Basic Research in Computer Science

BRICS RS-95-52

A. Kucera: Deciding Regularity in Process Algebras

Deciding Regularity in Process Algebras

Anton´n Kucera

BRICS Report Series **RS-95-52**

ISSN 0909-0878 **October 1995**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Deciding Regularity in Process Algebras

Antonín Kučera*

email: tony@informatics.muni.cz

Abstract

We consider the problem of deciding regularity of normed BPP and normed BPA processes. A process is regular if it is bisimilar to a process with finitely many states. We show, that regularity of normed BPP processes is decidable and we provide a constructive regularity test. We also show, that the same result can be obtained for the class of normed BPA processes.

Regularity can be defined also w.r.t. other behavioural equivalences. We define notions of strong regularity and finite characterisation and we examine their relationship with notions of regularity and finite representation. The introduced notion of the finite characterisation is especially interesting from the point of view of possible verification of concurrent systems.

In the last section we present some negative results. If we extend the BPP algebra with the operator of restriction, regularity becomes undecidable and similar results can be obtained also for other process algebras.

*Presented results were obtained during the author's stay at BRICS (Basic Research in Computer Science), Department of Computer Science, University of Aarhus. The stay was supported by The Danish Ministry of Education.

Contents

1 Introduction

One of the most popular models for concurrency are process algebras like CCS, CSP or ACP. Various properties of these models have beed studied in the last decades. This paper belongs to the bunch which could be labeled "decidability results". The dominating subject in this area is the problem of deciding various behavioural equivalences in certain subclasses of mentioned algebras.

Milner in [1] has shown that bisimulation equivalence is decidable in the class of regular (finite-state) process. In [3] Baeten, Bergstra and Klop proved that bisimulation equivalence is decidable in the class of normed BPA processes. It was the first result, showing that bisimulation equivalence can remain decidable in a class of processes, in which the language equivalence is undecidable. Much simpler proof of this was later given by Caucal [4] and Groote [5]. In [6] Hüttel and Stirling used a tableau decision method and gave also sound and complete equational theory for the class of normed BPA processes.

This result was later extended to the whole class of BPA processes by Christensen, Hüttel and Stirling [7]. Another class of processes, BPP, is examined in [8]. Christensen, Hirsfeld and Moller proved that bisimulation equivalence is decidable in this class, using a tableau technique similar to [6].

An open problem was the question whether it is decidable if a given process is regular (i.e. it is bisimilar to a process with finitely many states). This natural problem is generally undecidable (see [11]), but Mauw and Mulder showed in [2], that regularity is decidable in the class of BPA systems.

In this paper we prove that regularity is decidable in the class of normed BPP processes. Moreover, if the tested process is regular then our algorithm outputs also the normal form of this regular process. We also show, that the result of [2] can serve as a constructive regularity test for the class of normed BPA processes.

The notion of regularity can be defined also w.r.t. other behavioural equivalences. Regular processes have finite representations (see [11]), but a finite representation of a process Δ generally does not express the behaviour of reachable states of Δ. We introduce the notion of strong regularity, which in many cases guarantees an existence of a finite characterisation, which describes a process as a whole. We also study the relationship between finite representations and finite characterisations.

In the last section we present some negative results, stating that regu-

larity and strong regularity are undecidable in some process classes. First we consider a calculus, obtained by extending BPP with the rectriction operator and we prove that regularity and strong regularity are undecidable. This result is obtained via a simple reduction, which can be applied also to other process algebras.

2 Basic definitions

2.1 Subclasses of CCS - BPA, BPP, BPP$_\tau$

Let $Act = \Lambda \cup \overline{\Lambda} \cup \{\tau\}$ be a set of *atomic actions*, where $\Lambda = \{a, b, c, \ldots\}$ is a countably infinite set of *labels*, $\overline{\Lambda} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ is a countably infinite set of *co-labels* with the convention $\overline{\overline{a}} = a$ and τ is a distinguished element which does not belong to $\Lambda \cup \overline{\Lambda}$. Let Var be a countably infinite set of *variables*, $Var = \{X, Y, Z, \ldots\}$. The classes of recursive BPA, BPP and BPP$_\tau$ expressions are defined by the following abstract syntax equations:

$$\begin{array}{lcl} E_{BPA} & ::= & a \mid X \mid E_{BPA}.E_{BPA} \mid E_{BPA} + E_{BPA} \\ E_{BPP} & ::= & a \mid X \mid a.E_{BPP} \mid E_{BPP} \| E_{BPP} \mid E_{BPP} + E_{BPP} \\ E_{BPP_\tau} & ::= & a \mid X \mid a.E_{BPP_\tau} \mid E_{BPP_\tau} | E_{BPP_\tau} \mid E_{BPP_\tau} + E_{BPP_\tau} \end{array}$$

Here a ranges over Act and X ranges over Var. We also let greek letters $\alpha, \beta, \ldots$ to range over process expressions. The symbol Act^* denotes the set of all finite strings over Act and the symbol Act^+ denotes the set of all nonempty finite strings over Act. The parallel operator "$\|$" of BPP is sometimes called *the merge operator*, and the operator "$|$" of BPP$_\tau$ is called *the full parallel operator* because it allows synchronizations

As usual, we restrict our attention to guarded expressions. A process expression (BPA, BPP or BPP$_\tau$) is *guarded* iff every variable occurence is within the scope of an atomic action.

A *guarded process* (BPA, BPP or BPP$_\tau$) 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, and the E_i are guarded expressions (BPA, BPP or BPP$_\tau$), containing the variables from $\{X_1, \ldots, X_n\}$. The set of variables, which appear in Δ, is denoted by $Var(\Delta)$.

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 (ϵ denotes empty expression):

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

$$\frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'} \qquad \frac{E \xrightarrow{a} E'}{E|F \xrightarrow{a} E'|F} \qquad \frac{F \xrightarrow{a} F'}{E|F \xrightarrow{a} E|F'}$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{\overline{a}} F'}{E|F \xrightarrow{\tau} E'|F'} \qquad \frac{E \xrightarrow{a} E'}{E\|F \xrightarrow{a} E'\|F} \qquad \frac{F \xrightarrow{a} F'}{E\|F \xrightarrow{a} E\|F'}$$

$$\frac{E \xrightarrow{a} E'}{X \xrightarrow{a} E'} \; (X \stackrel{def}{=} E \in \Delta)$$

Nodes of the transition system generated by Δ are process expressions, which are often called *states of* Δ, or just "states" when Δ is understood from the context. The transitive closure of "$\rightarrow$" is denoted by "$\rightarrow_+$", the reflexive and transitive closure by "$\rightarrow_*$". Given two states α, β, we say that β is *reachable from* α, if $\alpha \xrightarrow{w}_* \beta$ for some $w \in Act^*$. States of Δ, which are reachable from X_1 are said to be *reachable*.

Remark 1 *Processes are often identified with their leading variables. Furthermore, if we assume a fixed process Δ, we can view any variable $Y \in Var(\Delta)$ as a process too; we simply change the order of defining equations so that Y becomes the leading variable of the process. Similarly, any guarded expression α, containing variables from $Var(\Delta)$ denotes a process - we simply define a new system Δ', which is identical to Δ with one exception - it has a new leading variable N and the equation $N \stackrel{def}{=} \alpha$. All notions originally defined for processes can be used for variables and process expressions in this sense too.*

Remark 2 *Guarded processes generate finitely branching transition graphs, that is, the set $\{\beta \mid \alpha \xrightarrow{a} \beta\}$ is finite for each state α. It is easy to show, that it would not be true if we allowed unguarded expressions.*

2.2 Normed processes

An important subclass of processes can be obtained by an extra restriction of *normedness*. A variable $X \in Var(\Delta)$ is *normed* iff there is $w \in Act^*$ such that $X \xrightarrow{w} \epsilon$. In that case we define the *norm* of X, $[X]$, to be the length of the shortest such w, counting the length of "τ" as two; all other actions have the length one. Thus $[X] = \min\{length(w) \mid X \xrightarrow{w} \epsilon\}$. A process Δ is *normed*, if all variables of $Var(\Delta)$ are normed. The norm of the process is then defined to be the norm of X_1.

As normed processes are intensively studied in this paper, we emphasize some properties of norm:

- The norm of a normed process is easy to compute: $[a] = 1$, $[\alpha + \beta] = \min\{[\alpha], [\beta]\}$, $[\alpha.\beta] = [\alpha] + [\beta]$, $[\alpha \| \beta] = [\alpha] + [\beta]$, $[\alpha|\beta] = [\alpha] + [\beta]$, and if $X_i \stackrel{def}{=} E_i$ and $[E_i] = n$, then $[X_i] = n$.
- For each normed BPA, BPP or BPP$_\tau$ process α there are $a \in Act$ and α', such that $\alpha \stackrel{a}{\rightarrow} \alpha'$ and $[\alpha] = [\alpha'] + 1$
- Bisimilar processes (see section 2.3) must have the same norm.

2.3 Bisimulation

The equivalence between process expressions (states) we are here interested in is *bisimilarity* [1], defined as follows:

Definition 1 *A binary relation R over process expressions is a* bisimulation *if whenever $\alpha R \beta$ then for each $a \in Act$*

- *if $\alpha \stackrel{a}{\rightarrow} \alpha'$, then $\beta \stackrel{a}{\rightarrow} \beta'$ for some β' such that $\alpha' R \beta'$*
- *if $\beta \stackrel{a}{\rightarrow} \beta'$, then $\alpha \stackrel{a}{\rightarrow} \alpha'$ for some α' such that $\alpha' R \beta'$*

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

2.4 Greibach normal form

Any BPA process Δ can be effectively presented in so-called 3-Greibach normal form (see [3]):

Definition 2 *A BPA process Δ is said to be in* Greibach normal form (GNF) *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 $\alpha_{ij} \in Var(\Delta)^*$. If $length(\alpha_{ij}) \leq 2$, then Δ is said to be in 3-GNF. ($Var(\Delta)^*$ denotes the set of all finite sequences of variables from $Var(\Delta)$).

A similar result holds for BPP, resp. BPP$_\tau$ (see [10]). Any BPP resp. BPP$_\tau$ process Δ' can be effectively represented in the normal form, which is very similar to the 3-GNF of BPA - hence it is called 3-GNF too. Before its presentation we need to introduce the set $Var(\Delta')^{\otimes}$ of all finite multisets over $Var(\Delta')$. Each multiset of $Var(\Delta')^{\otimes}$ denotes a BPP resp. BPP$_\tau$ expression by combining its elements in parallel using the merge, resp. the full parallel operator.

Definition 3 *A BPP, resp. BPP$_\tau$ process Δ' is said to be in* Greibach normal form (GNF) *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 $\alpha_{ij} \in Var(\Delta')^{\otimes}$. If $card(\alpha_{ij}) \leq 2$, then Δ' is said to be in 3-GNF. ($card(\alpha_{ij})$ denotes the cardinality of α_{ij}).

From now on we assume, that all BPA, BPP and BPP$_\tau$ processes we are working with are presented in 3-GNF. This justifies also the assumption, that all reachable states of a BPA process Δ are elements of $Var(\Delta)^*$ and all reachable states of a BPP or BPP$_\tau$ process Δ' are elements of $Var(\Delta')^{\otimes}$. Furthermore, all variables of a normed BPA, BPP or BPP$_\tau$ process in 3-GNF have norm at least one. Occasionally we will also use the notation α^i, where α is a state of some BPA, BPP or BPP$_\tau$ process, $i \in N$. The notation has the following meaning:

$$\alpha^i = \underbrace{\alpha.\alpha.\ldots.\alpha}_{i} \qquad \text{if } \alpha \text{ is a state of some BPA process}$$

$$\alpha^i = \underbrace{\alpha\|\alpha\|\ldots\|\alpha}_{i} \qquad \text{if } \alpha \text{ is a state of some BPP process}$$

$$\alpha^i = \underbrace{\alpha|\alpha|\ldots|\alpha}_{i} \qquad \text{if } \alpha \text{ is a state of some BPP}_\tau \text{ process}$$

2.5 Regularity of processes

The main question considered in this paper is, whether the behaviour of a given process is regular, i.e. whether it is bisimilar to a process with finitely many states.

Definition 4 *A process Δ is* regular *if there is a process Δ' with finitely many states, such that $\Delta \sim \Delta'$.*

It is easy to show that a process is regular iff it can reach only finitely many states up to bisimilarity. In [1] it is shown, that regular processes can be represented in the following normal form:

Definition 5 *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 $\Delta \sim \Delta'$.

3 The constructive test of regularity for normed BPP processes

3.1 The inheritence tree

Let Δ be a BPP process in 3-GNF, given by the set of equations

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

where $1 \leq i \leq n$. With each $\alpha_{i,j}$ we associate some linear ordering $\preceq_{i,j}$ on $\alpha_{i,j}$. We define a partial function $Lbl : N \times N \times N \rightarrow Var(\Delta)$:

$$Lbl(i,j,k) = \begin{cases} k^{th} \text{ element of } \alpha_{i,j} \text{ w.r.t. } \preceq_{i,j} & \text{if } 1 \leq i \leq n \ \wedge \ 1 \leq j \leq n_i \\ & \wedge \ k \leq card(\alpha_{i,j}) \\ \bot & \text{otherwise} \end{cases}$$

Now let $X_1 = \beta_0 \stackrel{a_0}{\rightarrow} \beta_1 \stackrel{a_1}{\rightarrow} \beta_2 \stackrel{a_2}{\rightarrow} \beta_3 \stackrel{a_3}{\rightarrow} \ldots$ be an infinite path. Each transition $\beta_i \stackrel{a_i}{\rightarrow} \beta_{i+1}$ is due to some variable $X_r \in \beta_i$, which emits the action a_i and enters the state $\alpha_{r,s}$, where $a_i \alpha_{r,s}$ is a summand in the defining equation of X_r in Δ. We say, that X_r is *the active variable of* β_i and $\alpha_{r,s}$ is *the step of* β_i. Generally, X_{r_i} denotes the active variable of β_i and α_{r_i,s_i} denotes the step of β_i for each $i \in N \cup \{0\}$. To be able to examine properties of

such an infinite path, we define the associated *inheritence tree*. Nodes of the inheritence tree are formed by a subset of $(N \cup \{0\}) \times (N \cup \{0\})$:

$$Nodes = \bigcup_{i=0}^{\infty} \{[i,j] \mid 0 \leq j < card(\beta_i)\}$$

Furthermore, we define the function $Label : Nodes \rightarrow Var(\Delta)$:

$$Label([0,0]) = X_1$$

$$Label([i+1,j]) =$$

$$= \begin{cases} Label([i,j]) & \text{if } 0 \leq j < k_i \\ Lbl(r_i, s_i, j - k_i + 1) & \text{if } k_i \leq j < k_i + card(\alpha_{r_i,s_i}) \\ Label([i, j - card(\alpha_{r_i,s_i}) + 1]) & \text{if } k_i + card(\alpha_{r_i,s_i}) \leq j < card(\beta_{i+1}) \end{cases}$$

where $k_i = \max\{j \in N \cup \{0\} \mid 0 \leq j < card(\beta_i) \ \wedge \ Label[i,j] = X_{r_i}\}$

The way how k_i is chosen is not crucial in fact. There is no need to distinguish between multiple occurence of the same variable within a single state, but we want to keep our construction of the inheritence tree deterministic. Finally, we define the binary relation $Edges$ on $Nodes$:

$[i,j]\ Edges\ [l,m] \stackrel{def}{\iff} l = i+1$ and one of the following conditions holds:

1. $0 \leq j < k_i \quad \wedge \quad j = m$
2. $j = k_i \quad \wedge \quad k_i \leq m < k_i + card(\alpha_{r_i,s_i})$
3. $k_i < j < card(\beta_i) \quad \wedge \quad m = j + card(\alpha_{r_i,s_i}) - 1$

The inheritence tree associated with the path $X_1 = \beta_0 \stackrel{a_0}{\rightarrow} \beta_1 \stackrel{a_1}{\rightarrow} \beta_2 \stackrel{a_2}{\rightarrow} \beta_3 \stackrel{a_3}{\rightarrow} \ldots$ is a triple $[Nodes,\ Edges,\ [0,0]]$. We also need some further notions:

Definition 6 *Let Δ be a normed BPP process, $X_1 = \beta_0 \stackrel{a_0}{\rightarrow} \beta_1 \stackrel{a_1}{\rightarrow} \beta_2 \stackrel{a_2}{\rightarrow} \beta_3 \stackrel{a_3}{\rightarrow} \ldots$ be an infinite path and let IT be the inheritence tree associated with this path.*

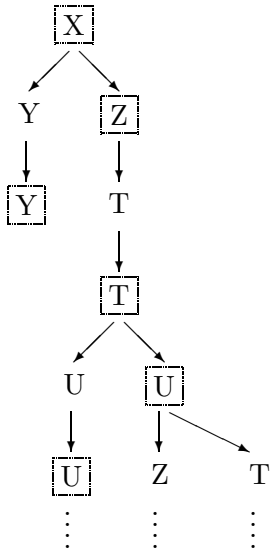
A node $[i,j]$ is a son *of a node $[k,l]$ iff $[[k,l],[i,j]] \in Edges$. Moreover, if $[i,j]$ is a son of $[k,l]$, then $[k,l]$ is said to be* the father *of $[i,j]$. Note that each node may have 0,1 or 2 sons, but each node except $[0,0]$ has exactly one father.*

Similarly, $[i,j]$ is a descendant *of $[k,l]$ iff there exists a path from $[k,l]$ to $[i,j]$ in IT. Finally, a node $[i,j]$ is* branching *iff it has two different sons.*

Example 1 Let Δ be a process given by the following set of equations:

$$\begin{aligned} X &\stackrel{def}{=} a(Y \| Z) \\ Y &\stackrel{def}{=} b \ + \ c(T \| U) \\ Z &\stackrel{def}{=} b \ + \ cT \ + \ a(Y \| T) \\ T &\stackrel{def}{=} a \ + \ a(U \| U) \\ U &\stackrel{def}{=} b(Z \| T) \ + \ a(X \| Z) \end{aligned}$$

Process Δ is normed and its leading variable is X. The linear ordering $\preceq_{i,j}$ is determined by the position of variables in $\alpha_{i,j}$, e.g. $\preceq_{0,0}= \{[Y,Y],[Z,Z],[Y,Z]\}$, that is $Y \preceq_{0,0} Z$. Let $X \stackrel{a}{\rightarrow} Y \| Z \stackrel{c}{\rightarrow} Y \| T \stackrel{b}{\rightarrow} T \stackrel{a}{\rightarrow} U \| U \stackrel{b}{\rightarrow} U \| Z \| T \ldots$ be an infinite path. If we draw a fragment of the associated inheritence tree and replace each $[i,j]$ with $Label([i,j])$, we get the following picture (active variables are placed in a box):



3.2 Decidability of regularity for normed BPP processes

Definition 7 *Let Δ be a normed BPP process. We define the binary relation "$\rightsquigarrow$" on $Var(\Delta)$:*

$$X \rightsquigarrow Y \stackrel{def}{\Longleftrightarrow} \exists\, w \in Act^+, \gamma \in Var(\Delta)^{\otimes},\ \textit{such that}\ X \stackrel{w}{\rightarrow}_+ Y \| \gamma \ \wedge\ \gamma \neq \emptyset$$

A variable $Y \in Var(\Delta)$ is said to be growing *iff $Y \rightsquigarrow Y$*

We show, that "$\rightsquigarrow$" is the least fixed-point of a function, which represents an iterative procedure. We also show, that this fixed-point must be reached after a finite number of iterations. We need several definitions:

Definition 8 *Let Δ be a normed BPP process. We define the binary relation "$\hookrightarrow$" on $Var(\Delta)$:*

$X \hookrightarrow Y \stackrel{def}{\Longleftrightarrow}$ *there is a summand $a_{ij}\alpha_{ij}$ in the defining equation of X, such that $Y \in \alpha_{ij}$.*

The symbol "$\hookrightarrow_*$" denotes the reflexive and transitive closure of "$\hookrightarrow$".

It is a standard result that for any binary relation R on a finite set, the reflexive and transitive closure of R can be effectively constructed. Hence there is an algorithm, which constructs "$\hookrightarrow_*$".

Definition 9 *Let Δ be a normed BPP process. We define the function $\mathcal{F}$ over subsets of $Var(\Delta) \times Var(\Delta)$ as follows. If $\mathcal{M} \subseteq Var(\Delta) \times Var(\Delta)$, then $[X, Y] \in \mathcal{F}(\mathcal{M}) \stackrel{def}{\Longleftrightarrow}$ one of the following conditions holds:*

- $[X, Y] \in \mathcal{M}$
- $\exists\, Z \in Var(\Delta),\ \textit{such that}\ [X, Z] \in \mathcal{M}\ \wedge\ Z \hookrightarrow_* Y$
- $\exists\, Z \in Var(\Delta),\ \textit{such that}\ [Z, Y] \in \mathcal{M}\ \wedge\ X \hookrightarrow_* Z$

Definition 10 *Let Δ be a normed BPP process. For each $i \in N \cup \{0\}$ we define the binary relation "$\rightsquigarrow_i$" over $Var(\Delta)$:*

$$\begin{aligned} \rightsquigarrow_0 &= \{[X, Y] \mid \textit{there is a summand } a_{ij}\alpha_{ij} \textit{ in the defining equation of } X, \textit{ such that } Y \in \alpha_{ij} \ \wedge\ card(\alpha_{ij}) = 2\} \\ \rightsquigarrow_{i+1} &= \mathcal{F}(\rightsquigarrow_i) \end{aligned}$$

Now it is easy to see, that the relation "$\rightsquigarrow$" is exactly the least fixed-point of $\mathcal{F}$, which belongs to the set $\{\rightsquigarrow_i \mid i \in N \cup \{0\}\}$:

$$\rightsquigarrow = \rightsquigarrow_k, \quad \text{where } k = \min\{i \in N \cup \{0\} \mid \rightsquigarrow_i = \mathcal{F}(\rightsquigarrow_i)\}$$

As $Var(\Delta) \times Var(\Delta)$ is finite and $\rightsquigarrow_i \subseteq \rightsquigarrow_{i+1}$ for each $i \in N \cup \{0\}$, the least fixed-point must be reached in a finite number of steps.

Definition 11 *Let Δ be a normed BPP process. A variable $Y \in Var(\Delta)$ is said to be accessible iff there is a reachable state $\alpha \in Var(\Delta)^{\otimes}$, such that $Y \in \alpha$.*

Again, it is easy to define a function, such that the set of all accessible variables of $Var(\Delta)$ is the least fixed-point of this function. So we have the following lemma:

Lemma 1 *Let Δ be a normed BPP process. It is decidable, whether the set $Var(\Delta)$ contains an accessible growing variable.*

Lemma 2 *Let Δ be a normed BPP process and let $\alpha, \beta \in Var(\Delta)^{\otimes}$ be its states. If β is reachable from α, then for each variable $Y \in \beta$ the state $\{Y\}$ is reachable from α.*

Proof: Since β is reachable from α, there is $v \in Act^*$, such that $\alpha \stackrel{v}{\rightarrow} \beta$. Suppose $\beta = Y\|A_1\| \ldots \|A_n$. As Δ is normed, each of the variables $A_1, \ldots, A_n$ is normed. Hence for each i, $1 \leq i \leq n$, there is $w_i \in Act^+$, such that $A_i \stackrel{w_i}{\rightarrow}_+ \epsilon$. Let w denotes the concatenation of w_i, $1 \leq i \leq n$. The state $\{Y\}$ is reachable from α, because $\alpha \stackrel{v}{\rightarrow}_* Y\|A_1\| \ldots \|A_n \stackrel{w}{\rightarrow}_+ Y$.

□

We have emphasized this trivial property of normed BPP processes, because this becomes a crucial point when we start to think about possible extension of the presented result to the whole class of BPP processes.

Lemma 3 *A process Δ is not regular iff there is an infinite path, $X_1 = \beta_0 \stackrel{a_0}{\rightarrow} \beta_1 \stackrel{a_1}{\rightarrow} \beta_2 \stackrel{a_2}{\rightarrow} \beta_3 \stackrel{a_3}{\rightarrow} \ldots$, such that $\beta_i \not\sim \beta_j$ for $i \neq j$.*

Proof:
"$\Leftarrow$:" Obvious - Δ can reach infinitely many pairwise non-bisimilar states.
"$\Rightarrow$:" Let $TS(\Delta)$ be a transition system generated by Δ. If we identify

bisimilar nodes in $TS(\Delta)$, we get a transition system $TS(\Delta)/\sim$, which is bisimilar to $TS(\Delta)$ and all its states are pairwise non-bisimilar. The set of reachable states of $TS(\Delta)/\sim$ is infinite and $TS(\Delta)/\sim$ is finitely branching (see Remark 2). Hence due to König lemma there must be an infinite path $\widetilde{X_1} \stackrel{a_0}{\to} \widetilde{\beta_1} \stackrel{a_1}{\to} \widetilde{\beta_2} \stackrel{a_2}{\to} \widetilde{\beta_3} \stackrel{a_3}{\to} \ldots$, where $\widetilde{\beta_i}$ denotes the equivalence class containing β_i. Using this path (whose elements are pairwise non-bisimilar), we can construct the required path in $TS(\Delta)$ - just by taking a suitable representative of $\widetilde{\beta_i}$.

□

Lemma 4 *Let Δ be a normed BPP process, $X_1 = \beta_0 \stackrel{a_0}{\to} \beta_1 \stackrel{a_1}{\to} \beta_2 \stackrel{a_2}{\to} \beta_3 \stackrel{a_3}{\to} \ldots$ be an infinite path, such that $\beta_i \not\sim \beta_j$ for $i \neq j$. Then there is a path in the corresponding inheritence tree, which contains infinitely many branching nodes.*

Proof: First we show that the inheritence tree contains infinitely many branching nodes. It suffices to prove that for any $i \in N$ there exists a branching node $[j,p]$, such that $j > i$. Assume the opposite - then there is $k \in N$, such that $[j,p]$ is branching $\Rightarrow \;\; j < k$. But then there is no way how the number of variables in $\beta_q,\;\; q > k$, could increase - each $\beta_q,\;\; q > k$ contains at most $card(\beta_k)$ variables. As Δ has only finitely many variables, the set of all multisets over $Var(\Delta)$, whose cardinality is at most $card(\beta_k)$, is finite. Therefore there must be $r \neq s$ such that $\beta_r = \beta_s$; hence $\beta_r \sim \beta_s$, so we have a contradiction.

It remains to find a path containing infinitely many branching nodes. To do this, we first construct *the branching tree:* Nodes of the branching tree are branching nodes of the inheritence tree. Edges are determined as follows: $[[i,j],[k,l]]$ is an egde in the branching tree iff there is a path from $[i,j]$ to $[k,l]$ in the inheritence tree, which does not contain any branching nodes except $[i,j]$ and $[k,l]$. The root of the branching tree is the branching node $[i,j]$, such that all other branching nodes are its descendants (there is just one node of this property). We have already proved that the branching tree is infinite. As each node which is not leaf has exactly two successors, we can use König lemma and conclude that the branching tree contains an infinite path. This path corresponds to an infinite path in the inheritence tree, which contains infinitely many branching nodes (realize that each edge of the branching tree represents a finite path in the inheritence tree).

□

Now we are ready to prove the main result of this section:

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

Proof:
"$\Rightarrow$:" Assume there is an accessible growing variable $Y \in Var(\Delta)$. Then $X_1 \xrightarrow{w}_* Y$ for some $w \in Act^*$ and $Y \xrightarrow{v}_+ Y\|\gamma$, where $v \in Act^+$, $\gamma \in Var(\Delta)^\oplus$ and $\gamma \neq \emptyset$. But then each state of the form $Y\|\gamma^i,\ i \in N$ is reachable. Furthemore, $Y\|\gamma^i \not\sim Y\|\gamma^j$ for $i \neq j$, as these two states have different norms. Hence Δ is not regular, as it can reach infinitely many pairwise non-bisimilar states.
"$\Leftarrow$:" Assume Δ is not regular. We show, that then $Var(\Delta)$ contains an accessible growing variable. As Δ is not regular, there is an infinite sequence $X_1 = \beta_0 \stackrel{a_0}{\rightarrow} \beta_1 \stackrel{a_1}{\rightarrow} \beta_2 \stackrel{a_2}{\rightarrow} \beta_3 \stackrel{a_3}{\rightarrow} \ldots$, whose elements are pairwise non-bisimilar. Now we examine the corresponding inheritence tree. With each node $[i,j]$ we associate the set $Seek([i,j]) \subseteq Var(\Delta)$ in the following way:

$$Seek([0,0]) = \emptyset$$

$$Seek([i+1,k]) = \begin{cases} Seek(Father([i+1,k])) & \text{if } Father([i+1,k]) \\ & \text{is not branching.} \\ Seek(Father([i+1,k])) & \text{if } Father([i+1,k]) \\ \cup\{Label(Father([i+1,k]))\} & \text{is branching.} \end{cases}$$

$Father([i+1,k])$ denotes the father of $[i+1,k]$. $Seek([i,j])$ contains in fact accessible variables, which are potential candidates to be growing - if we find a node $[i,j]$, such that $Label([i,j]) \in Seek([i,j])$, we can conclude that $Label([i,j])$ is an accessible growing variable. This is due to the following consideration: As $Label([i,j]) \in Seek([i,j])$, there is a branching node $[p,k],\ p < i$, such that $Label([p,k]) = Label([i,j])$ and $[i,j]$ is a descendant of $[p,k]$. Since $[p,k]$ is branching, it has two sons $[p+1,l],\ [p+1,l+1]$, thus

$$Label([p,k]) \stackrel{a_p}{\rightarrow} Label([p+1,l]) \parallel Label([p+1,l+1]),$$

where $a_p \in Act$. As $[i,j]$ is a descendant of $[p,k]$, one of these four possibilities holds:

1. $[p+1,l] = [i,j]$
2. $[p+1,l+1] = [i,j]$

3. $[i,j]$ is a descendant of $[p+1,l]$

4. $[i,j]$ is a descendant of $[p+1,l+1]$

In the first two cases we immediately get that $Label([i,j])$ ($= Label([p,k])$) is an accessible growing variable. If $[i,j]$ is a descendant of $[p+1,l]$, then due to Lemma 1:

$$Label([p+1,l]) \stackrel{w}{\rightarrow}_{+} Label([i,j])$$

for some $w \in Act^{+}$, hence

$$Label([p,k]) \stackrel{a_p}{\rightarrow} Label([p+1,l]) \parallel Label([p+1,l+1]) \stackrel{w}{\rightarrow}_{+} Label([i,j]) \parallel Label([p+1,l+1])$$

so the variable $Label([p,k])$ is again accessible and growing. Possibility 4 is handled in a similar way.

Due to Lemma 4 we know, that our inheritence tree contains a path with infinitely many branching nodes. Going down this path, labels of branching nodes are successively added to the $Seek$ set. As $Var(\Delta)$ is finite, we must find a branching node $[i,j]$, such that $Label([i,j]) \in Seek([i,j])$, thus $Label([i,j])$ is the desired accessible growing variable.

□

3.3 The constructive algorithm

Each regular process can be represented in normal form (see Section 2). In this section we provide an algorithm, which inputs a normed BPP process Δ in 3-GNF and outputs YES iff Δ is regular and NO otherwise. In the first case our algorithm also constructs a regular process Δ' in normal form, such that $\Delta \sim \Delta'$. During the construction of Δ' we take advantage of the fact, that bisimilarity is known to be decidable in the class of normed BPP processes (see [8]).

The algorithm first checks, whether $Var(\Delta)$ contains any accessible growing variable. If so, it outputs NO and terminates. Otherwise it initiates Δ' to be Δ and starts to remove the multisets $\alpha_{i,j}$, $card(\alpha_{i,j}) > 1$ from the defining equations of Δ'. Note this is the only thing which has to be done to obtain normal form of Δ'.

Each such $\alpha_{i,j}$ is first compared with elements of $Var(\Delta')$. If we find a variable $Y \in Var(\Delta')$, such that $Y \sim \alpha_{i,j}$, we simply replace $\alpha_{i,j}$ with Y.

Otherwise we introduce a new variable $P \in Var,\ P \notin Var(\Delta')$ and a new equation

$$P \stackrel{def}{=} \alpha_{i,j}$$

But this equation cannot be immediately added to Δ', as it is not of the form, which is prescribed by GNF ($\alpha_{i,j}$ is even not guarded). To obtain the desired form of this equation, we have to apply the CCS expansion law (see [1]):

$$A_1 \| \ldots \| A_n \ = \ \sum \{\ a(A_1 \| \ldots \| \alpha_i \| \ldots \| A_n) \ : \ A_i \stackrel{a}{\rightarrow} \alpha_i,\ a \in Act\}$$

After the application of the expansion law we get an equation, which is of the form of GNF (not necessarly 3-GNF), thus it can be added to Δ'.

We go on in this fashion, until all multisets $\alpha_{i,j},\ card(\alpha_{i,j}) > 1$ are removed. The construction must terminate, because otherwise the process Δ' could reach infinitely many pairwise non-bisimilar states (realize that newly added variables are reachable and pairwise non-bisimilar states of Δ'). As Δ' remains bisimilar to Δ after the processing of each $\alpha_{i,j}$, it contradicts the regularity of Δ.

We also describe this algorithm formally, using a Pascal-like pseudocode. The form is very simple in order to keep the description as short as possible:

Algorithm:	The constructive regularity test for normed BPP processes
Input:	A normed BPP process Δ in 3-GNF
Output:	YES and a regular process Δ' in normal form, such that $\Delta \sim \Delta'$ if Δ is regular;
	NO otherwise;

```
IF Var(Δ) contains an accessible growing variable
    THEN output:=NO;
ELSE BEGIN
          Δ' := Δ;
          WHILE (Δ' contains α_{i,j}, such that card(α_{i,j}) > 1) DO
              BEGIN
                  I:=TRUE;
                  FOR each V ∈ Var(Δ') DO
                      IF (V ~ α_{i,j})
                          THEN BEGIN I:=FALSE;
                                     replace α_{i,j} with V;
                                     EXITFOR;
                               END;
                  IF(I=TRUE)
                      THEN BEGIN N:=newvariable(Δ');
                                 Δ' := Δ' ∪ {N =def expand(α_{i,j})};
                           END;
              END;
              output:=YES;
     END;
```

The function *expand* applies the CCS expansion law on its argument and returns the expanded expression. The function *newvariable* returns a variable from Var, which is not in $Var(\Delta)$. Here is an example:

Example 2: Let Δ be a normed BPP process in 3-GNF, given by the following set of equations:

$$\begin{aligned} X &\stackrel{def}{=} bC \ + \ a(B\|C) \\ A &\stackrel{def}{=} a \\ B &\stackrel{def}{=} b \\ C &\stackrel{def}{=} b(B\|A) \\ D &\stackrel{def}{=} aB \ + \ bA \end{aligned}$$

The set $Var(\Delta)$ does not contain any accessible growing variable, hence our algorithm can be applied. We show, how Δ' changes its form after each pass through the WHILE loop:

$$
\begin{aligned}
X &\stackrel{def}{=} bC \ + \ aE \\
A &\stackrel{def}{=} a \\
B &\stackrel{def}{=} b \\
C &\stackrel{def}{=} b(B \| A) \\
D &\stackrel{def}{=} aB \ + \ bA \\
E &\stackrel{def}{=} expand(B \| C) \ = \ bC \ + \ b(B \| B \| A)
\end{aligned}
$$

$$
\begin{aligned}
X &\stackrel{def}{=} bC \ + \ aE \\
A &\stackrel{def}{=} a \\
B &\stackrel{def}{=} b \\
C &\stackrel{def}{=} bD \\
D &\stackrel{def}{=} aB \ + \ bA \\
E &\stackrel{def}{=} bC \ + \ b(B \| B \| A)
\end{aligned}
$$

$$
\begin{aligned}
X &\stackrel{def}{=} bC \ + \ aE \\
A &\stackrel{def}{=} a \\
B &\stackrel{def}{=} b \\
C &\stackrel{def}{=} bD \\
D &\stackrel{def}{=} aB \ + \ bA \\
E &\stackrel{def}{=} bC \ + \ bF \\
F &\stackrel{def}{=} expand(B \| B \| A) = a(B \| B) \ + \ b(A \| B)
\end{aligned}
$$

$$
\begin{aligned}
X &\stackrel{def}{=} bC \ + \ aE \\
A &\stackrel{def}{=} a \\
B &\stackrel{def}{=} b \\
C &\stackrel{def}{=} bD \\
D &\stackrel{def}{=} aB \ + \ bA \\
E &\stackrel{def}{=} bC \ + \ bF \\
F &\stackrel{def}{=} aG \ + \ b(A \| B) \\
G &\stackrel{def}{=} expand(B \| B) = bB
\end{aligned}
$$

$$
\begin{aligned}
X &\stackrel{def}{=} bC + aE \\
A &\stackrel{def}{=} a \\
B &\stackrel{def}{=} b \\
C &\stackrel{def}{=} bD \\
D &\stackrel{def}{=} aB + bA \\
E &\stackrel{def}{=} bC + bF \\
F &\stackrel{def}{=} aG + bD \\
G &\stackrel{def}{=} bB
\end{aligned}
$$

3.4 Replacing merge with the full parallel operator

A natural question is, whether our algorithm still works if we replace the merge operator with the full parallel operator and thus move to the class BPP_τ. The answer is positive, but some modifications of definitions and proofs are needed. We could in fact start the Section 3 directly with this version of our algorithm, but the main idea is more or less the same - therefore we have given the simplified version first and now we show what has to be changed to obtain the full result.

We begin with a new version of the inheritence tree (we denote it IT_τ). Let Δ be a normed BPP_τ process and let $X_1 = \beta_0 \stackrel{a_0}{\to} \beta_1 \stackrel{a_1}{\to} \beta_2 \stackrel{a_2}{\to} \beta_3 \stackrel{a_3}{\to} \ldots$ be an infinite sequence of transitions. Each transition $\beta_i \stackrel{a_i}{\to} \beta_{i+1}$ is either due to a single variable X_{r_i}, which emits the action a_i and enters the state α_{r_i,s_i}, or due to a communication between two variables X_{r_i} and X_{t_i}, which emit the complementary actions $b, \overline{b} \in Act$ (hence producing τ) and enter states α_{r_i,s_i} and α_{t_i,u_i}. In the latter case we say that the transition $\beta_i \stackrel{a_i}{\to} \beta_{i+1}$ is *synchronized*, variables X_{r_i} and X_{t_i} are *communicating* and $\alpha_{r_i,s_i}, \alpha_{t_i,u_i}$ are *steps of communication*. Nodes of IT_τ are defined as previously:

$$Nodes_\tau = \bigcup_{i=0}^{\infty} \{[i,j] \mid 0 \leq j < card(\beta_i)\}$$

Now we introduce a new version of the function $Label$, denoted $Label_\tau$:

$$Label_\tau([0,0]) = X_1$$

$$Label_\tau([i+1,j]) = \begin{cases} Label([i+1,j]) & \text{if } \beta_i \stackrel{a_i}{\to} \beta_{i+1} \text{ is not synchronized} \\ Label2([i+1,j]) & \text{if } \beta_i \stackrel{a_i}{\to} \beta_{i+1} \text{ is synchronized} \end{cases}$$

where the function $Label2 : Nodes_\tau \to Var(\Delta)$ is defined as follows:

$$Label2([i+1,j]) =$$
$$= \begin{cases} Label_\tau([i,j]) & \text{if } 0 \le j < k_i \\ Lbl(r_i, s_i, j - k_i + 1) & \text{if } k_i \le j < k_i + C_1 \\ Label_\tau([i, j - C_1 + 1]) & \text{if } k_i + C_1 \le j < l_i + C_1 - 1 \\ Lbl(t_i, u_i, j - l_i - C_1 + 2) & \text{if } l_i + C_1 - 1 \le j < l_i + C_1 + C_2 - 1 \\ Label_\tau([i, j - C_1 - C_2 + 2]) & \text{if } l_i + C_1 + C_2 - 1 \le j < card(\beta_{i+1}) \end{cases}$$

where

$$k_i = \max\{j \in N \cup \{0\} \mid 0 \le j < card(\beta_i) \ \wedge \ Label_\tau[i,j] = X_{r_i}\}$$

$$l_i = \begin{cases} \max\{j \in N \cup \{0\} \mid 0 \le j < card(\beta_i) \ \wedge \ Label_\tau[i,j] = X_{t_i}\} & \text{if } X_{r_i} \neq X_{t_i} \\ \max\{j \in N \cup \{0\} \mid 0 \le j < k_i \ \wedge \ Label_\tau[i,j] = X_{t_i}\} & \text{if } X_{r_i} = X_{t_i} \end{cases}$$

where X_{r_i} and X_{t_i} are communicating variables and α_{r_i,s_i}, α_{t_i,u_i} are corresponding steps of communication. We can assume w.l.o.g., that $k_i < l_i$ (otherwise we change their roles). Constants C_1, C_2 are just abbreviations:

$$C_1 = card(\alpha_{r_i,s_i})$$
$$C_2 = card(\alpha_{t_i,u_i})$$

Finally, we define the edges of IT_τ:

$[i,j]\ Edges_\tau\ [l,m] \stackrel{def}{\iff} l = i+1$ and one of these two conditions holds:

1. $\beta_i \stackrel{a_i}{\to} \beta_{i+1}$ is not synchronized and $[i,j]\ Edges\ [l,m]$
2. $\beta_i \stackrel{a_i}{\to} \beta_{i+1}$ is synchronized and one of the following conditions holds:
   - $0 \le j < k_i \quad \wedge \quad m = j$
   - $j = k_i \quad \wedge \quad k_i \le m < k_i + C_1$
   - $k_i < j < l_i \quad \wedge \quad m = j + C_1 - 1$
   - $j = l_i \quad \wedge \quad l_i + C_1 - 1 \le m < l_i + C_1 + C_2 - 1$
   - $l_i < j < card(\beta_i) \quad \wedge \quad m = j + C_1 + C_2 - 2$

The inheritence tree associated with the sequence $X_1 = \beta_0 \stackrel{a_0}{\to} \beta_1 \stackrel{a_1}{\to} \beta_2 \stackrel{a_2}{\to} \beta_3 \stackrel{a_3}{\to} \ldots$ is a triple $IT_\tau = [Nodes_\tau,\ Edges_\tau,\ [0,0]]$. We can keep previously defined notions of the son, father, descendant and branching node. Lemmas 1, 2 and 3 are still valid. Theorem 1 holds too, but the proof has

to be done more carefully. *Seek* sets are associated with nodes of the inheritence tree in the same way. Again, if we find a node $[i,j]$, such that $Label_\tau([i,j]) \in Seek([i,j])$, we can conclude that $Label_\tau([i,j])$ is an accessible growing variable. The consideration is similar to the previous one, but the crucial thing is to realize that whenever $X|Y|\gamma \stackrel{\tau}{\rightarrow} \alpha|\beta|\gamma$, where X, Y are communicating variables and α, β are steps of communication, then also $X|Y|\gamma \stackrel{b}{\rightarrow} \alpha|Y|\gamma$ and $X|Y|\gamma \stackrel{\overline{b}}{\rightarrow} X|\beta|\gamma$, where $b, \overline{b} \in Act$ are complementary actions - there is no way how to force synchronizations. The proof is now easy to complete.

The constructive algorithm described in the previous section also requires a small modification - we need a more general version of the CCS expansion law, which reflects the new possibility of synchronizations (see [1]):

$$\begin{aligned} A_1|\ldots|A_n &= \sum \{\ a(A_1|\ldots|\alpha_i|\ldots|A_n)\ :\ A_i \stackrel{a}{\rightarrow} \alpha_i,\ a \in Act\} \\ &+ \sum \{\ \tau(A_1|\ldots|\alpha_i|\ldots|\alpha_j|\ldots|A_n)\ :\ A_i \stackrel{b}{\rightarrow} \alpha_i,\ A_j \stackrel{\overline{b}}{\rightarrow} \alpha_j,\ \ b, \overline{b} \in Act\} \end{aligned}$$

Now we can finish this section with the following theorem:

Theorem 2 *There is a constructive regularity test for the class of normed BPP_τ processes.*

4 The constructive test of regularity for normed BPA processes

In [2] Mauw and Mulder presented a constructive regularity test for BPA systems. Their notion of the BPA system is exactly what we call the BPA process here. But we keep this notion, because the regularity of BPA systems is defined differently from the regularity of processes:

Definition 12 *Let Δ be a BPA process in GNF. A variable $Y \in Var(\Delta)$ is said to be accessible if $\exists\ w \in Act^*,\ \gamma \in Var(\Delta)^*$, such that $X_1 \stackrel{w}{\rightarrow}_* Y\gamma$.*

Definition 13 *A BPA system Δ is regular iff each accessible variable $Y \in Var(\Delta)$ is a regular process.*

Remember that if we have a BPA system Δ, each of its variables can be seen as a process (see Remark 1).

Each regular BPA system is a regular BPA process. The following example shows, that there is a regular BPA process, which is not a regular

BPA system (this example is also due to [2]):

Example 3: Let Δ be a BPA process given by the following set of equations:

$$\begin{array}{lcl} X & \stackrel{def}{=} & aYZ \\ Y & \stackrel{def}{=} & bYC \ + \ d \\ Z & \stackrel{def}{=} & cZ \\ C & \stackrel{def}{=} & c \end{array}$$

It is easy to check that Δ is a regular process, but it contains an accessible variable Y, which is not regular (process Y can reach infinitely many pairwise non-bisimilar states). Hence Δ is not a regular BPA system.

Furthermore, the result of [2] is constructive - it not only checks the regularity of BPA systems, but if the answer is positive, it also outputs a regular process in the normal form, which is bisimilar to the original one. In this section we prove, that if we restrict out attention to the class of normed BPA processes, then the result of [2] can serve as a constructive regularity test for processes of this class. The following lemma is due to D. Caucal [4]:

Lemma 5 (Cancelation) *Let Δ be a normed BPA process in GNF, $\alpha, \beta, \gamma \in Var(\Delta)^*$. If $\alpha\gamma \sim \beta\gamma$, then also $\alpha \sim \beta$.*

Proof: The set $\{[\delta_1, \delta_2] \mid \delta_1, \delta_2 \in Var(\Delta)^*, \ \delta_1\gamma \sim \delta_2\gamma\}$ is a bisimulation containing the pair $[\alpha, \beta]$.

□

Now it is possible to prove the promised result:

Lemma 6 *A normed BPA process Δ is regular iff all accessible variables of $Var(\Delta)$ are regular processes.*

Proof:
"$\Leftarrow$:" trivial.
"$\Rightarrow$:" Let Y be an accessible variable, which is not a regular process. Due to Lemma 2 there must be an infinite sequence $Y = \alpha_0 \stackrel{a_0}{\rightarrow} \alpha_1 \stackrel{a_1}{\rightarrow} \alpha_2 \stackrel{a_2}{\rightarrow} \alpha_3 \ldots$, such that $\alpha_i \not\sim \alpha_j$ for $i \neq j$. As Y is accessible, there exists $w \in Act^*$, such that $X_1 \stackrel{w}{\rightarrow}_* Y\beta$. But then the states $\alpha_0\beta, \ \alpha_1\beta, \ \alpha_2\beta, \ \ldots$ are reachable. As Δ is regular, it cannot have infinitely many reachable states, which are

pairwise non-bisimilar. Thus there are $i, j \in N,\ i \neq j$, such that $\alpha_i\beta \sim \alpha_j\beta$. But now we can use the Lemma 5 and conclude $\alpha_i \sim \alpha_j$, so we have a contradiction.

□

We present here also the main theorem of [2] (in a slightly different form):

Definition 14 *Let Δ be a BPA system. A variable $Y \in Var(\Delta)$ is said to be growing iff $\exists w \in Act^+,\ \alpha \in Var(\Delta)^*$, such that $Y \xrightarrow{w} Y\alpha$ and $Y\alpha$ is a normed state.*

Theorem 3 *Let Δ be a BPA system. System Δ is regular iff $Var(\Delta)$ does not contain any accessible growing variable.*

Proof: can be found in [2].

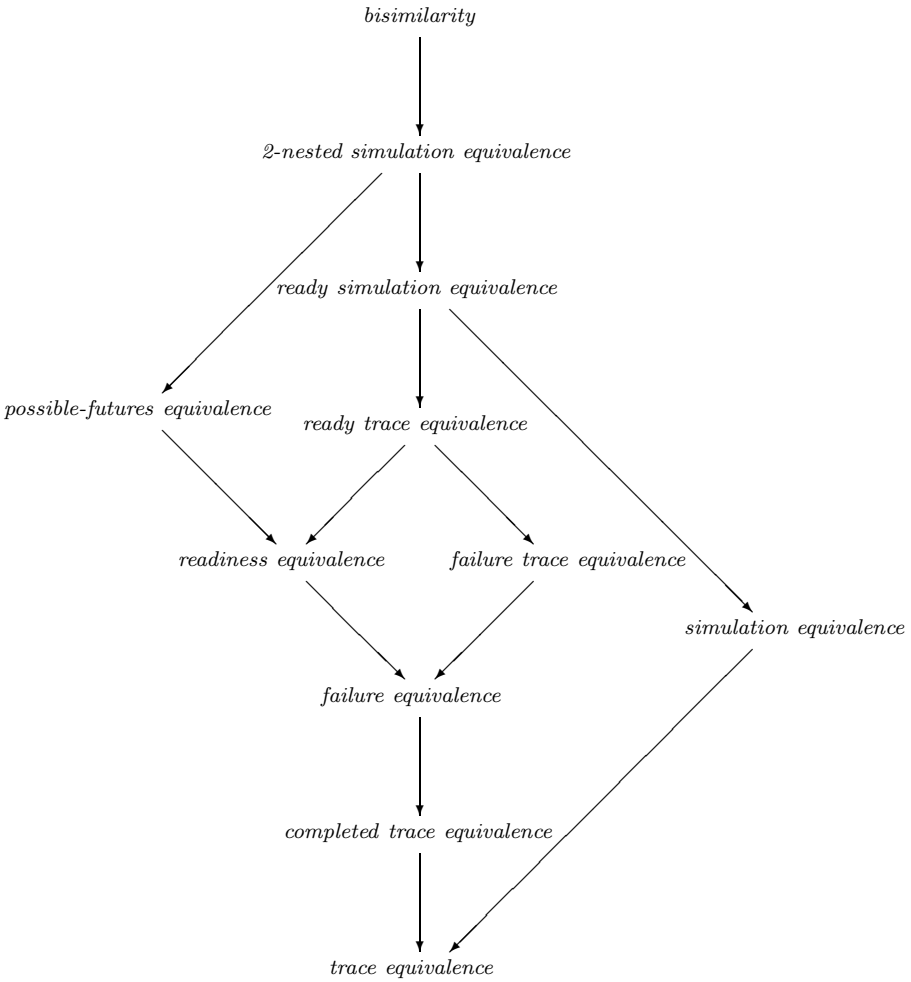
□

As a consequence of Theorem 3 and Lemma 6 we get the following:

Theorem 4 *Let Δ be a normed BPA process. Process Δ is regular iff $Var(\Delta)$ does not contain any accessible growing variable.*

5 Deciding regularity w.r.t. other behavioural equivalences

Bisimilarity is not the only behavioural equivalence which appeared in the literarure. In certain situations it may be advantageous to define the notion of "sameness" in another way. R. van Glabbeek presented in [13] various equivalences in a uniform way, relating them w.r.t. their *coarseness*, i.e. how many identifications they make:



The order is determined by the relation "makes strictly more identifications than". Definitions of these equivalences can be found in Appendix A.

The notion of regularity can be defined also w.r.t. these equivalences in the same way as in the case of bisimilarity. In this section we examine properties of these equivalences. We introduce notions of strong regularity and finite characterisation and then we describe their relationship with regularity and finite representations.

As we want to keep this section general, we abstract from the concrete model of process algebras and assume that all behavioural equivalences are

binary relations over the class of transition systems:

Definition 15 A transition system *is a 4-tuple (N,L,→,r) consisting of a set of* nodes *N, a set of* labels *L, a* transition relation $\rightarrow \subseteq N \times L \times N$ *and a distinguished element* $r \in N$ *called* root.

As usual we write $A \xrightarrow{a} B$ if $[A, a, B] \in \rightarrow$ and we extend this notation also to elements of L^* in an obvious way. A node $A \in N$ is *reachable* if there is $w \in L^*$ such that $r \xrightarrow{w} A$. The class of all transition systems is denoted by $\mathcal{T}$.

Remark 3 *Each node A of the transition system* $T = [N, L, \rightarrow, r]$ *determines a unique transition system* $T(A) = [N, L, \rightarrow, A]$. *All notions originally defined for transition systems can be used for their nodes in this sense too.*

Definition 16 *Let* $\leftrightarrow$ *be an equivalence over* $\mathcal{T}$. *A transition system* $T \in \mathcal{T}$ *is said to be* regular w.r.t. $\leftrightarrow$ *if there is a finite system* $T' \in \mathcal{T}$, *such that* $T \leftrightarrow T'$.

The transition system T' from the previous definition can be seen as a finite *representation* of T, because it represents the behaviour of the process which is associated with the root of T. As we will see, representations generally do not say much about the behaviour of reachable nodes of T. We need another notion:

Definition 17 *Let* $T \in \mathcal{T}$ *be a transition system and let* $\leftrightarrow$ *be an equivalence over* $\mathcal{T}$. T *has a* finite characterisation w.r.t. $\leftrightarrow$ *if there is a finite* $T' \in \mathcal{T}$, *whose nodes are pairwise non-equivalent w.r.t.* $\leftrightarrow$, $T \leftrightarrow T'$ *and for each reachable node n of* T *there is a reachable node* n' *of* T' *with* $n \leftrightarrow n'$.

A finite characterisation T' of T describes the system T as a whole - for each reachable node of T there is its finite characterisation within T'. An existence of a finite characterisation is especially interesting from the point of view of possible verification of concurrent systems.

Now we examine the question when finite characterisations exist and what is their relationship with representations. First we need to introduce further notions:

Definition 18 *Let* $T \in \mathcal{T}$ *be a transition system and let* $\leftrightarrow$ *be an equivalence over* $\mathcal{T}$. T *is* strongly regular *w.r.t.* $\leftrightarrow$ *if each reachable node of* T *is regular and* T *can reach only finitely many nodes up to* $\leftrightarrow$.

Definition 19 *Let $\leftrightarrow$ be an equivalence over $\mathcal{T}$. For each $T \in \mathcal{T}$ we define the transition system $T/\leftrightarrow$: Nodes of $T/\leftrightarrow$ are equivalence classes of $\leftrightarrow$, root is the class $[r]$ and transitions are determined as follows: if $n \xrightarrow{a} n'$ is a transition in T, then $[n] \xrightarrow{a} [n']$ is a transition in $T/\leftrightarrow$.*

The equivalence $\leftrightarrow$ is said to have quotients *if for any $T \in \mathcal{T}$ the natural projection $p : T \longrightarrow T/\leftrightarrow$, assigning to each node n of T the node $[n]$ of $T/\leftrightarrow$, is a part of $\leftrightarrow$ (i.e. $n \leftrightarrow [n]$ for each node n of T).*

Lemma 7 *Let $T \in \mathcal{T}$ and let $\leftrightarrow$ be an equivalence over $\mathcal{T}$ which has quotients. Then T has a finite characterisation w.r.t. $\leftrightarrow$ iff T is strongly regular w.r.t. $\leftrightarrow$.*

Proof:
"$\Rightarrow$" Let T' be a finite characterisation of T. Each reachable node n of T is regular, because it is equivalent to some node n' of T' and T' is finite. Assume that T can reach infinitely many pairwise non-equivalent nodes $n_i, i \in N$. Each n_i is equivalent to some node n'_i of T'. As T' is finite, there are $i, j \in N,\ i \neq j$ such that $n'_i \leftrightarrow n'_j$. Hence also $n_i \leftrightarrow n_j$ and we have a contradiction.
"$\Leftarrow$" As T is strongly regular and $\leftrightarrow$ has quotients, the transition system $T/\leftrightarrow$ is a finite characterisation of T.

□

The first theorem of this section shows, that the requirement of "having quotients" of the previous lemma is not too restrictive in fact. There are many reasonable equivalences, which fulfil this condition.

Lemma 8 *Equivalences $=_{tr},\ =_{ct},\ =_f,\ =_r,\ =_{ft},\ =_{rt},\ =_{pf}$ have quotients.*

Proof: We will not give a separate proof for each of these equivalences, because the main idea is always the same. It corresponds to the fact, that all these equivalences are defined in a similar way. The crucial thing is to realize, that in spite of the fact that none of these equivalences is a congruence w.r.t. the transition relation, equivalent nodes have always the same sets of initial actions (see Appendix A). We present here a full proof for failure equivalence. The other proofs should be easy to complete using the same kind of argument.

Let $T \in \mathcal{T}$ be a transition system and let $n \in N$ be a node of T. We show that $F(n) = F([n])$, where $[n]$ denotes the equivalence class of $T/=_f$

containing the node n:

"$\subseteq$": Let $[w, \Phi] \in L^* \times \mathcal{P}(L)$ be a failure pair of n (see Appendix A). By definition, there is a node $n' \in N$ such that $n \stackrel{w}{\rightarrow} n' \ \wedge \ I(n') \cap \Phi = \emptyset$. But then also $[n] \stackrel{w}{\rightarrow} [n']$. The set $I([n'])$ is the union of all $I(p)$, such that $p \in [n']$. As $p =_f q$ implies $I(p) = I(q)$, we can conclude that $I([n']) = I(n')$, hence $I([n']) \cap \Phi = \emptyset$, thus $[w, \Phi] \in F([n])$.

"$\supseteq$": Let $[w, \Phi] \in L^* \times \mathcal{P}(L)$ be a failure pair of $[n]$ and let $w = a_k a_{k-1} \ldots a_1$. By definition, there is a sequence $[n_k] \stackrel{a_k}{\rightarrow} [n_{k-1}] \stackrel{a_{k-1}}{\rightarrow} \ldots \stackrel{a_1}{\rightarrow} [n_0]$ in $T/ =_f$, such that $n \in [n_k]$ and $I([n_0]) \cap \Phi = \emptyset$. We show, that for each node m of T such that $m \in [n_i]$, where $i \in \{0, \ldots, k\}$, the pair $[a_i \ldots a_1, \Phi]$ belongs to $F(m)$. We proceed by induction on i:

- $i = 0$: as $I(m) = I([n_0])$, the pair $[\epsilon, \Phi] \in F(m)$.
- induction step: as $[n_i] \stackrel{a_i}{\rightarrow} [n_{i-1}]$, there are nodes p, q of T, such that $p \stackrel{a_i}{\rightarrow} q$, $p \in [n_i]$ and $q \in [n_{i-1}]$. By induction hypothesis, the pair $[a_{i-1} \ldots a_1, \Phi] \in F(q)$, hence $[a_i \ldots a_1, \Phi] \in F(p)$. As $m =_f p$, the pair $[a_i \ldots a_1, \Phi]$ belongs to $F(m)$.

□

Lemma 9 *Simulation equivalence, ready simulation equivalence and 2-nested simulation equivalence have quotients.*

Proof: Let $T \in \mathcal{T}$ be a transition system and let $n \in N$ be a node of T. First we show that $n =_s [n]$, where $[n]$ denotes the equivalence class of $T/ =_s$. By definition, two simulations R, S, such that $[n, [n]] \in R$, $[[n], n] \in S$ have to be defined. The simulation R is exactly the natural projection $p : T \rightarrow T/ =_s$:

$$R = \{[k, [k]] : k \in N\}$$

It is easy to check, that R is indeed a simulation. The way how S is defined is more complicated:

$$[[p], q] \in S \textit{ iff there exists a } \text{derivation scheme } \textit{for } [[p], q].$$

The derivation scheme for $[[p], q]$ consists of:

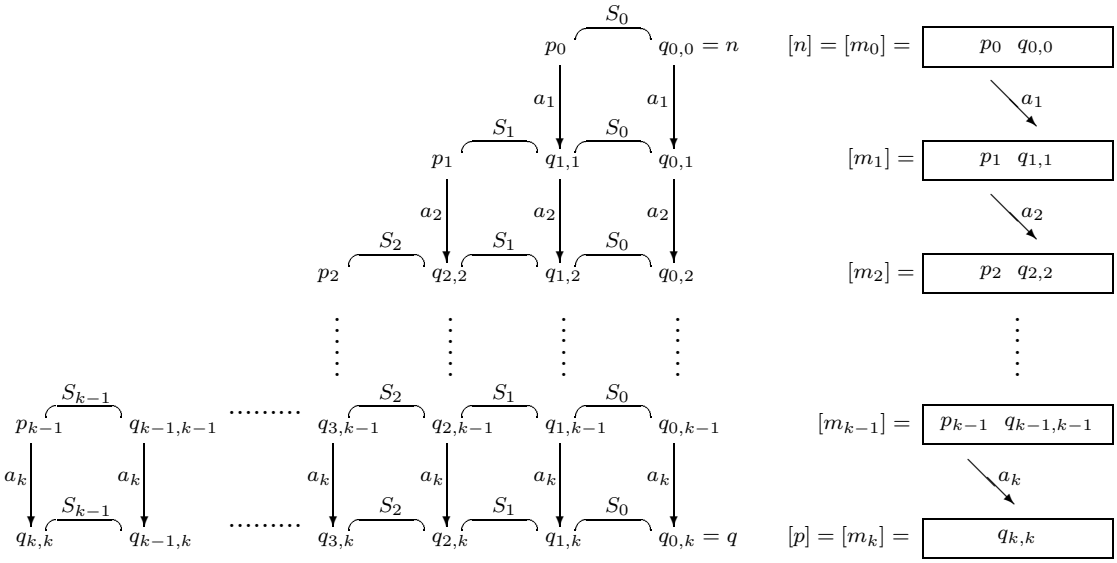
- a path $[m_0] \stackrel{a_1}{\rightarrow} [m_1] \stackrel{a_2}{\rightarrow} \ldots \stackrel{a_k}{\rightarrow} [m_k]$ in $T/ =_s$, $k \geq 0$
- a set of nodes $\{p_0, \ldots, p_{k-1}\} \subseteq N$, *if* $k > 0$

- a set of nodes $\{q_{i,j} \mid 0 \leq i \leq k,\ i \leq j \leq k\} \subseteq N$
- a set of simulations $\{S_0, \ldots, S_{k-1}\},\ \textit{if}\ k > 0$

such that:

- $n = q_{0,0}$, $q = q_{0,k}$, $n \in [m_0]$, $p \in [m_k]$
- $p_i \in [m_i]$ for $i \in \{0, \ldots, k-1\}$, $q_{i,i} \in [m_i]$ for $i \in \{0, \ldots, k\}$
- $p_i \overset{a_{i+1}}{\rightarrow} q_{i+1,i+1}$ for $i \in \{0, \ldots, k-1\}$
- $q_{i,j} \overset{a_{i+1}}{\rightarrow} q_{i,j+1}$ for $0 \leq i \leq k-1$, $i \leq j < k-1$
- $[p_i, q_{i,i}] \in S_i$ for $i \in \{0, \ldots, k-1\}$
- $[q_{i+1,j}, q_{i,j}] \in S_i$ for $0 \leq i \leq k-1$, $i < j \leq k$

The following picture could be helpful:



Relation S is a simulation - whenever $[[p], q] \in S$ and $[p] \overset{a}{\rightarrow} [p']$, then there is a q', such that $q \overset{a}{\rightarrow} q'$ and $[[p'], q'] \in S$. This is due to the existence of a derivation scheme for the pair $[[p], q]$. We can simply add a new "layer" to the scheme and construct a derivation scheme for the pair $[[p'], q']$. The way how it is done is obvious. Moreover, S contains the pair $[[n], n]$.

This construction can be used also for ready simulation equivalence. The simulation R becomes a ready simulation. It follows directly from the fact that two nodes, which are ready simulation equivalent, have the same sets of initial actions. The notion of the derivation scheme has to be modified slightly - we now require $\{S_0, \ldots, S_{k-1}\}$ to be a set of ready simulations. Then S is also a ready simulation: assume that $[[p], q] \in S$. Then $I([p]) = I(q)$ because $q_{k,k} \in [p]$ and the simulations $S_0, \ldots, S_{k-1}$ are ready simulations now.

In the case of 2-nested simulation equivalence the construction can be used too. The simulation R becomes a 2-nested simulation, because we have already proved that $n =_s [n]$ for each node n of T. The notion of the derivation scheme has to be modified again - $\{S_0, \ldots, S_{k-1}\}$ are 2-nested simulations now. We prove that S is a 2-nested simulation. Let $[[p], q] \in S$. We need to show that $[p] =_s q$. By definition, two simulations P, Q such that $[[p], q] \in P$ and $[q, [p]] \in Q$ have to be constructed. Clearly S is a simulation which contains the pair $[[p], q]$, hence we can choose $P = S$. The construction of Q is slightly more complicated. As $[[p], q] \in S$, there is a derivation scheme for $[[p], q]$. Now $S_0, \ldots, S_{k-1}$ are not only simulations, but 2-nested simulations, hence $q_{k,k} =_s q$. Therefore there is a simulation T containing the pair $[q, q_{k,k}]$. It is easy to check that $Q = \{[u, [v]] : [u, v] \in T\}$ is a simulation. Moreover, $[q, [p]] \in Q$ because $q_{k,k} \in [p]$.

□

We have proved the first theorem of this section:

Theorem 5 *Each equivalence in van Glabbeek hierarchy has quotients.*

There are also other well-known equivalences which have quotients - e.g. weak bisimilarity (see [1]) or branching bisimilarity (see [14]). But this property is naturally not general; there are also equivalences which do not have quotients. A simple example is language equivalence. Two transition systems are language equivalent if their roots have the same completed traces (realize that language equivalence is different from completed trace equivalence and is even incomparable with trace equivalence). A counterexample is easy to find.

We have seen that in many cases the condition of strong regularity becomes sufficient and necessary for the existence of a finite characterisation. An interesting question is, what is the exact relationship between conditions of regularity and strong regularity (the first one guarantees the existence of

a finite representation, the other guarantees the existence of a finite characterisation). Strong regularity always implies regularity, but the converse is not generally true.

Definition 20 *An equivance $\leftrightarrow$ over $\mathcal{T}$ is* safe *if whenever $T \leftrightarrow T'$ then:*

- *for each reachable node n of T there is a reachable node n' of T' such that $n \leftrightarrow n'$*
- *for each reachable node n' of T' there is a reachable node n of T such that $n \leftrightarrow n'$*

Lemma 10 *Let $\leftrightarrow$ be a safe equivalence over $\mathcal{T}$. Then T is strongly regular w.r.t. $\leftrightarrow$ iff T is regular w.r.t. $\leftrightarrow$*

Proof:
"$\Rightarrow$" Obvious.
"$\Leftarrow$" The arguments of Lemma 7 can be used.

□

An immediate consequence of Lemma 7 and Lemma 8 is:

Lemma 11 *Let $\leftrightarrow$ be a safe equivalence over $\mathcal{T}$ which has quotients. Then $\forall T \in \mathcal{T}$: T has a finite representation iff T has a finite characterisation.*

Thus in the case of a safe equivalence which has quotients the notions of regularity and strong regularity coincide. We have already mentioned some examples - bisimilarity, weak bisimilarity and branching bisimilarity are safe and have quotients. But there are also equivalences, for which these two notions are really different.

Lemma 12 *For each behavioural equivalence $\leftrightarrow$ which lies under ready simulation equivalence in van Glabbeek hierarchy (including this relation) there is a transition system $T \in \mathcal{T}$, such that T is regular w.r.t. $\leftrightarrow$ and T is not strongly regular w.r.t. $\leftrightarrow$.*

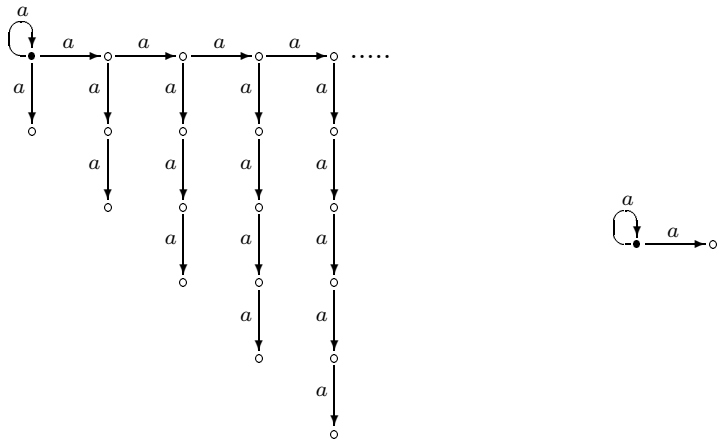
Proof: Let $T_1 = [N_1, L_1, \rightarrow_1, r_1]$, $T_2 = [N_2, L_2, \rightarrow_2, r_2]$ be transition systems, where $N_1 \subseteq N \cup \{0\} \times N \cup \{0\}$:

$$N_1 \;=\; \bigcup_{i=0}^{\infty} \{[i,j] \mid 0 \leq j \leq i+1\}$$

$$
\begin{aligned}
L_1 &= \{a\} \\
\rightarrow_1 &= \bigcup_{i=0}^{\infty} \{\ [[i,j],a,[i,j+1]] \mid j \in \{0,\ldots,i\}\} \ \cup\ \{[[0,0],a,[0,0]]\} \\
&\quad \cup \{\ [[i,0],a,[i+1,0]] \mid i \in N \cup \{0\}\} \\
r_1 &= [0,0]
\end{aligned}
$$

$$
\begin{aligned}
N_2 &= \{A,B\} \\
L_2 &= \{a\} \\
\rightarrow_2 &= \{[A,a,A],[A,a,B]\} \\
r_2 &= A
\end{aligned}
$$

If we draw these transition systems, we obtain the following pictures:



T_1 $\qquad\qquad$ T_2

System T_1 is not strongly regular w.r.t. trace equivalence, because $tr([i,1]) \subset tr([i+1,1])$ for each $i \in N \cup \{0\}$, thus T_1 contains infinitely many states w.r.t. trace equivalence. Therefore T_1 is not strongly regular w.r.t. any equivalence in van Glabbeek hierarchy.

Now we show that $T_1 =_{rs} T_2$. By definition, two ready simulations R, S, such that $r_1 R r_2$ and $r_2 S r_1$ have to be constructed:

$$
\begin{aligned}
R &= \bigcup_{i=0}^{\infty} \{\ [[i,j],A] :\ 0 \le j \le i\} \ \cup\ \bigcup_{i=0}^{\infty} \{\ [[i,i+1],B]\ \} \\
S &= \{\ [A,[0,0]],\ [B,[0,1]]\ \}
\end{aligned}
$$

It is easy to check that R, S are ready simulations. Moreover, $[0,0] = r_1\ R\ r_2 = A$ and $A = r_2\ S\ r_1 = [0,0]$.

As $T_1 =_{rs} T_2$, transition systems T_1, T_2 are equivalent w.r.t. any behavioural equivalence which lies under ready simulation equivalence in van Glabbeek hierarchy. As T_2 is finite, the system T_1 is regular w.r.t. each of these equivalences.

□

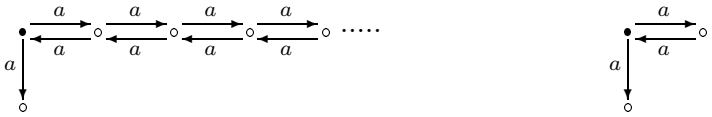
Lemma 13 *There is a transition system $T \in \mathcal{T}$, such that T is regular w.r.t. possible-futures equivalence and 2-nested simulation equivalence, but T is not strongly regular w.r.t. these equivalences.*

Proof: Let $T_1 = [N_1, L_1, \rightarrow_1, r_1]$, $T_2 = [N_2, L_2, \rightarrow_2, r_2]$ be transition systems, where:

$$\begin{aligned} N_1 &= N \cup \{0\} \\ L_1 &= \{a\} \\ \rightarrow_1 &= \{\ [i, i+1] \mid i \in N\} \cup \{\ [i, i-1] \mid i \in N\} \\ r_1 &= 1 \end{aligned}$$

$$\begin{aligned} N_2 &= \{A, B, C\} \\ L_2 &= \{a\} \\ \rightarrow_2 &= \{\ [A, a, B], [A, a, C], [C, a, A]\ \} \\ r_2 &= A \end{aligned}$$

Systems T_1, T_2 can be depicted as follows:



T_1 $\qquad\qquad$ T_2

We show that T_1 has infinitely many nodes w.r.t. $=_{pf}$ and $=_2$. Let $i, j \in N,\ i < j$ be nodes of T_1. The node i has a possible future $[a^i, \emptyset]$. Clearly $[a^i, \emptyset] \notin PF(j)$, hence $i \neq_{pf} j$. As 2-nested simulation equivalence is above possible-futures equivalence in van Glabbeek hierarchy, the system T_1 has

infinitely many states also w.r.t. $=_2$, thus T_1 is not strongly regular w.r.t. $=_2$ and $=_{pf}$.

It remains to prove that T_1 is regular w.r.t. $=_2$ and $=_{pf}$. We show, that $T_1 =_2 T_2$. First we have to classify which nodes of T_1 and T_2 are simulation equivalent. Clearly $0 =_s B$. If $i \in N$ is odd then $i =_s A$, and if $i \in N$ is even then $i =_s C$. Following relations are the required simulations:

$$\begin{aligned} R_i &= \{\ [k, A] \mid k \in N \wedge k \textit{ is odd}\} \cup \{\ [k, C] \mid k \in N \cup \{0\} \wedge k \textit{ is even}\} \\ S_i &= \{\ [A, i], [B, i+1], [C, i+1]\ \} \end{aligned}$$

Now we can define two 2-nested simulations, which relate roots of T_1 and T_2:

$$\begin{aligned} R &= \{\ [i, A] \mid i \in N \wedge i \textit{ is odd}\} \cup \{\ [i, C] \mid k \in N \wedge i \textit{ is even}\} \cup \{[0, B]\} \\ S &= \{\ [A, 1], [B, 0], [C, 2]\ \} \end{aligned}$$

Elements of R, S are pairs of simulation equivalent nodes. Now it is easy to check that R, S are 2-nested simulations. As $[1, A] \in R \ \wedge \ [A, 1] \in S$, transition systems T_1, T_2 are 2-nested simulation equivalent.

As $T_1 =_2 T_2$ and possible-futures equivalence lies under 2-nested simulation equivalence in van Glabbeek hierarchy, systems T_1 and T_2 are also possible-futures equivalent. Thus T_1 is regular w.r.t. $=_2$ and $=_{pf}$

□

We have just proved the following theorem:

Theorem 6 *Let $\leftrightarrow$ be an equivalence in van Glabbeek hierarchy, which lies under bisimilarity. Then there is $T \in \mathcal{T}$, such that T is regular w.r.t. $\leftrightarrow$ and T is not strongly regular w.r.t. $\leftrightarrow$*

An open problem is, whether the notions of regularity and strong regularity have different features w.r.t. their decidability. In the next section we present some negative results, stating that both regularity and strong regularity can be undecidable in certain process algebras. From the practical point of view it would be much more interesting to obtain some positive results, but this area seems to be quite unexplored. Recently, Jančar and Moller presented in [15] an interesting result, stating that trace, simulation and bisimulation equivalence are decidable for pairs of Petri nets, such that one member of this pair is a bounded Petri net (i.e. a finite-state process).

6 Negative results

In this section we present some negative results, stating that regularity and strong regularity are undecidable in some process algebras. This area was first examined by Taubner in [11]. We generalize these results using a simple reduction of the halting problem of Minsky machine.

6.1 The Minsky machine

A Minsky machine (denoted here by M) is equipped with two counters $C_1,\ C_2$, which can store non-negative integers. The behaviour of M is determined by a finite-state program, composed of $m \in N$ labelled statements:

$$\begin{array}{lcl} l_1 & : & s_1 \\ l_2 & : & s_2 \\ \vdots & & \\ l_{m-1} & : & s_{m-1} \\ l_m & : & \texttt{HALT} \end{array}$$

where for each $i,\ 1 \leq i < m$ the statement s_i has one of the two forms:

$$s_i = \begin{cases} C_j = C_j + 1; \texttt{goto}\ l_k \\ \texttt{if}\ C_j = 0\ \texttt{then goto}\ l_k\ \texttt{else}\ C_j = C_j - 1; \texttt{goto}\ l_n; \end{cases}$$

where $j \in \{1, 2\}$. The machine M starts its execution (with given input values on $C_1,\ C_2$) from the command with the label l_1. M *halts* if it reaches the command HALT in a finite number of steps, and *diverges* otherwise. Naturally, the halting problem of Minsky machine is undecidable (see [12]).

6.2 Extending BPP_τ with the operator of restriction

In this section we explore a calculus obtained by extending BPP_τ with the restriction operator. In [11] Taubner proved that there is no algorithm which, for some process Δ of this class as input, outputs a regular process Δ' in normal form with $\Delta \sim \Delta'$ if such a Δ' exists, and which outputs "no" otherwise. We extend this result also for other equivalences and for the notion of strong regularity, which was introduced in the previous section. All negative results are proved in a uniform way using a very simple technique.

We begin by formally introducing the restriction operator. Let L be a subset of Act, such that $\tau \notin L$. The restriction operator, denoted by "$\backslash L$",

has the following meaning:

$$\frac{E \xrightarrow{a} E'}{E\backslash L \xrightarrow{a} E'\backslash L} \ (a \notin L \cup \overline{L})$$

It is mainly used as a tool for forcing synchronizations on certain actions. If we extend BPP_τ expressions with the restriction operator, we get a new class of processes, denoted by BPP_τ^R.

Processes of BPP_τ^R are able to simulate the execution of an arbitrary Minsky machine M (see [11]). The simulating process can be automaticly constructed and has the following form:

$$S = (K_1|K_2|P_1)\backslash L$$

Processes $K_1,\ K_2$ simulate counters. We will not describe the way how they are defined - it is not important for our purposes (see [11] or [10] for details). Process P_1 simulates the program of M and L contains all visible actions of $K_1,\ K_2$ and P_1, forcing the three components to cooperate.

The program of M is simulated by P_1, which consists of m defining equations containing variables from $\{P_1, \ldots, P_m\}$. Each equation is determined as follows:

1. $P_i \stackrel{def}{=} i_j P_{l_k}$
   if s_i is of the form $C_j = C_j + 1$; `goto` l_k;

2. $P_i \stackrel{def}{=} z_j P_{l_k} + d_j P_{l_n}$
   if s_i is of the form `if` $C_j = 0$ `then goto` l_k `else` $C_j = C_j - 1$; `goto` l_n;

3. $P_i \stackrel{def}{=} 0$
   if $s_i =$ `HALT`

Actions $i_j, d_j, z_j,\ j \in \{1, 2\}$ represent operations on counters and have their complements in K_1 and K_2. The expression 0 denotes a process which does nothing (it is an explicit name for the empty process expression). The execution of M is thus simulated by communications among $K_1,\ K_2$ and P_1. Machine M diverges iff $S \sim X$, where $X \stackrel{def}{=} \tau X$.

Lemma 14 *There is a BPP_τ^R process Y, which is not regular w.r.t. trace equivalence.*

Proof: Let $Y \stackrel{def}{=} (a.(A|C))\backslash\{b\}$, $A \stackrel{def}{=} a.(A|C) + b$, $C \stackrel{def}{=} \overline{b}.c.b$. Process Y is normed, hence its traces are just prefixes of its completed traces. The set $ct(Y) = \{a^k(\tau c)^k \mid k \in N\}$. Assume that there is a finite process Y' with n states, such that Y and Y' are trace equivalent. As $a^{n+1}(\tau c)^{n+1} \in ct(Y)$, this sequence of actions is also a trace of Y'. As Y' has only n states, there is a state φ of Y' such that

$$Y' \stackrel{a^k}{\rightarrow} \varphi \stackrel{a^l}{\rightarrow} \varphi \stackrel{a^m}{\rightarrow} \psi \stackrel{(\tau c)^{n+1}}{\rightarrow} \chi$$

where ψ, χ are states of Y', $k + l + m = n + 1$ and $l > 0$. But then also

$$Y' \stackrel{a^k}{\rightarrow} \varphi \stackrel{a^m}{\rightarrow} \psi \stackrel{(\tau c)^{n+1}}{\rightarrow} \chi$$

thus $a^k a^m (\tau c)^{n+1}$ is a trace of Y'. But this sequence of actions is not a prefix of any completed trace of Y, hence $a^k a^m (\tau c)^{n+1} \notin tr(Y)$ and $Y \neq_{tr} Y'$.

□

As Y is not regular w.r.t. trace equivalence, it is not (strongly) regular w.r.t. any equivalence in van Glabbeek hierarchy.

Theorem 7 *Regularity w.r.t. any equivalence in van Glabbeek hierarchy is undecidable in the class* BPP_τ^R.

Proof: Let $\leftrightarrow$ be an equivalence in van Glabbeek hierarchy. We define a new process P_1' with variables from $\{P_1', \ldots, P_m', A, C\}$. These variables are defined in the same way as variables of P_1 except A, C and P_i', where $s_i = HALT$ (we denote this variable just P'):

$$P' \stackrel{def}{=} (a.(A|C))\backslash\{b\}, \quad A \stackrel{def}{=} a.(A|C) + b, \quad C \stackrel{def}{=} \overline{b}.c.b$$

where $a, b, \overline{b}, c \notin L$. Let $S' = (K_1|K_2|P_1')\backslash L$. If M diverges, then S' is regular w.r.t $\leftrightarrow$, because $S' \sim X$ where $X \stackrel{def}{=} \tau X$, thus also $S' \leftrightarrow X$. If S' is regular w.r.t $\leftrightarrow$, then it cannot reach a state of the form $(K_1'|K_2'|P')\backslash L$ in a finite number of τ moves. To see this, assume the converse. As $(K_1'|K_2'|P')\backslash L \sim P'$ and $(K_1'|K_2'|P')\backslash L$ was reached from S' under a finite number of τ moves, we can conclude $S' \sim \tau^k.P'$ where $k \in N$ (realize that the first k moves of S' are completely deterministic). The process $\tau^k.P'$ is clearly non-regular w.r.t. trace equivalence (arguments of Lemma 10 can be used), hence S' is also non-regular w.r.t. trace equivalence (otherwise we can use the fact

that bisimilarity implies trace equivalence and conclude that $\tau^k.P'$ is regular w.r.t. trace equivalence just by transitivity). As S' is not regular w.r.t trace equivalence, it is not regular w.r.t. $\leftrightarrow$ and we have a contradiction.

We have just proved that M diverges iff S' is regular w.r.t. $\leftrightarrow$. As S' can be for any Minsky machine M constructed by an algorithm, we have the desired reduction.

□

This method can be used also for other equivalences and other process algebras which have a parallel operator and can simulate counters (see [11]). Moreover, it works also for strong regularity.

7 Conclusions, future work

If we compare the decidability results, obtained for classes of normed BPP and normed BPA processes, we can observe that they are of a similar form. This is not surprising if fact - the only difference between BPP and BPA algebras is the way of binary composition they provide - the parallel composition in the case of BPP and the sequential composition in the case of BPA. But these two operators have similar algebraic properties and it reflects in many things - processes of BPP and BPA can be represented in similar normal forms (GNF), there are similar cancelation properties, the notion of self-bisimulation, introduced in [4], can be defined in a uniform way (see [9]) and so on.

An open problem still remains the question of deciding regularity in the whole classes of BPP and BPA. This problem is at least semi-decidable, because bisimilarity is known to be decidable in these algebras - hence we can take a BPA or BPP process Δ in GNF and start to remove sequences (in the case of BPA) or multisets (in the case of BPP) of variables, whose cardinality is greater then one, from defining equations. We have already described the removal procedure for BPP class (see Section 3.3). The procedure for BPA is similar, but the right distributivity law (see [3]) has to be used instead of the expansion law.

Another interesting question mentioned already in the Section 5 is, whether there are behavioural equivalences, for which conditions of regularity and strong regularity have different decidability properties and this is the area we would like to examine in the future.

Acknowledgement

All presented results were obtained during my stay at BRICS (Basic Research in Computer Science), Department of Computer Science, University of Aarhus. I would like to thank Mogens Nielsen for his support and encouragement. He was always willing to listen to my ideas, commenting them in a very inspiring way. Thanks are also due to Mojmír Křetínský for reading the first draft of this paper and providing me with various helpful information.

References

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

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

[3] J.C.M. Beaten, J.A. Bergstra, J.W. Klop. *Decideability of bisimulation equivalence for processes generating contex-free languages.* In *LNCS 259*, pages 93-114. Springer-Verlag, 1987.

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

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

[6] H. Hüttel, C. Stirling. *Actions speak louder than words: Proving bisimilarity for contex free processes.* In Proceedings of the 6th annual symposium on logic in computer science (LICS91), pages 376-386. IEEE computer society press, 1991.

[7] S. Christensen, H. Hüttel, C. Stirling. *Bisimulation equivalence is decidable for all contex-free processes.* In Proceedings of CONCUR 92, Lecture Notes in Computer Science 630, pages 138-147. Springer-Verlag, 1992.

[8] S. Christensen, Y. Hirsfeld, F. Moller. *Bisimulation equivalence is Decidable for basic parallel processes.* In Proceedings of CONCUR 93, Lecture Notes in Computer Science 715, pages 143-157. Springer-Verlag, 1993.

[9] Y. Hirshfeld. *Deciding equivalences in simple process algebras.* Technical report ECS-LFCS-94-294, Department of Computer Science, University of Edinburgh, 1994.

[10] S. Christensen. *Decidability and Decomposition in Process Algebras.* Ph.D Thesis, University of Edinburgh, 1993.

[11] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets.* Lecture Notes in Computer Science 369. Springer-Verlag, 1989.

[12] M.L. Minsky. *Computation-Finite and Infinite machines.* Prentice-Hall, 1967

[13] R.J. van Glabbeek. *The Linear Time - Branching Time spectrum.* In Proceedings of CONCUR 90, Lecture Notes in Computer Science 458, pages 278-297. Springer-Verlag, 1990.

[14] R.J. van Glabbeek, W.P. Weijland. *Branching time and abstraction in bisimulation semantics (extended abstract).* In G.X. Ritter, editor, Information Processing 89, pages 613-618. North-Holland, 1989.

[15] P. Jančar, F. Moller. *Checking Regular Properties of Petri Nets.* In Proceedings of CONCUR 95, Lecture Notes in Computer Science 962. Springer-Verlag, 1995.

Appendix A

To keep this report self-contained, we present here also the definitions of behavioural equivalences in van Glabbeek hierarchy. We adopt here the definition of the transition system from Section 5 ($T = [N, L, \rightarrow, r]$). If $A \in N$, then $I(A) = \{a \in L \mid \exists B \in N \textit{ such that } A \xrightarrow{a} B\}$ denotes the set of initial actions (labels) of A. Moreover, if M is a set, then $\mathcal{P}(M)$ denotes the power-set of M.

Definition 21 (Trace equivalence)
Let $T \in \mathcal{T}$ be a transition system. We define the set of traces *of T, denoted $tr(T)$, in the following way:*

$$tr(T) = \{w \in L^* \mid \exists A \in N, \textit{ such that } r \xrightarrow{w} A\}$$

Transition systems T_1, T_2 are trace equivalent*, notation $T_1 =_{tr} T_2$, if $tr(T_1) = tr(T_2)$.*

Definition 22 (Completed trace equivalence)
Let $T \in \mathcal{T}$ be a transition system. We define the set of completed traces *of T, denoted $ct(T)$, in the following way:*

$$ct(T) = \{w \in L^* \mid \exists A \in N, \textit{ such that } r \xrightarrow{w} A \wedge A \not\xrightarrow{a} B \textit{ for any } a \in Act, B \in N\}$$

Transition systems T_1, T_2 are completed trace equivalent*, notation $T_1 =_{ct} T_2$, if $tr(T_1) = tr(T_2) \ \wedge \ ct(T_1) = ct(T_2)$.*

Definition 23 (Failure equivalence)
Let $T \in \mathcal{T}$ be a transition system. A pair $[w, \Phi] \in L^* \times \mathcal{P}(L)$ is a failure pair *of T, if there is a node $A \in N$ such that $r \xrightarrow{w} A$ and $I(A) \cap \Phi = \emptyset$. Let $F(T)$ denote the set of all failure pairs of T. Transition systems T_1, T_2 are* failure equivalent*, notation $T_1 =_f T_2$, if $F(T_1) = F(T_2)$*

Definition 24 (Readiness equivalence)
Let $T \in \mathcal{T}$ be a transition system. A pair $[w, \Phi] \in L^* \times \mathcal{P}(L)$ is a ready pair *of T, if there is a node $A \in N$ such that $r \xrightarrow{w} A$ and $I(A) = \Phi$. Let $R(T)$ denote the set of all ready pairs of T. Transition systems T_1, T_2 are* readiness equivalent*, notation $T_1 =_r T_2$, if $R(T_1) = R(T_2)$.*

Definition 25 (Failure trace equivalence)
Let $T \in \mathcal{T}$ be a transition system. The refusal relations $\xrightarrow{\Phi}$ *for $\Phi \in \mathcal{P}(L)$*

are definined by: $A \xrightarrow{\Phi} B$ *iff* $A = B \ \wedge \ I(A) \cap \Phi = \emptyset$. *The* failure trace relations $\xrightarrow{\delta}$ *for* $\delta \in (L \cup \mathcal{P}(L))^*$ *are defined as the reflexive and transitive closure of both the transition and the refusal relations.* $\delta \in (L \cup \mathcal{P}(L))^*$ *is a* failure trace *of* T*, if there is a node* $A \in N$ *such that* $r \xrightarrow{\delta} A$*. Let* $FT(T)$ *denote the set of failure traces of* T*. Transition systems* T_1, T_2 *are* failure trace equivalent*, notation* $T_1 =_{ft} T_2$*, if* $FT(T_1) = FT(T_2)$.

Definition 26 (Ready trace equivalence)
Let $T \in \mathcal{T}$ *be a transition system. The* ready trace relations $\stackrel{\delta}{\Rightarrow}$ *for* $\delta \in (L \cup \mathcal{P}(L))^*$ *are defined inductively by:*

1. $A \stackrel{\epsilon}{\Rightarrow} A$ *for any* $A \in N$.
2. $A \xrightarrow{a} B$ *implies* $A \stackrel{a}{\Rightarrow} B$.
3. $A \stackrel{\Phi}{\Rightarrow} B$ *with* $\Phi \in \mathcal{P}(L)$ *whenever* $A = B$ *and* $I(A) = \Phi$.
4. $A \stackrel{\delta}{\Rightarrow} B \stackrel{\rho}{\Rightarrow} C$ *implies* $A \stackrel{\delta\rho}{\Rightarrow} C$.

$\delta \in (L \cup \mathcal{P}(L))^*$ *is a* ready trace *of* T*, if there is a node* $A \in N$ *such that* $r \stackrel{\delta}{\Rightarrow} A$*. Let* $RT(T)$ *denote the set of ready traces of* T*. Transition systems* T_1, T_2 *are* ready trace equivalent*, notation* $T_1 =_{rt} T_2$*, if* $RT(T_1) = RT(T_2)$.

Definition 27 (Simulation equivalence)
Let $T_1, T_2 \in \mathcal{T}$*. A binary relation* $R \subseteq N_1 \times N_2$ *is a* simulation *if whenever* $A_1 R A_2$ *then*

$$\forall a \in L_1 : \ A_1 R A_2 \wedge A_1 \xrightarrow{a} A_1' \ \Rightarrow \ \exists A_2' : \ A_2 \xrightarrow{a} A_2' \wedge A_1' R A_2'$$

Transition systems T_1, T_2 *are* simulation equivalent*, notation* $T_1 =_s T_2$*, if there exists a simulation* R *with* $r_1 R r_2$ *and a simulation* S *with* $r_2 S r_1$.

Definition 28 (Ready simulation equivalence)
Let $T_1, T_2 \in \mathcal{T}$*. A binary relation* $R \subseteq N_1 \times N_2$ *is a* ready simulation *if whenever* $A_1 R A_2$ *then:*

- $\forall a \in L_1 : \ A_1 R A_2 \wedge A_1 \xrightarrow{a} A_1' \ \Rightarrow \ \exists A_2' : \ A_2 \xrightarrow{a} A_2' \wedge A_1' R A_2'$
- $I(A_1) = I(A_2)$

Transition systems T_1, T_2 *are* ready simulation equivalent*, notation* $T_1 =_{rs} T_2$*, if there exists a ready simulation* R *with* $r_1 R r_2$ *and a ready simulation* S *with* $r_2 S r_1$.

Definition 29 (Possible futures equivalence)
Let $T \in \mathcal{T}$ be a transition system. A pair $[w, \Phi] \in L^* \times \mathcal{P}(L^*)$ is a possible future *of T, if there is a node $B \in N$ such that $A \xrightarrow{w} B$ and $tr(B) = \Phi$. The set of all possible futures of T is denoted $PF(T)$. Transition systems T_1, T_2 are* possible-futures equivalent*, notation $T_1 =_{pf} T_2$, if $PF(T_1) = PF(T_2)$.*

Definition 30 (2-nested simulation equivalence)
Let $T_1, T_2 \in \mathcal{T}$. A binary relation $R \subseteq N_1 \times N_2$ is a 2-nested simulation *if whenever $A_1 R A_2$ then*

- $\forall a \in L_1 : \ A_1 R A_2 \wedge A_1 \xrightarrow{a} A_1' \ \Rightarrow \ \exists A_2' : \ A_2 \xrightarrow{a} A_2' \wedge A_1' R A_2'$
- $A_1 =_s A_2$

Transition systems T_1, T_2 are 2-nested simulation equivalent*, notation $T_1 =_2 T_2$, if there exists a 2-nested simulation R with $r_1 R r_2$ and a 2-nested simulation S with $r_2 S r_1$.*

Recent Publications in the BRICS Report Series

RS-95-52 **Anton´n Kucera.** ***Deciding Regularity in Process Algebras*****. October 1995. 42 pp.**

RS-95-51 **Rowan Davies.** ***A Temporal-Logic Approach to Binding-Time Analysis*****. October 1995. 11 pp.**

RS-95-50 **Dany Breslauer.** ***On Competitive On-Line Paging with Lookahead*****. September 1995. 12 pp.**

RS-95-49 **Mayer Goldberg.** ***Solving Equations in the λ-Calculus using Syntactic Encapsulation*****. September 1995. 13 pp.**

RS-95-48 **Devdatt P. Dubhashi.** ***Simple Proofs of Occupancy Tail Bounds*****. September 1995. 7 pp. To appear in** ***Random Structures and Algorithms*****.**

RS-95-47 **Dany Breslauer.** ***The Suffix Tree of a Tree and Minimizing Sequential Transducers*****. September 1995. 15 pp.**

RS-95-46 **Dany Breslauer, Livio Colussi, and Laura Toniolo.** ***On the Comparison Complexity of the String Prefix-Matching Problem*****. August 1995. 39 pp. Appears in Leeuwen, editor,** ***Algorithms - ESA '94: Second Annual European Symposium proceedings,*** **LNCS 855, 1994, pages 483–494.**

RS-95-45 **Gudmund Skovbjerg Frandsen and Sven Skyum.** ***Dynamic Maintenance of Majority Information in Constant Time per Update*****. August 1995. 9 pp.**

RS-95-44 **Bruno Courcelle and Igor Walukiewicz.** ***Monadic Second-Order Logic, Graphs and Unfoldings of Transition Systems*****. August 1995. 39 pp. To be presented at CSL '95.**

RS-95-43 **Noam Nisan and Avi Wigderson.** ***Lower Bounds on Arithmetic Circuits via Partial Derivatives (Preliminary Version)*****. August 1995. 17 pp. To appear in** ***36th Annual Conference on Foundations of Computer Science,*** **FOCS '95, IEEE, 1995.**

RS-95-42 **Mayer Goldberg.** ***An Adequate Left-Associated Binary Numeral System in the λ-Calculus*****. August 1995. 16 pp.**