processes such that one process of this pair is a normed PA process and the other process is regular.

A natural question is whether it is possible to replace the pure merge operator ('$\|$') with another form of parallel composition without the loss of decidability of regularity. It can be easily shown that presented results are still valid if we replace the merge operator with the full parallel operator of CCS (which allows synchronisations on complementary actions). However, if we use e.g. the operator '$\|_A$' of CSP (which can *force* synchronisations), regularity becomes undecidable—see [Kuč95] for details.

An interesting open problem is whether our result can be extended to the class of all (not necessarily normed) PA processes. Another related open problem is the decidability of bisimulation equivalence in the class of (normed) PA processes.

5 Acknowledgement

I would like to thank Ivana Černá and Mojmír Křetínský for reading the first draft of this paper. Their comments made this article much more readable.

References

[BBK87] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings of PARLE 87*, volume 259 of *LNCS*, pages 93–114. Springer-Verlag, 1987.

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In *Proceedings of CONCUR 96*, volume 1119 of *LNCS*, pages 247–262. Springer-Verlag, 1996.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proceedings of POPL 95*, pages 95–106. ACM Press, 1995.

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

[Cau88] D. Caucal. Graphes canoniques de graphes algebriques. Rapport de Recherche 872, INRIA, 1988.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation is decidable for all basic parallel processes. In *Proceedings of CONCUR 93*, volume 715 of *LNCS*, pages 143–157. Springer-Verlag, 1993.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In *Proceedings of CONCUR 92*, volume 630 of *LNCS*, pages 138–147. Springer-Verlag, 1992.

[EJ96] J. Esparza and P. Jančar. Deciding finiteness of Petri nets up to bisimilarity. In *Proceedings of ICALP 96*, volume 1099 of *LNCS*, pages 478–489. Springer-Verlag, 1996.

[Gro91] J. F. Groote. A short proof of the decidability of bisimulation for normed BPA processes. *Information Processing Letters*, 42:167–171, 1991.

[HS91] H. Hüttel and C. Stirling. Actions speak louder than words: Proving bisimilarity for context-free processes. In *Proceedings of LICS 91*, pages 376–386. IEEE Computer Society Press, 1991.

[Jan94] P. Jančar. Decidability questions for bisimilarity of Petri nets and some related problems. In *Proceedings of STACS 94*, volume 775 of *LNCS*, pages 581–592. Springer-Verlag, 1994.

[JM95] P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proceedings of CONCUR 95*, volume 962 of *LNCS*, pages 348–362. Springer-Verlag, 1995.

[Kuč95] A. Kučera. Deciding regularity in process algebras. BRICS Report Series RS-95-52, Department of Computer Science, University of Aarhus, October 1995.

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

[MM94] S. Mauw and H. Mulder. Regularity of BPA-systems is decidable. In *Proceedings of CONCUR 94*, volume 836 of *LNCS*, pages 34–47. Springer-Verlag, 1994.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings* 5^{th} *GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer-Verlag, 1981.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

**Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic**