Register Machine Based Processes[0]

Jan A. Bergstra[1,2] Alban Ponse[1]

[1] *University of Amsterdam, Programming Research Group*
http://www.wins.uva.nl/research/prog/

[2] *Utrecht University, Department of Philosophy*
http://www.phil.uu.nl/eng/home.html

June 13, 1999

Abstract. We consider the process algebra axiom system ACP extended with two recursive operations: the binary Kleene star $*$, defined by $x^*y = x(x^*y) + y$, and the *push-down* operation \$, defined by $x^\$ y = x((x^\$ y)(x^\$ y)) + y$. In this setting register machine computation can simply be represented, and an equational theory results that is undecidable. We further show that with branching bisimulation equivalence each computable process can be expressed. With rooted τ-bisimulation equivalence almost each semi-computable process can be expressed. Finally we show that all $*$-occurrences can be eliminated by using abstraction and auxiliary actions.

Key words & Phrases: Concurrency, process algebra, iteration, Kleene star, push-down operation, bisimulation equivalence, expressivity, computability.

1 Introduction

In this paper we take as a point of departure the process algebra axiom system $\mathsf{ACP}(A, \gamma)$, i.e., the Algebra of Communicating Processes (defined in [BK84]) with actions in set A and communication function γ. We consider an extension of $\mathsf{ACP}(A, \gamma)$ with two recursive operations:

1. The binary Kleene star $*$ which stems from Kleene [Kle56], in process algebra notation defined by $x^*y = x(x^*y) + y$ (see [BBP94]), and

2. The push-down operation \$, defined by $x^\$ y = x((x^\$ y)(x^\$ y)) + y$ (see [BP98]).

In the framework thus obtained we describe a setting in which recursive functions are computed by register machine based processes. The essential

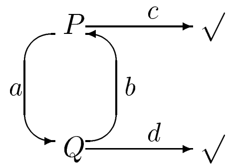
[0] Available at http://adam.wins.uva.nl/~alban/publist/pubs.html.

ingredients of register machine computation, i.e., register machine programs and registers, are modeled by sequential, deterministic processes defined with $*$ and \$. The idea is that the process representing a register machine program invokes a sequence of synchronization actions: each instruction models an update of the register it addresses by a communication action. In detail we set out a particular encoding of register machine computation. As a consequence it follows that the resulting theory (the set of provable consequences) is undecidable.

Though having established a set-up in which all recursive functions can be 'implemented', the resulting setting is not yet sufficiently expressive: even some very simple processes cannot be defined in $\mathsf{ACP}(A,\gamma)$ extended with $*$ and \$. This is for instance the case for the process P recursively defined below with actions a, b, c, d:

$$\begin{array}{rcl} P &=& a \cdot Q + c, \\ Q &=& b \cdot P + d, \end{array}$$

or in a picture where $\surd$ expresses termination:



This lack of expressivity is solved in the second part of the paper (Section 3), where we include *abstraction* as additional feature. We consider two well-known approaches that both yield sufficient expressive power:

- in the setting of *rooted branching bisimilarity* each computable process over a finite alphabet of labels can be expressed;

- with *rooted τ-bisimilarity* each semi-computable process over a finite alphabet of labels that initially is finitely branching can be expressed.

Both these expressivity results are based on a representation of processes by transition systems with states of which the out-degree is at most two.

The paper is ended with some conclusions (Section 4). We added an appendix (Appendix A) on the uniform construction of register machines with two registers.

2 Register Machine Based Processes in $\mathsf{ACP}^{*\$}(A,\gamma)$

In this section we shortly recall the process algebra axiom system $\mathsf{ACP}(A,\gamma)$ and its extension with the recursive operations $*$ and \$. Then we provide a (standard) operational semantics. Next we turn to register machines, and establish a process algebraic representation of register machine programming for a particular repertoire of actions and handshake communications. As a consequence it follows that the resulting theory is undecidable (for only a small number of actions).

2.1 Axiom Systems Up To $\mathsf{ACP}^{*\$}(A, \gamma)$

Let A be a finite set of actions $a, b, \ldots$ and let $\gamma : A \times A \to A$ be a partial function that is *commutative* and *associative*:

$$\begin{array}{ll} \gamma(a,b) = \gamma(b,a) & \text{if } \gamma(a,b)\downarrow \text{(thus } \gamma(a,b) \textit{ defined}), \\ \gamma(a,\gamma(b,c)) = \gamma(\gamma(a,b),c) & \text{if } \gamma(a,\gamma(b,c))\downarrow. \end{array}$$

The function γ defines *communication actions* and models the simultaneous execution of actions. In the case that for all $a, b, c \in A$, $\gamma(a, \gamma(b, c))$ is undefined and that binary communication occurs, we speak of *handshaking* (two-party communication, see [BT84]). The action set A and the communication function γ can be regarded as the parameters of the axiom system $\mathsf{ACP}(A, \gamma)$ defined in [BK84]. The signature of $\mathsf{ACP}(A, \gamma)$ can be summarised as follows:

sorts:	A	(a given, finite set of actions),
	$\mathcal{P}$	(the set of process expressions; $A \subseteq \mathcal{P}$),
operations:	$+ : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(alternative composition or sum),
	$\cdot : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(sequential composition or product),
	$\| : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(parallel composition or merge),
	$⌊\!⌊ : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(left merge),
	$\mid : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(communication merge, given γ),
	$\partial_H : \mathcal{P} \to \mathcal{P}$	(encapsulation, $H \subseteq A$),
constants:	$\delta \in \mathcal{P} \setminus A$	(deadlock or inaction).

In Table 1 the axioms of the system $\mathsf{ACP}(A, \gamma)$ are collected, where a ranges over $A_\delta = A \cup \{\delta\}$. The first five axioms (A1)–(A5) form the core system $\mathsf{BPA}(A)$ (Basic Process Algebra), and adding (A6) and (A7) to $\mathsf{BPA}(A)$ yields $\mathsf{BPA}_\delta(A)$. We take $\cdot$ to be the operation that binds strongest, and $+$ the one that binds weakest. As usual in algebra, we often write xy instead of $x \cdot y$. Furthermore, for $n > 0$ we define x^{n+1} as $x \cdot x^n$, and x^1 as x. For a detailed introduction to $\mathsf{BPA}(A)$ – $\mathsf{ACP}(A, \gamma)$ see e.g. [BW90].

Table 1: The axioms of $\mathsf{ACP}(A,\gamma)$ where $a,b \in A_\delta$, $H \subseteq A$.

(A1)	$x+y$	$=$	$y+x$	
(A2)	$x+(y+z)$	$=$	$(x+y)+z$	
(A3)	$x+x$	$=$	x	
(A4)	$(x+y)z$	$=$	$xz+yz$	
(A5)	$(xy)z$	$=$	$x(yz)$	
(A6)	$x+\delta$	$=$	x	
(A7)	δx	$=$	δ	
(CF1)	$a \mid b$	$=$	$\gamma(a,b)$	if $\gamma(a,b)\downarrow$
(CF2)	$a \mid b$	$=$	δ	otherwise
(CM1)	$x \parallel y$	$=$	$(x \mathbin{\|\!\!\lfloor} y + y \mathbin{\|\!\!\lfloor} x) + x \mid y$	
(CM2)	$a \mathbin{\|\!\!\lfloor} x$	$=$	ax	
(CM3)	$ax \mathbin{\|\!\!\lfloor} y$	$=$	$a(x \parallel y)$	
(CM4)	$(x+y) \mathbin{\|\!\!\lfloor} z$	$=$	$x \mathbin{\|\!\!\lfloor} z + y \mathbin{\|\!\!\lfloor} z$	
(CM5)	$ax \mid b$	$=$	$(a \mid b)x$	
(CM6)	$a \mid bx$	$=$	$(a \mid b)x$	
(CM7)	$ax \mid by$	$=$	$(a \mid b)(x \parallel y)$	
(CM8)	$(x+y) \mid z$	$=$	$x \mid z + y \mid z$	
(CM9)	$x \mid (y+z)$	$=$	$x \mid y + x \mid z$	
(D1)	$\partial_H(a)$	$=$	a	if $a \notin H$
(D2)	$\partial_H(a)$	$=$	δ	if $a \in H$
(D3)	$\partial_H(x+y)$	$=$	$\partial_H(x)+\partial_H(y)$	
(D4)	$\partial_H(xy)$	$=$	$\partial_H(x)\cdot\partial_H(y)$	

Table 2: Additional axioms for binary Kleene star and push-down.

(BKS1)	x^*y	$=$	$x(x^*y)+y$
(BKS2)	$x^*(yz)$	$=$	$(x^*y)z$
(BKS3)	$x^*(y((x+y)^*z)+z)$	$=$	$(x+y)^*z$
(BKS4)	$\partial_H(x^*y)$	$=$	$\partial_H(x)^*\partial_H(y)$
(Push-Down)	$x^\$ y$	$=$	$x((x^\$ y)(x^\$ y))+y$

The binary Kleene star was added to process algebra in [BBP94], with axioms given in Table 2. In [BP98] we defined the recursive operation *push-down*, notation \$, by the single axiom also displayed in Table 2. Extension of one of the systems mentioned above with $*$ or \$ and relevant axioms is denoted by adding the appropriate symbol as a superscript, e.g. $\mathsf{BPA}^*(A)$ stands for $\mathsf{BPA}(A)$ extended with $*$ and the axioms (BKS1)–(BKS3), and

$$\mathsf{ACP}^{*\$}(A,\gamma)$$

stands for $\mathsf{ACP}(A,\gamma)$ extended with $*$ and \$, the axioms (BKS1)–(BKS4) and the defining axiom for \$. In the process algebra systems without recursive operations mentioned above, one can prove that $P \parallel Q = Q \parallel P$ and $P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$ for closed process expressions P, Q, R by structural induction. This property is lost if $*$ and/or \$ are included. However, since it is convenient to omit parentheses in (large) $\parallel$-expressions we further adopt the notational convention that $\parallel$ associates to the right, thus

$$x \parallel y \parallel z = x \parallel (y \parallel z).$$

2.2 Transitions and Bisimulation Equivalence

We relate processes to transition systems and define bisimulation equivalence between transition systems. Then we provide bisimulation equivalence models for the process algebra systems introduced in the previous section.

A *transition system* is a tuple $\langle S, L, \longrightarrow, s\rangle$, where S is a set of *states*, L is a set of *labels*, $\longrightarrow$ is a *transition relation*, and $s \in S$ is the *initial state*. Consider one of the process algebra axiom systems $\mathsf{BPA}(A)$ – $\mathsf{ACP}^{*\$}(A,\gamma)$, and let $\mathcal{P}$ represent all process expressions given by its signature. In order to associate transition systems with elements of $\mathcal{P}$, we take $\mathcal{P}$ itself as the set of states. As labels we take the atomic actions from A. The transition relation $\longrightarrow$ contains transitions

$$\_ \xrightarrow{\_} \_ \subseteq \mathcal{P} \times A \times \mathcal{P},$$

and for modeling *(successful) termination*, special transitions

$$\_ \xrightarrow{\_} \surd \subseteq \mathcal{P} \times A.$$

($\surd$ is pronounced "tick".)

The idea is that for $a \in A$, a transition $P \xrightarrow{a} P'$ expresses that by executing a, the process represented by P can evolve into the remainder process represented by P'. The transition $P \xrightarrow{a} \surd$ expresses that the process represented by P can terminate (successfully) after executing a. The rules in Table 3 define the transition relation $\longrightarrow$, where the signature and parameters of $\mathcal{P}$ (possibly including a communication function γ) determine which rules are appropriate. Note that the state δ has no outgoing transitions. Finally, the transition system related to a process expression P has P itself as initial state. If $\mathcal{P}$ is fixed and no confusion can arise, we often write P for the transition system $\langle \mathcal{P}, L, \longrightarrow, P \rangle$.

Definition 2.2.1. A *bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{P}$ that satisfies

- $P\mathcal{R}Q$ and $P \xrightarrow{a} P'$ implies $\exists Q'.\ Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$,
- $P\mathcal{R}Q$ and $Q \xrightarrow{a} Q'$ implies $\exists P'.\ P \xrightarrow{a} P'$ and $P'\mathcal{R}Q'$,
- $P\mathcal{R}Q$ implies $(P \xrightarrow{a} \surd \iff Q \xrightarrow{a} \surd)$.

Two states P, Q are *bisimilar*, notation $P \leftrightarrows Q$, if there exists a bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$. $\square$

Note that $\leftrightarrows$ is an equivalence relation. Now if we take $\mathcal{P}$ as the set of closed process expressions over $\mathsf{ACP}^{*\$}(A, \gamma)$, it follows that $\leftrightarrows$ is a congruence relation for all operations involved [BV93, GV92]. We write $\mathsf{ACP}^{*\$}(A, \gamma)/_{\leftrightarrows} \models P = Q$ whenever $P \leftrightarrows Q$ according to the notions just defined, and for variable sequence $\vec{x} = x_1, ..., x_n$

$$\mathsf{ACP}^{*\$}(A, \gamma)/_{\leftrightarrows} \models t_1(\vec{x}) = t_2(\vec{x})$$

if for all $\vec{P} = P_1, ..., P_n$ it holds that $t_1(\vec{P}) = t_2(\vec{P})$. It is not difficult, but tedious to establish that in the bisimulation model thus obtained all equations of Tables 1 and 2 are true. So we have the following result.

Table 3: Transition rules for $\mathsf{BPA}(A) - \mathsf{ACP}^{*\$}(A,\gamma)$ where $a,b \in A,\ H \subseteq A$.

$$a \xrightarrow{a} \surd, \quad a \in A$$

$$\frac{x \xrightarrow{a} \surd}{\begin{array}{l} x+y \xrightarrow{a} \surd \\ y+x \xrightarrow{a} \surd \\ x\cdot y \xrightarrow{a} y \\ x \parallel y \xrightarrow{a} y \\ y \parallel x \xrightarrow{a} y \\ x \,⌊\!⌊\, y \xrightarrow{a} y \\ \partial_H(x) \xrightarrow{a} \surd \text{ if } a \notin H \\ x^* y \xrightarrow{a} x^* y \\ x^\$ y \xrightarrow{a} (x^\$ y)(x^\$ y) \end{array}} \qquad \frac{x \xrightarrow{a} x'}{\begin{array}{l} x+y \xrightarrow{a} x' \\ y+x \xrightarrow{a} x' \\ x\cdot y \xrightarrow{a} x'\cdot y \\ x \parallel y \xrightarrow{a} x' \parallel y \\ y \parallel x \xrightarrow{a} y \parallel x' \\ x \,⌊\!⌊\, y \xrightarrow{a} x' \parallel y \\ \partial_H(x) \xrightarrow{a} \partial_H(x') \text{ if } a \notin H \\ x^* y \xrightarrow{a} x'(x^* y) \\ x^\$ y \xrightarrow{a} x'((x^\$ y)(x^\$ y)) \end{array}}$$

$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd}{\begin{array}{l} x \parallel y \xrightarrow{\gamma(a,b)} \surd \text{ if } \gamma(a,b)\downarrow \\ x \mid y \xrightarrow{\gamma(a,b)} \surd \text{ if } \gamma(a,b)\downarrow \end{array}} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{\begin{array}{l} x \parallel y \xrightarrow{\gamma(a,b)} x' \parallel y' \text{ if } \gamma(a,b)\downarrow \\ x \mid y \xrightarrow{\gamma(a,b)} x' \parallel y' \text{ if } \gamma(a,b)\downarrow \end{array}}$$

$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{\begin{array}{l} x \parallel y \xrightarrow{\gamma(a,b)} y' \text{ if } \gamma(a,b)\downarrow \\ x \mid y \xrightarrow{\gamma(a,b)} y' \text{ if } \gamma(a,b)\downarrow \end{array}} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd}{\begin{array}{l} x \parallel y \xrightarrow{\gamma(a,b)} x' \text{ if } \gamma(a,b)\downarrow \\ x \mid y \xrightarrow{\gamma(a,b)} x' \text{ if } \gamma(a,b)\downarrow \end{array}}$$

Lemma 2.2.2. *The system* $\mathsf{ACP}^{*\$}(A,\gamma)$ *is sound with respect to bisimulation: if* $\mathsf{ACP}^{*\$}(A,\gamma) \vdash t_1(\vec{X}) = t_2(\vec{X})$*, then*

$$\mathsf{ACP}^{*\$}(A,\gamma)/_{\leftrightarrows} \models t_1(\vec{X}) = t_2(\vec{X}).$$

Finally, the axioms of $\mathsf{ACP}(A,\gamma)$ completely characterize bisimilarity between the processes that can be expressed [BK84]. Moreover, bisimilarity over $\mathsf{BPA}^*(A)$ is completely axiomatized by the axioms of $\mathsf{BPA}(A)$ (A1–A5) and (BKS1)–(BKS3), as was first proved in [FZ94].

2.3 Alphabets, Registers, and Register Processes

In this section we define a setting in which registers are straightforwardly modeled as processes in $\mathsf{ACP}^{*\$}(A,\gamma)$ for a particular choice of A and γ, and introduce some notation. For $i \in \mathbb{N}$, let

$$\alpha_i = \{a_i, s_i, z_i, c_i\}$$

contain atomic actions representing instructions for a register named i.

The register is modeled as a process R_i over alphabet

$$\overline{\alpha}_i = \{\overline{a}_i, \overline{s}_i, \overline{z}_i, \overline{c}_i\}$$

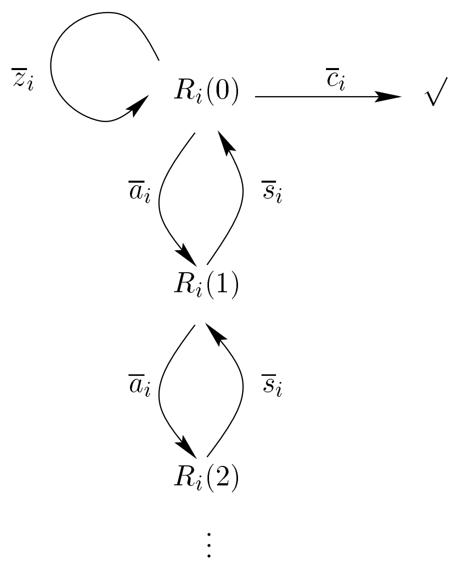
as follows:

$$R_i = (\overline{a}_i(\overline{a}_i{}^\$ \overline{s}_i) + \overline{z}_i)^* \overline{c}_i.$$

For $j \in \mathbb{N}$ we define the following abbreviations:

$$\begin{array}{lcl} R_i(0) & = & R_i, \\ R_i(j+1) & = & (\overline{a}_i{}^\$ \overline{s}_i) \cdot R_i(j). \end{array}$$

So $R_i(j)$ represents register i containing value j. The behaviour of R_i is visualized alongside:



We shall often consider a number of registers operating in parallel (a register *machine*). Therefore we introduce for $k > 0$ the abbreviation

$$\mathcal{R}_k \text{ defined by } \mathcal{R}_1 = R_0,\ \mathcal{R}_{n+2} = R_{n+1} \parallel \mathcal{R}_{n+1}$$

So $\mathcal{R}_k$ represents k empty registers $k-1,...,1,0$ in parallel. (Recall that $\|$ associates to the right.) In the case that each of these contains value n_i, we use the notation

$$\mathcal{R}_k(n_{k-1},...,n_0).$$

Let B be a finite set of atomic actions not containing t and disjoint from α_i, $\overline{\alpha}_i$. We define the following sets of atomic actions:

$$\begin{aligned} B_t &= B \cup \{t\}, \\ A_n(B) &= B_t \cup H_n, \\ H_n &= \bigcup_{j<n} (\alpha_j \cup \overline{\alpha}_j). \end{aligned}$$

Communication on $A_n(B)$ is defined by $\gamma(a,b) \neq \delta$ if and only if for some $j<n$, either $a \in \alpha_j, b \in \overline{\alpha}_j$ and $b=\overline{a}$, or $b \in \alpha_j, a \in \overline{\alpha}_j$ and $a=\overline{b}$. In both cases, $\gamma(a,b)=t$.

As it turns out, $\mathsf{ACP}^{*\$}(A_n(B),\gamma)$ will allow a practical encoding of register machine programming. All phenomena known to us and connected with the option to encode computability in general become visible with $n \geq 4$. The cases $n=1,2,3$ may feature various anomalies due to a lack of expressive power, and are of no concern to us.

2.4 Register Machine Programming in $\mathsf{ACP}^{*\$}(A,\gamma)$

We first define a class of process expressions representing structured register machine programs. Such a process will be synchronized with a register machine process, i.e., a number of register processes in parallel, representing all registers to which the program makes reference to.

Definition 2.4.1. For $W \subseteq V \subseteq \mathbb{N}$ and $\emptyset \subseteq C \subseteq B_t$ the Register Machine Iterative Programs using registers in V and retaining registers in $W \subseteq V$, notation

$$\mathsf{RMIP}(C,V,W),$$

is a collection of expressions with alphabet in $C \cup \bigcup_{j\in V} \alpha_j$. Using the alternative notation

$$P :_{rmi} V \to W,$$

for such a program P, its syntax is inductively defined by the following clauses:

1. $a_i :_{rmi} V \cup \{i\} \to V \cup \{i\}$

2. $\dfrac{P :_{rmi} V \to W \quad Q :_{rmi} W \to U}{PQ :_{rmi} V \to U}$

3. $s_i{}^* c_i :_{rmi} V \cup \{i\} \to V \setminus \{i\}$

4. $s_i{}^* z_i :_{rmi} V \cup \{i\} \to V \cup \{i\}$

5. $\dfrac{P :_{rmi} V \cup \{i\} \to V \cup \{i\}}{\begin{array}{l}(s_iP)^* z_i :_{rmi} V \cup \{i\} \to V \cup \{i\} \\ (z_iP)^* s_i :_{rmi} V \cup \{i\} \to V \cup \{i\}\end{array}}$

6. $\dfrac{P, Q :_{rmi} V \cup \{i\} \to W \quad R :_{rmi} V \setminus \{i\} \to W}{\begin{array}{c}s_iP + z_iQ :_{rmi} V \cup \{i\} \to W \\ s_iP + c_iR :_{rmi} V \cup \{i\} \to W\end{array}}$

7. $a :_{rmi} V \to V$ for all $a \in C$

8. $\dfrac{P, Q :_{rmi} V \to W \quad a, b \in C}{aP + bQ :_{rmi} V \to W}$

Instead of $\mathsf{RMIP}(C, \{n \in \mathbb{N} \mid n < i\}, \{n \in \mathbb{N} \mid n < j\})$ we further write $\mathsf{RMIP}(C, i, j)$. □

In the sequel we will often consider expressions of the form

$$\partial_{H_k}(Px \parallel \mathcal{R}_k(m_{k-1}, ..., m_0))$$

abbreviating $\partial_{H_k}(Px \parallel (R_{k-1}(m_{k-1}) \parallel ...(R_1(m_1) \parallel R_0(m_0))...))$ with P in $\mathsf{RMIP}(C, k, k)$ and x a process variable. Note that in the case that $P \in \mathsf{RMIP}(\emptyset, k, k)$ such an expression specifies a deterministic process; a non-deterministic process can only be defined with use of the actions in B_t. The following basic result states that we can 'implement' any computable function in $\mathsf{ACP}^{*\$}(A_4(\emptyset), \gamma)$.

Lemma 2.4.2. *Let $f : \mathbb{N} \to \mathbb{N}$ be computable (not necessarily total). There exists $P \in \mathsf{RMIP}(\emptyset, 4, 4)$ and computable $g : \mathbb{N} \to \mathbb{N}$ such that if $f(n)$ is defined, then $g(n)$ is defined and*

$$\begin{array}{l}\mathsf{ACP}^{*\$}(A_4(\emptyset), \gamma) \vdash \\ \partial_{H_4}(Px \parallel \mathcal{R}_4(0, 0, 0, n)) = t^{g(n)} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0, 0, 0, f(n))),\end{array}$$

and if $f(n)$ is not defined, then $g(n)$ is not defined and for each $i \in \mathbb{N} \setminus \{0\}$ there exists a process term M such that

$$\mathsf{ACP}^{*\$}(A_4(\emptyset), \gamma) \vdash \partial_{H_4}(Px \parallel \mathcal{R}_4(0,0,0,n)) = t^i \cdot M.$$

Proof. Let $\overline{P}$ be a register machine program which computes f using three registers and a set of instructions numbered $1, ..., k$ of the following form:

halt	halt,
(a_i, l)	add 1 to register i and go to instruction l,
(s_i, l, l')	if register i contains a value larger than 0, substract 1 from register i and go to instruction l, otherwise go to l'.

(This is possible, see e.g. [Min67] or Appendix A.) We turn $\overline{P}$ into a process P in $\mathsf{RMIP}(\emptyset, 4, 4)$ by using a register process R_3 to store 'the next instruction number', and taking

$$P = P_1' P', \quad P' = (s_3 P_1)^* z_3$$

where

$$P_m \quad = \quad \begin{cases} P_m' & \text{if } k = 1, \\ s_3 P_{m+1} + z_3 P_m' & \text{if } k > 1 \text{ and } m+1 < k, \\ s_3 P_{m+1}' + z_3 P_m' & \text{if } k > 1 \text{ and } m+1 = k, \end{cases}$$

$$P_m' \quad = \quad \begin{cases} (s_1{}^* z_1)(s_2{}^* z_2)(s_3{}^* z_3) & \text{if } \mathit{line}(m, \overline{P}) = \mathsf{halt}, \\ a_i \cdot a_3^l & \text{if } \mathit{line}(m, \overline{P}) = (a_i, l), \\ s_i \cdot a_3^l + z_i \cdot a_3^{l'} & \text{if } \mathit{line}(m, \overline{P}) = (s_i, l, l'). \end{cases}$$

By inspection it follows that starting from the initial state

$$\partial_{H_4}(P_1' P' x \parallel R_3 \parallel R_2 \parallel R_1 \parallel R_0(n)), \qquad \text{i.e., } \partial_{H_4}(P_1' P' x \parallel \mathcal{R}_4(0,0,0,n)),$$

each 'program state' $P_m' P' x$, i.e., the state that models execution of instruction m, can only occur in one of the following four patterns:

$$\partial_{H_4}(P_m' P' x \parallel R_3 \parallel R_2(m_2) \parallel R_1(m_1) \parallel R_0(m_0)),$$
$$\partial_{H_4}(P_m' P' x \parallel R_3 \parallel R_2(m_2) \parallel R_0(m_0) \parallel R_1(m_1)),$$
$$\partial_{H_4}(P_m' P' x \parallel R_3 \parallel ((R_0(m_0) \parallel R_1(m_1)) \parallel R_2(m_2))),$$
$$\partial_{H_4}(P_m' P' x \parallel R_3 \parallel ((R_1(m_1) \parallel R_0(m_0)) \parallel R_2(m_2))).$$

This can easily be seen by considering the three instructions $(a_0, 1)$, $(a_1, 1)$ and $(a_2, 1)$: in none of the four cases a new pattern arises. Furthermore, in the case the P_m' above models halt, each of these patterns reduces to the desired form

$$\partial_{H_4}(x \parallel \mathcal{R}_4(0,0,0,m_0)),$$

by $m_1 + m_2 + 4$ t-steps. We show this for the first pattern:

$$\begin{array}{l}\partial_{H_4}((s_1{}^* z_1)(s_2{}^* z_2)(s_3{}^* z_3)P'x \parallel \mathcal{R}_4(0, m_2, m_1, m_0))\\ \quad = \partial_{H_4}((s_1{}^* z_1)(s_2{}^* z_2)(s_3{}^* z_3)P'x \parallel R_3 \parallel R_2(m_2) \parallel R_1(m_1) \parallel R_0(m_0))\\ \quad = t^{m_1+1} \cdot \partial_{H_4}((s_2{}^* z_2)(s_3{}^* z_3)P'x \parallel (((R_1 \parallel R_0(m_0)) \parallel R_2(m_2)) \parallel R_3))\\ \quad = t^{m_1+m_2+2} \cdot \partial_{H_4}((s_3{}^* z_3)P'x \parallel ((R_2 \parallel (R_1 \parallel R_0(m_0))) \parallel R_3))\\ \quad = t^{m_1+m_2+3} \cdot \partial_{H_4}(P'x \parallel R_3 \parallel R_2 \parallel R_1 \parallel R_0(m_0))\\ \quad = t^{m_1+m_2+4} \cdot \partial_{H_4}(x \parallel R_3 \parallel R_2 \parallel R_1 \parallel R_0(m_0))\\ \quad = t^{m_1+m_2+4} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0,0,0,m_0)).\end{array}$$

It can be easily checked that also the remaining three patterns equal $t^{m_1+m_2+4} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0,0,0,m_0))$.

Therefore, analyzing $\partial_{H_4}(Px \parallel \mathcal{R}_4(0,0,0,n))$ in terms of these patterns allows the assumption that $\parallel$ is associative and commutative in this case. To this end we set

$$\mathsf{Cf}(m, x_2, x_1, x_0) = \partial_{H_4}(P'_m P'x \parallel \mathcal{R}_4(0, x_2, x_1, x_0))$$

for the configuration in which the register machine has to execute instruction $\mathit{line}(m, \overline{P})$ while register 3 is empty and for $i \leq 2$, register i contains value x_i. Note that

$$\begin{aligned}\partial_{H_4}(Px \parallel \mathcal{R}_4(0,0,0,n)) &= \partial_{H_4}(P'_1 P'x \parallel \mathcal{R}_4(0,0,0,n))\\ &= \mathsf{Cf}(1,0,0,n).\end{aligned}$$

In order to show that $\mathsf{Cf}(m, x_2, x_1, x_0)$ yields a faithful modeling of the register machine which has to execute instruction m on machine configuration $\langle x_2, x_1, x_0\rangle$ we must consider the effect of instructions of the form (a_i, l) and (a_i, l, l').

In case that $\mathit{line}(m, \overline{P}) = (a_i, l)$ (recall $0 < l \leq k$) and a_i modifies the machine configuration $\langle x_2, x_1, x_0\rangle$ into $\langle y_2, y_1, y_0\rangle$ where $y_i = x_i + 1$ and $y_j = x_j$ for $j \neq i$, we derive

$$\begin{aligned}\mathsf{Cf}(m, x_2, x_1, x_0) &= \partial_{H_4}(P'_m P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\ &= \partial_{H_4}(a_i a_3^l P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\ &= t \cdot \partial_{H_4}(a_3^l P'x \parallel R_3 \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\ &= t^{l+1} \cdot \partial_{H_4}(P'x \parallel R_3(l) \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\ &= t^{2l+2} \cdot \partial_{H_4}(P'_l P'x \parallel R_3 \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\ &= t^{2l+2} \cdot \mathsf{Cf}(l, y_2, y_1, y_0).\end{aligned}$$

If $\mathit{line}(m, \overline{P}) = (s_i, l, l')$ there are two cases to distinguish depending on the value x_i of the current machine configuration $\langle x_2, x_1, x_0\rangle$. If $x_i = 0$, then

$$\begin{aligned}
\mathsf{Cf}(m,x_2,x_1,x_0) &= \partial_{H_4}(P'_m P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= \partial_{H_4}((s_i a_3^l + z_i a_3^{l'})P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= t \cdot \partial_{H_4}(a_3^{l'} P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= t^{l'+1} \cdot \partial_{H_4}(P'x \parallel R_3(l') \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= t^{2l'+2} \cdot \partial_{H_4}(P'_{l'} P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= t^{2l'+2} \cdot \mathsf{Cf}(l',x_2,x_1,x_0).
\end{aligned}$$

If $x_i > 0$ and s_i modifies the machine configuration $\langle x_2, x_1, x_0\rangle$ into $\langle y_2, y_1, y_0\rangle$ where $y_i = x_i - 1$ and $y_j = x_j$ for $j \neq i$, then

$$\begin{aligned}
\mathsf{Cf}(m,x_2,x_1,x_0) &= \partial_{H_4}(P'_m P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= \partial_{H_4}((s_i a_3^l + z_i a_3^{l'})P'x \parallel R_3 \parallel R_2(x_2) \parallel R_1(x_1) \parallel R_0(x_0))\\
&= t \cdot \partial_{H_4}(a_3^{l} P'x \parallel R_3 \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\
&= t^{l+1} \cdot \partial_{H_4}(P'x \parallel R_3(l) \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\
&= t^{2l+2} \cdot \partial_{H_4}(P'_{l} P'x \parallel R_3 \parallel R_2(y_2) \parallel R_1(y_1) \parallel R_0(y_0))\\
&= t^{2l+2} \cdot \mathsf{Cf}(l,y_2,y_1,y_0).
\end{aligned}$$

Clearly, in terms of machine configurations decorated with reference to the next instruction, i.e., in terms of $\mathsf{Cf}(m,x_2,x_1,x_0)$ the above provides a faithful modeling of computation of the register machine program $\overline{P}$. Now let functions $F_m(x_2,x_1,x_0)$ for $m = 1,...,k$ be defined as follows:

- If $P'_m = (s_1{}^*z_1)(s_2{}^*z_2)(s_3{}^*z_3)$, the encoding of halt, then $F_m(x_2,x_1,x_0) = x_2 + x_1 + 4$,
- If $P'_m = a_i a_3^l$, the encoding of instruction (a_i, l), then $F_m(x_2,x_1,x_0) = F_l(y_2,y_1,y_0) + 2l + 2$, where $y_i = x_i + 1$ and $y_j = x_j$ for $j \neq i$,
- If $P'_m = s_i a_3^l + z_i a_3^{l'}$, the encoding of instruction (s_i, l, l'), then

$$F_m(x_2,x_1,x_0) = \begin{cases} F_l(y_2,y_1,y_0) + 2l + 2 & \text{if } x_i > 0,\ y_i = x_i - 1 \text{ and}\\ & y_j = x_j \text{ for } j \neq i,\\ F_{l'}(x_2,x_1,x_0) + 2l' + 2 & \text{otherwise.}\end{cases}$$

Observe that the F_m are not necessarily *total*, even if f is. However, if $\overline{P}$ started at line m with register values $m2, m1, n$ computes to halt with register values $m1', m2', n'$ then

$$\begin{aligned}
&\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash\\
&\partial_{H_4}(P'_m P'x \parallel \mathcal{R}_4(0,m2,m1,n)) = t^{F_m(m1,m2,n)} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0,0,0,n')).
\end{aligned}$$

This follows by induction on the length of (terminating) computations, say h. If $h = 0$ then $\mathit{line}(m,\overline{P}) = \mathsf{halt}$ and $P'_m = (s_1{}^*z_1)(s_2{}^*z_2)(s_3{}^*z_3)$. If $h = h' + 1$, then P'_m is either $a_i a_3^l$ or $s_i a_3^l + z_i a_3^{l'}$ for some i, l, l'. The identities above suffice to make the induction step.

Taking $m = 1$ and $m1 = m2 = 0$ yields the required information: $g(n) = F_1(0,0,n)$. Clearly, $g(n)$ is only defined if $f(n)$ is.

In the case that $\overline{P}$ started at line m with certain register values does not halt, thus performance of successive instructions is perpetual, the second statement of the lemma follows immediately. □

Corollary 2.4.3. *Let $f_0, f_1, ..., f_m : \mathbb{N} \to \mathbb{N}$ be computable functions (not necessarily total). Then there exists $P \in \mathsf{RMIP}(\emptyset, m+4, m+4)$ such that for some partially computable $g : \mathbb{N} \to \mathbb{N}$ and for all n,*

- *if $f_0(n), ..., f_m(n)$ are defined, then so is $g(n)$ and*

$$\begin{aligned}&\mathsf{ACP}^{*\$}(A_{m+4}(\emptyset), \gamma) \vdash \\ &\qquad \partial_{H_{m+4}}(Px \parallel \mathcal{R}_{m+4}(0, ..., 0, n)) \quad = \\ &\qquad t^{g(n)} \cdot \partial_{H_{m+4}}(x \parallel \mathcal{R}_{m+4}(0,0,0, f_m(n), ..., f_0(n)))\,,\end{aligned}$$

- *otherwise $g(n)$ is not defined and for all $i \in \mathbb{N} \setminus \{0\}$ there exists a process expression M such that*

$$\mathsf{ACP}^{*\$}(A_{m+4}(\emptyset), \gamma) \vdash \partial_{H_{m+4}}(Px \parallel \mathcal{R}_{m+4}(0, ..., 0, n)) = t^i \cdot M.$$

Proof. Without proof we claim the existence of $Q \in \mathsf{RMIP}(\emptyset, m+4, m+4)$ such that

$$\begin{aligned}&\mathsf{ACP}^{*\$}(A_{m+4}(\emptyset), \gamma) \vdash \\ &\partial_{H_{m+4}}(Qx \parallel \mathcal{R}_{m+4}(0, ..., 0, n)) = t^k \cdot \partial_{H_{m+4}}(x \parallel \mathcal{R}_{m+4}(0,0,0,n, ..., n)))\end{aligned}$$

for some value k. So the argument n can be copied into register processes $R_1, ..., R_m$ according to the associative pattern indicated above.

Now function $f_j(n)$ can be computed from argument register $R_j(n)$ with auxiliary registers R_{m+3}, R_{m+2}, R_{m+1} and some register program process P_j, as was shown in the previous proof. Without loss of generality we may assume that each of the P_j affects process R_j: otherwise, increase the instruction numbers and references of register programme $\overline{P}_j$ by 2, and use the equivalent program

$$\begin{aligned}1:&\quad (a_j, 2)\\ 2:&\quad (s_j, 3, 3)\\ 3:&\quad ...(\overline{P}_j\text{'s increased instructions})\end{aligned}$$

This guarantees that $\partial_{H_{m+4}}(P_0 P_1 ... P_m x \parallel \mathcal{R}_{m+4}(0,0,0,n,...,n)) = \partial_{H_{m+4}}(x \parallel \mathcal{R}_{m+4}(0,0,0, f_m(n), ..., f_0(n)))$ in case of termination, where the right-hand side indeed has the indicated bracket pattern. So we find that $P = QP_0P_1...P_m$ does the job. □

2.5 Undecidability Results

In this section we show that $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma)$ has an undecidable theory. We provide a family of terms u_n, v_n such that $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash u_n = v_n$ is not decidable.

Theorem 2.5.1. $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash u = v$ *need not be decidable.*

Proof. Let W_{e_1}, W_{e_2} be recursively inseparable sets. Let $f : \mathbb{N} \to \mathbb{N}$ be the partial recursive function defined by

$$f(n) = \begin{cases} 0 & \text{if } n \in W_{e_1}, \\ 1 & \text{if } n \in W_{e_2}, \\ \uparrow & \text{otherwise.} \end{cases}$$

By Lemma 2.4.2 we can choose $P \in \mathsf{RMIP}(\emptyset, 4, 4)$ such that if $f(n)$ is defined, then there is a function g satisfying

$$\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash \partial_{H_4}(Px \parallel \mathcal{R}_4(0,0,0,n)) = t^{g(n)} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0,0,0,f(n))),$$

and if $f(n)$ is not defined, then for all m there exists M with

$$\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash \partial_{H_4}(Px \parallel \mathcal{R}_4(0,0,0,n)) = t^m \cdot M.$$

Now let $P_1', P_2' \in \mathsf{RMIP}(\emptyset, 4, 0)$ and $P_1(n), P_2(n)$ be defined by

$$\begin{aligned} P_1' &= P(s_3{}^*c_3)(s_2{}^*c_2)(s_1{}^*c_1)(s_0{}^*c_0), \\ P_2' &= P(s_3{}^*c_3)(s_2{}^*c_2)(s_1{}^*c_1)((s_0(s_0{}^*z_0))^*c_0), \\ P_i(n) &= \partial_{H_4}(P_i' \parallel \mathcal{R}_4(0,0,0,n)). \end{aligned}$$

Then, writing $\vdash u = v$ for $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash u = v$, we find

$$\begin{array}{llllll} n \in W_{e_1} & \Rightarrow & f(n) = 0 & \Rightarrow & \vdash P_1(n) = P_2(n) & (= t^{g(n)+4}), \\ n \notin W_{e_1} \cup W_{e_2} & \Rightarrow & f(n) = \uparrow & \Rightarrow & \vdash P_1(n) = P_2(n) & \\ & & & & & \text{(an infinite } t\text{-sequence)}, \\ n \in W_{e_2} & \Rightarrow & f(n) = 1 & \Rightarrow & \not\vdash P_1(n) = P_2(n) & (t^{g(n)+5} \neq t^{g(n)+6}). \end{array}$$

As to the latter implication: assume otherwise, i.e., $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash t^k = t^{k+1}$ for some $k > 0$. Then by Lemma 2.2.2, $t^k \leftrightarrow t^{k+1}$, which clearly is a contradiction.

Thus, decidability of $\mathsf{ACP}^{*\$}(A_4(\emptyset),\gamma) \vdash P_1(n) = P_2(n)$ provides a recursive separation of W_{e_1} and W_{e_2}, which is contradictory. □

To be a little more general, we call a model $\mathcal{M}$ for $\mathsf{ACP}^{*\$}(A_4(B),\gamma)$ *left cancellating* or *trace consistent* if for all $a,b\in A_4(B)$,

$$\mathcal{M}\models \forall x,y(ax=by\rightarrow(a=b\wedge x=y)).$$

Then the following corollary follows immediately from the proof above.

Corollary 2.5.2. *Let $\mathcal{M}$ be a model of $\mathsf{ACP}^{*\$}(A_4(B),\gamma)$ that is left cancellating. Then the word problem of $\mathcal{M}$ is undecidable.*

3 Adding Abstraction

A basic ingredient of concurrency theory is the *silent* or *internal* action or *hidden move*, notation τ, dating back to Milner [Mil80]. We consider two combinations of the constant τ and $\mathsf{ACP}(A,\gamma)$, each of which goes with an operation that renames actions into τ, i.e., that defines the distinction between what is observable and what is not. This facility, known as *abstraction* or *hiding*, is a common feature in process algebra. It serves both verification styles and expressive power. In this section we show that the addition of abstraction yields sufficient expressive power. Furthermore, we show that the use of $*$ in all our results can be avoided.

3.1 Axiom Systems and Operational Semantics

In [BK85] the system $\mathsf{ACP}_\tau(A,\gamma)$ is defined. This involves the constant τ and the abstraction operator $\tau_I(\_)$ which renames the actions in $I\subseteq A$ into τ. The axioms of $\mathsf{ACP}_\tau(A,\gamma)$ are those of $\mathsf{ACP}(A,\gamma)$ extended with the axioms in Table 4. More recently another extension of $\mathsf{ACP}(A,\gamma)$ with abstraction was defined: the system $\mathsf{ACP}^\tau(A,\gamma)$ axiomatizing *rooted branching bisimilarity* (see [GW96]). Its additional axioms are given in Table 5. Note that in this case the a ranges over $A_{\delta\tau}=A_\delta\cup\{\tau\}$. For a detailed introduction to these process algebra systems see e.g. [BW90].

When the operation $*$ is added, there is an extra axiom for binary Kleene star:

$$\text{(BKS5)}\ \tau_I(x^*y)=\tau_I(x)^*\tau_I(y).$$

Extension of one of the systems mentioned above with $*$ or \$ and relevant axioms is denoted by adding the appropriate symbol as a superscript, e.g. $\mathsf{ACP}^{\tau*\$}(A,\gamma)$ stands for $\mathsf{ACP}^\tau(A,\gamma)$ extended with $*$ and \$, the axioms (BKS1)–(BKS5) and the defining axiom for \$.

Table 4: The additional axioms of $\mathsf{ACP}_\tau(A,\gamma)$ where $I \subseteq A$ and $a, b \in A_\delta$.

(T1)	$x\tau = x$				
(T2)	$\tau x + x = \tau x$				
(T3)	$a(\tau x + y) = a(\tau x + y) + ax$				
			(TI0)	$\tau_I(\tau) = \tau$	
(TC1)	$\tau \mid x = \delta$		(TI1)	$\tau_I(a) = a$	if $a \notin I$
(TC2)	$x \mid \tau = \delta$		(TI2)	$\tau_I(a) = \tau$	if $a \in I$
(TC3)	$\tau x \mid y = x \mid y$		(TI3)	$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$	
(TC4)	$x \mid \tau y = x \mid y$		(TI4)	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	

Table 5: The additional axioms of $\mathsf{ACP}^\tau(A,\gamma)$ where $I \subseteq A$ and $a \in A_{\delta\tau}$.

(B1)	$x\tau = x$	(TI1)	$\tau_I(a) = a$	if $a \notin I$
(B2)	$x(\tau(y+z)+y) = x(y+z)$	(TI2)	$\tau_I(a) = \tau$	if $a \in I$
		(TI3)	$\tau_I(x+y) = \tau_I(x) + \tau_I(y)$	
		(TI4)	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	

We define a structural operational semantics for the systems involving τ and τ_I. The transition relation is defined by the axioms and rules in Table 3 and those in Table 6.

Table 6: Additional transition rules for $\mathsf{ACP}^\tau(A,\gamma)$ where $a \in A_\tau$.

$$\tau \xrightarrow{\tau} \surd \qquad \frac{x \xrightarrow{a} \surd}{\begin{array}{l}\tau_I(x) \xrightarrow{a} \surd \;\text{ if } a \notin I\\ \tau_I(x) \xrightarrow{\tau} \surd \;\text{ if } a \notin I\end{array}} \qquad \frac{x \xrightarrow{a} x'}{\begin{array}{l}\tau_I(x) \xrightarrow{a} \tau_I(x') \;\text{ if } a \notin I\\ \tau_I(x) \xrightarrow{\tau} \tau_I(x') \;\text{ if } a \notin I\end{array}}$$

As to express general expressivity results, we define equivalence relations over transition systems of which the states are not necessarily closed process expressions. Let $\mathcal{Q}\cup\{\surd\}$ be the set of states under consideration and $\surd \notin \mathcal{Q}$ the (only) terminal state (so that $\surd$ has no outgoing transitions). We fist give the definitions and then provide some comments. Let $P, P' \in \mathcal{Q}$ and let $\equiv$ denote syntactic equivalence. The binary relation $\_ \Rightarrow \_$ on $\mathcal{Q}$ is defined by

$$P \Rightarrow P' \;\text{ if either } P \equiv P' \text{ or for some } P'',\ P \xrightarrow{\tau} P'' \Rightarrow P'.$$

In a similar way, the unary relation $\_ \Rightarrow \surd$ is defined by $P \Rightarrow \surd$ if $P \Rightarrow P' \xrightarrow{\tau} \surd$. We write $P \overset{a}{\Rightarrow} P'$ if $P \Rightarrow Q \xrightarrow{a} Q' \Rightarrow P'$, and $P \overset{a}{\Rightarrow} \surd$ if either $P \Rightarrow Q \xrightarrow{a} Q' \Rightarrow \surd$ or $P \Rightarrow Q \xrightarrow{a} \surd$ for some $Q, Q' \in \mathcal{Q}$. Observe that the relations $\overset{\tau}{\Rightarrow}$ and $\Rightarrow$ coincide (as do the $\surd$-variants of both).

Definition 3.1.1. A τ-*bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{Q} \cup \{\surd\}$ that satisfies

- $P\mathcal{R}Q$ and $P \xrightarrow{a} P'$ implies either $a = \tau$ and $P'\mathcal{R}Q$, or $\exists Q'.\ Q \overset{a}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$ where Q' may be $\surd$,
- $P\mathcal{R}Q$ and $Q \xrightarrow{a} Q'$ implies either $a = \tau$ and $P\mathcal{R}Q'$, or $\exists P'.\ P \overset{a}{\Rightarrow} P'$ and $P'\mathcal{R}Q'$ where P' may be $\surd$,
- $P\mathcal{R}Q$ and $P \xrightarrow{a} \surd$ implies either $a = \tau$ and $Q \equiv \surd$, or $Q \overset{a}{\Rightarrow} \surd$,
- $P\mathcal{R}Q$ and $Q \xrightarrow{a} \surd$ implies either $a = \tau$ and $P \equiv \surd$, or $P \overset{a}{\Rightarrow} \surd$.

Two states P, Q are τ-*bisimilar*, notation $P \leftrightarrow_\tau Q$, if there exists a τ-bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$.

The relation $\mathcal{R}$ is a *rooted* τ-bisimulation if for two root states P and Q designating a transition system the following conditions are satisfied:

- $P \xrightarrow{\tau} P'$ implies $\exists Q'.\ Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$ where P', Q' may be $\surd$, and
- $Q \xrightarrow{\tau} Q'$ implies $\exists P'.\ P \xrightarrow{\tau} P'$ and $P'\mathcal{R}Q'$ where P', Q' may be $\surd$.

Two states P, Q are *rooted τ-bisimilar*, notation $P \leftrightarroweq_{rb} Q$, if there exists a rooted branching bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$. □

The relations $\leftrightarroweq_{\tau}$ and $\leftrightarroweq_{r\tau}$ are equivalence relations.

Definition 3.1.2. A *branching bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{Q} \cup \{\surd\}$ that satisfies

- $P\mathcal{R}Q$ and $P \xrightarrow{a} P'$ implies either $a = \tau$ and $P'\mathcal{R}Q$, or $\exists Q', Q''.\ Q \Rightarrow Q'' \xrightarrow{a} Q'$ and $P\mathcal{R}Q''$ and $P'\mathcal{R}Q'$ where Q' may be $\surd$,
- $P\mathcal{R}Q$ and $Q \xrightarrow{a} Q'$ implies either $a = \tau$ and $Q'\mathcal{R}P$, or $\exists P', P''.\ P \xrightarrow{a} P'$ and $P''\mathcal{R}Q$ and $P'\mathcal{R}Q'$ where P' may be $\surd$,
- $P\mathcal{R}Q$ and $P \xrightarrow{a} \surd$ implies either $a = \tau$ and $Q \equiv \surd$, or $\exists Q'.\ Q \Rightarrow Q' \xrightarrow{a} \surd$ and $P\mathcal{R}Q'$,
- $P\mathcal{R}Q$ and $Q \xrightarrow{a} \surd$ implies either $a = \tau$ and $P \equiv \surd$, or $\exists P'.\ P \Rightarrow P' \xrightarrow{a} \surd$ and $P'\mathcal{R}Q$.

Two states P, Q are *branching bisimilar*, notation $P \leftrightarroweq_b Q$, if there exists a branching bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$.

The relation $\mathcal{R}$ is a *rooted* branching bisimulation if for two root states P and Q designating a transition system the following conditions are satisfied:

- $P \xrightarrow{a} P'$ where $a \in A_\tau$ and P' may be $\surd$ implies $\exists Q'.\ Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$ where Q' may be $\surd$,
- $Q \xrightarrow{a} Q'$ where $a \in A_\tau$ and Q' may be $\surd$ implies $\exists P'.\ P \xrightarrow{a} P'$ and $P'\mathcal{R}Q'$ where P' may be $\surd$.

Two states P, Q are *rooted branching bisimilar*, notation $P \leftrightarroweq_{rb} Q$, if there exists a rooted branching bisimulation $\mathcal{R}$ with $P\mathcal{R}Q$. □

Observe that also $\leftrightarroweq_b$ and $\leftrightarroweq_{rb}$ are equivalence relations.

The restriction to the rooted versions serves congruence properties. Taking $\mathcal{Q}$ the set of closed process expressions over $\mathsf{ACP}_\tau(A, \gamma)$ and $\_ \xrightarrow{a} \_$ and

$\_ \xrightarrow{a} \surd$ as defined by the rules in Table 3, the relation $\leftrightarrow_{r\tau}$ is a congruence (but $\leftrightarrow_{\tau}$ is not). Moreover, $\mathsf{ACP}^{*\$}_{\tau}(A,\gamma)$ is sound with respect to rooted τ-bisimilarity, and its fragment $\mathsf{ACP}_{\tau}(A,\gamma)$ is complete [BK85, BW90]. For $\mathsf{ACP}^{\tau}(A,\gamma)$ and $\mathsf{ACP}^{*\$}_{\tau}(A,\gamma)$ there are similar results with respect to rooted branching bisimilarity. Finally, note that bisimilar transition systems also are rooted τ-bisimilar and rooted branching bisimilar.

3.2 Expressiveness Results

In this section we first prove that each computable process over action set B can be expressed in $\mathsf{ACP}^{\tau*\$}(A_5(\{B\}),\gamma)$, thus $\mathsf{ACP}^{*\$}(A_5(B),\gamma)$ equipped with branching bisimulation equivalence. This means that for any transition system T representing a computable process over B there is an expression P in $\mathsf{ACP}^{\tau*\$}(A_5(\{B\}),\gamma)$ such that

$$T \leftrightarrow_{rb} P.$$

For $\mathsf{ACP}^{*\$}_{\tau}(A_5(\{B\}),\gamma)$, i.e., $\mathsf{ACP}^{*\$}(A_5(B),\gamma)$ with τ-bisimilarity, we can extend this result to a large class of semi-computable processes: those that initially are finitely branching. Therefore one may say that the theory of $\mathsf{ACP}^{*\$}_{\tau}(A_5(\{B\}),\gamma)$ is *expressively complete.* These results guarantee the expressivity of the systems under consideration, but have little practical value. More straightforward definitions of typical processes such as stacks, bags (multi-sets) and queues can be found in [BP98].

We consider a transition system T over alphabet B with states in $\mathbb{N}$, and in which 0 serves as initial state and 1 is the only terminal state (in particular, outgoing transitions from state 1 are considered irrelevant). For $a \in B$, $R_a \subseteq \mathbb{N}\times\mathbb{N}$ is the transition relation. We often write

$$n \xrightarrow{a} m$$

for $R_a(n,m)$. With $0^\frown 1T$ we denote the process (modulo strong bisimulation) represented by T, taking 0 as the root, 1 as the termination state, and nodes different from 1 without outgoing transitions representing deadlock. Formally: for $k \neq l$,

$$k^\frown lT = \begin{cases} \delta & \text{if } \forall n \forall a \neg R_a(k,n), \\ \left(\begin{array}{l} \sum_{\{a\in B \mid R_a(k,l)\}} a \; + \\ \sum_{\{a\in B, m\in\mathbb{N} \mid m\neq l, R_a(k,m)\}} a\cdot m^\frown lT \end{array}\right) & \text{otherwise.} \end{cases}$$

where sums with an empty index set are taken δ.

We call a process *computable* if in any state all possible transitions are finite in number and can be computed. Such a process can be modeled by a recursive transition system:

Definition 3.2.1. Let S be a recursive set of states with $0 \in S$. A transition system $T = \langle S, B, \{R_a \mid a \in B\}, 0\rangle$ is *recursive* iff for some injective function $h : B \to \mathbb{N}$ the transition system T can be represented by a (total) recursive function *next* such that for all $s \in S$ the value of $next(s)$ is the canonical index[1] (CI) of the finite set encoding all next steps from s:

$$next(s) = \mathrm{CI}(\{\langle h(a), s'\rangle \mid s \xrightarrow{a} s'\})$$

(so $D_{next(s)}$ contains all values $\langle h(a), s'\rangle$ for which $s \xrightarrow{a} s'$, and $next(0)$ represents T). □

We first do not concern rootedness, and formulate two basic lemma's. Using these, our main expressivity results follow in a straightforward manner.

Lemma 3.2.2. *Let T be a recursive transition system over B. Then $0^\frown 1T$ modulo branching bisimulation equivalence is expressible in* $\mathsf{ACP}^{\tau *\$}(A_5(\{B\}), \gamma)$

Proof. Define the bijective pairing function over $\mathbb{N}^2$ by $\langle n, m\rangle = \frac{1}{2}((n+m)^2 + 3m + n)$, with unpairing functions $(\_)_0$ and $(\_)_1$. So $\langle 0,0\rangle = 0$, $\langle 1,0\rangle = 1$, and $n = \langle (n)_0, (n)_1\rangle$.

We transform T to $\overline{T}$ by replacing each state with a τ-loop of appropriate size, of which each state has at most one outgoing B-transition:

1. $\overline{S} = \{\langle s, j\rangle \mid s \in S, j \le next(s)\}$,
2. $\forall s \in S \setminus \{1\}.\ \langle s, 0\rangle \xrightarrow{\tau} \langle s, next(s)\rangle$ and if $j < next(s)$ then $\langle s, j+1\rangle \xrightarrow{\tau} \langle s, j\rangle$,
3. $\langle s, j\rangle \xrightarrow{a} \langle s', 0\rangle$ if $s \xrightarrow{a} s'$ in T and $\langle h(a), s'\rangle = j$. (Note that $x \in D_{next(s)} \Rightarrow x < next(s)$).

This yields a transition system $\overline{T}$ with labels from $B \cup \{\tau\}$ in which the number of outgoing transitions of each state is at most two and that is recursive: extend the function h (encoding the actions of B in $\mathbb{N}$) to B_τ, then $\overline{next}(s)$ can be computed from $next(s)$ and the transitions defined above. Moreover, $\overline{T}$ satisfies

$$0^\frown 1T \leftrightarrow_b 0^\frown 1\overline{T}$$

[1] The canonical index of $\emptyset$ is 0, of $\{k_1, k_2, ..., k_l\}$ it is the number $2^{k_1} + 2^{k_2} + ... + 2^{k_l}$, and D_x is the finite set with canonical index x.

where the terminal state 1 plays the role of $\surd$. The branching bisimulation BB is as follows:

$$\mathrm{BB}(s,\langle s,j\rangle) \text{ for } s\in S,\ j\leq next(s).$$

Let function g be the inverse of h. We describe $\overline{T}$ with four computable functions:

$$\begin{array}{lcl}
action(n) & = & \begin{cases} 1 & \text{if } \exists a\in B, m\in\mathbb{N} \text{ with } n\xrightarrow{a} m \text{ in } \overline{T},\\ 0 & \text{otherwise,}\end{cases}\\[2ex]
label(n) & = & m \text{ if } action(n)=1 \text{ and } \exists l\in\mathbb{N} \text{ with } n\xrightarrow{g(m)} l \text{ in } \overline{T},\\[2ex]
\tau\text{-}step(n) & = & \begin{cases} \langle s,j\rangle & \text{if } n=\langle s,j+1\rangle \text{ and } j<next(s),\\ \langle s,next(s)\rangle & \text{if } n=\langle s,0\rangle \text{ and } s\neq 1,\end{cases}\\[2ex]
next\text{-}state(n) & = & m \text{ if } \exists a\in B,\ m\in\mathbb{N} \text{ with } n\xrightarrow{a} m \text{ in } \overline{T}.
\end{array}$$

For computable functions f_0, f_1 we write

$$P: 0,0,0,0,n\rightarrow 0,0,0,f_1(n),f_0(n)$$

if for all n there is an m such that

$$\partial_{H_5}(Px\parallel\mathcal{R}_5(0,0,0,0,n)) = t^m\cdot\partial_{H_5}(x\parallel\mathcal{R}_5(0,0,0,f_1(n),f_0(n))).$$

Choose $P_1,P_2,P_3,P_4\in\mathsf{RMIP}(\emptyset,5,5)$ such that $\mathsf{ACP}^{*\$}(A_5(\emptyset),\gamma)\vdash$

$$\begin{array}{llcll}
P_1: & 0,0,0,0,n & \rightarrow & 0,0,0,m,n & \text{if } m=action(n),\\
P_2: & 0,0,0,0,n & \rightarrow & 0,0,0,m,n & \text{if } m=label(n),\\
P_3: & 0,0,0,0,n & \rightarrow & 0,0,0,0,m & \text{if } m=\tau\text{-}step(n),\\
P_4: & 0,0,0,0,n & \rightarrow & 0,0,0,0,m & \text{if } m=next\text{-}state(n).
\end{array}$$

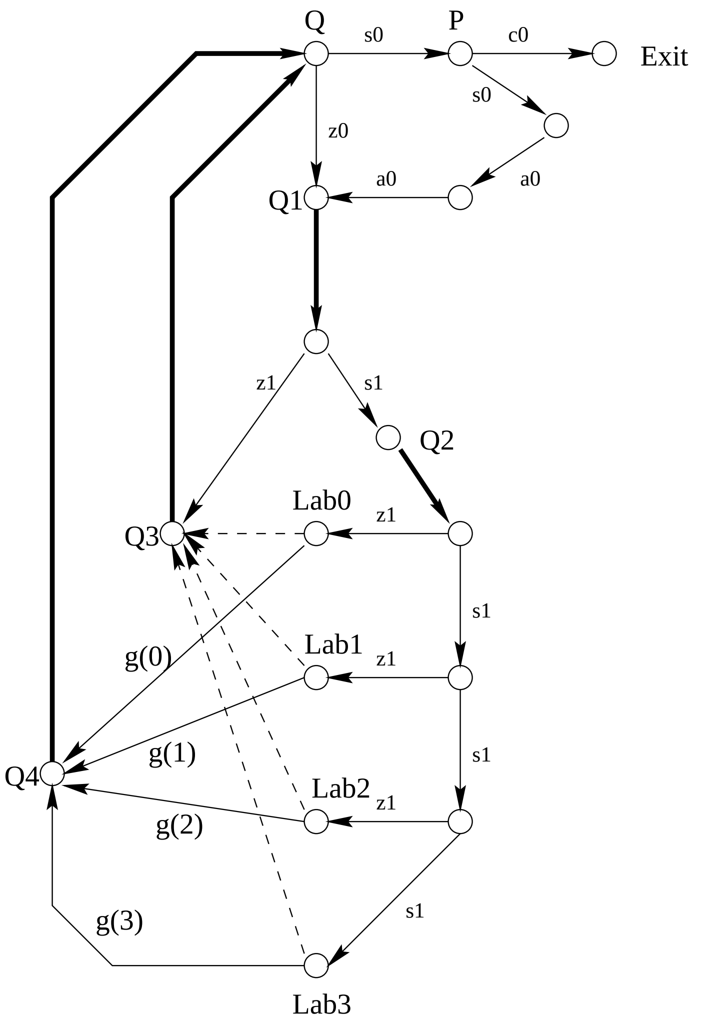
In the following we assume without loss of generality that $|B|>1$. For the case $|B|=4$ consider the program schematically depicted in Fig. 1, suggesting how the program should be adapted for other values of $|B|$. In this scheme a dashed arrow stands for a t-step, and a bold arrow starting from Q_i ($i=1,...,4$) for the sequence of t-steps invoked by register machine program P_i. We first argue that P, and therefore also each of its substates, is expressible in $\mathsf{RMIP}(B_t,5,0)$ (modulo strong bisimulation). Let S be the $*$-free process that starts in state Q_1 and ends in state Q. Then we define

$$\begin{array}{rcl}
R & = & s_0a_0a_0S((z_0S)^*s_0),\\
\mathsf{Exit} & = & (s_1{}^*c_1)(s_2{}^*c_2)(s_3{}^*c_3)(s_4{}^*c_4).
\end{array}$$

and conclude that $P \leftrightarrow R^*c_0\cdot\mathsf{Exit}$.

Now $0^\frown 1\overline{T}\leftrightarrow_b \tau_{\{t\}}\circ\partial_{H_5}(Q\parallel\mathcal{R}_5(0,0,0,0,0))$ where the terminal state 1 plays the role of $\surd$. We shall not define a witnessing bisimulation relation in detail but

Figure 1: A register program scheme.



sketch the idea. Each state $\langle s,j\rangle$ in $\overline{T}$ with $s\neq 1$ is related to each state in the τ-loop containing $\tau_{\{t\}}\circ\partial_{H_5}(Q\parallel\mathcal{R}_5(0,0,0,0,\langle s,0\rangle))$. The different states $\langle s,...\rangle$ of this $\overline{T}$-loop are maintained in register process R_0 by P_1, P_2 and P_3 and the remaining transitions in Fig. 1. Furthermore,

$$\langle s,j\rangle \quad \xrightarrow{g(i)} \quad \langle s',0\rangle\in\overline{T}$$

if and only if

$$\tau_{\{t\}}\circ\partial_{H_5}\left(\mathsf{Lab}_{g(i)}\parallel\mathcal{R}_5(0,0,0,0,\langle s,j\rangle)\right)$$

$$\downarrow g(i)$$

$$\tau_{\{t\}}\circ\partial_{H_5}(Q_4\parallel\mathcal{R}_5(0,0,0,0,\langle s,j\rangle))$$

$$\Downarrow\tau$$

$$\tau_{\{t\}}\circ\partial_{H_5}(Q\parallel\mathcal{R}_5(0,0,0,0,\langle s',0\rangle))\,.$$

Finally, the terminal state $\langle 1,0\rangle$ in $\overline{T}$ is related to

$$\tau_{\{t\}}\circ\partial_{H_5}(Q\parallel\mathcal{R}_5(0,0,0,0,\langle 1,0\rangle))\,,$$

which terminates with a τ trace via the Exit subprogram. □

For the case of τ-bisimilarity we have a stronger result. This result uses the same type of register program, but is based on a different transformation.

Lemma 3.2.3. *Let T be an r.e. transition system over B, i.e., all relations R_a are semi-computable. Then $0^\frown 1T$ modulo τ-bisimulation equivalence is expressible in $\mathsf{ACP}_\tau^{*\$}(A_5(\{B\}),\gamma)$.*

Proof. Define the bijective pairing function over $\mathbb{N}^2$ as in the previous proof and let $R_a(n,m)\iff\exists kR'_a(n,m,k)$ with R'_a decidable. We transform T to $\overline{T}$ as follows:

1. $\forall n\in\mathbb{N}\setminus\{1\},\ i\in\mathbb{N}\setminus\{0\},\ \langle n,i\rangle\xrightarrow{\tau}\langle n,i+1\rangle$,
2. $\forall n,i\in\mathbb{N}$ write $i=\underline{\langle act,\langle m,\langle k,l\rangle\rangle\rangle}$. Now, if $act=h(a)$ for h encoding the actions of B and $R'_a(n,m,k)$, then $\overline{R}_a(\langle n,i\rangle,\langle m,0\rangle)$.

We notice that $\langle 0,0\rangle^\frown\langle 1,0\rangle\overline{T}\ \underline{\leftrightarrow}_\tau 0^\frown 1T$. The τ-bisimulation TB is as follows:

TB$(0,0)$,
TB$(\langle k,n\rangle,k)$ for all n if $k\neq 1$, and
TB$(1,1)$.

Furthermore, notice that in $\overline{T}$ for each state $n\neq 1$

- there is exactly one outgoing τ-transition $n \xrightarrow{\tau} \langle (n)_0, (n)_1 + 1\rangle$,
- there is at most one outgoing a-step for at most one $a \in B$.

Let g be the inverse of h. We describe $\overline{T}$ with four computable functions, of which only τ-*step*(n) is different from the previously defined functions:

$$\begin{array}{lcl} \mathit{action}(n) & = & \begin{cases} 1 & \text{if } \exists a \in B, m \in \mathbb{N} \text{ with } n \xrightarrow{a} m \text{ in } \overline{T}, \\ 0 & \text{otherwise,} \end{cases} \\ \\ \mathit{label}(n) & = & m \text{ if } \mathit{action}(n) = 1 \text{ and } \exists l \in \mathbb{N} \text{ with } n \xrightarrow{g(m)} l \text{ in } \overline{T}, \\ \\ \tau\text{-}\mathit{step}(n) & = & \langle (n)_0, (n)_1 + 1\rangle \text{ if } n \neq 1, \\ \\ \mathit{next\text{-}state}(n) & = & m \text{ if } \exists a \in B,\ m \in \mathbb{N} \text{ with } n \xrightarrow{a} m \text{ in } \overline{T}. \end{array}$$

Using the notation $P : 0,0,0,0,n \to 0,0,0,f_1(n),f_0(n)$ from the previous proof, choose $P_1, P_2, P_3, P_4 \in \mathsf{RMIP}(\emptyset, 5, 5)$ such that $\mathsf{ACP}^{*\$}(A_5(\emptyset), \gamma) \vdash$

$$\begin{array}{llcll} P_1 : & 0,0,0,0,n & \to & 0,0,0,m,n & \text{if } m = \mathit{action}(n), \\ P_2 : & 0,0,0,0,n & \to & 0,0,0,m,n & \text{if } m = \mathit{label}(n), \\ P_3 : & 0,0,0,0,n & \to & 0,0,0,0,m & \text{if } m = \tau\text{-}\mathit{step}(n), \\ P_4 : & 0,0,0,0,n & \to & 0,0,0,0,m & \text{if } m = \mathit{next\text{-}state}(n). \end{array}$$

Again consider the register program $Q \in \mathsf{RMIP}(B \cup \{t\}, \{0,1,...,5\}, \emptyset)$ schematically depicted in Fig. 1 (with $B = \{g(0), g(1), g(2), g(3)\}$ and P_i defined as above).

Now $0 \frown 1\overline{T} \leftrightarrow_\tau \tau_{\{t\}} \circ \partial_{H_5} (Q \parallel \mathcal{R}_5(0,0,0,0,0))$ where the terminal state 1 plays the role of $\surd$. We sketch a τ-bisimulation: first note that each state $\langle s, j\rangle$ in $\overline{T}$ with $s \neq 1$ is related to each state in the infinite τ-sequence starting with $\tau_{\{t\}} \circ \partial_{H_5} (Q \parallel \mathcal{R}_5(0,0,0,0,\langle s, 0\rangle))$. The different states $\langle s, ...\rangle$ of this $\overline{T}$-sequence are maintained in register process R_0 by P_1, P_2 and P_3 and the remaining transitions in Fig. 1. Furthermore,

$$\langle s, j\rangle \quad \xrightarrow{g(i)} \quad \langle s', 0\rangle \in \overline{T}$$

if and only if

$$\begin{array}{l} \tau_{\{t\}} \circ \partial_{H_5} \left(\mathsf{Lab}_{g(i)} \parallel \mathcal{R}_5(0,0,0,0,\langle s, j\rangle)\right) \\ \qquad \downarrow g(i) \\ \tau_{\{t\}} \circ \partial_{H_5} (Q_4 \parallel \mathcal{R}_5(0,0,0,0,\langle s, j\rangle)) \\ \qquad \Downarrow \tau \\ \tau_{\{t\}} \circ \partial_{H_5} (Q \parallel \mathcal{R}_5(0,0,0,0,\langle s', 0\rangle)) \, . \end{array}$$

Finally, the terminal state $\langle 1, 0\rangle$ in $\overline{T}$ is related to $\tau_{\{t\}} \circ \partial_{H_5} (Q \parallel \mathcal{R}_5(0,0,0,0,\langle 1, 0\rangle))$, which terminates with a τ trace via the Exit subprogram. $\square$

With these two lemma's our main expressivity results follow immediately.

Theorem 3.2.4. *Let T be a recursive transition system over B. Then $0^\frown 1T$ modulo rooted branching bisimulation equivalence is expressible in* $\mathsf{ACP}^{\tau *\$}(A_5(\{B\}),\gamma)$.

Proof. Transform T into $\widetilde{T}$ by unfolding its root: rename all states in T according to $\rho:\mathbb{N}\to\mathbb{N}$ defined by

$$\rho(n)=\begin{cases}1 & \text{if } n=1,\\ n+2 & \text{otherwise.}\end{cases}$$

Then $\widetilde{T}$ with root state 0 and terminal state 1 has for each $n\xrightarrow{a}m\in T$ a transition $\rho(n)\xrightarrow{a}\rho(m)$, and for each $0\xrightarrow{a}n\in T$ a transition $0\xrightarrow{a}\rho(n)$. This implies that $0^\frown 1T \leftrightarroweq 0^\frown 1\widetilde{T}$, and

$$0^\frown 1\widetilde{T}=\begin{cases}\left(\begin{array}{l}\sum_{\{a\in B\mid R_a(0,1)\}}a\,+\\ \sum_{\{a\in B,m\in\mathbb{N}\mid R_a(0,m)\}}a\cdot m^\frown 1\widetilde{T}\end{array}\right) & \text{if } \bigcup_{a\in B,j\in\mathbb{N}}R_a(0,j) \text{ is not empty,}\\ \delta & \text{otherwise.}\end{cases}$$

Because $\widetilde{T}$ has no incoming transitions in its root state, it is sufficient to express $m^\frown 1\widetilde{T}$ modulo branching bisimulation for each appropriate value of $m\in\mathbb{N}\setminus\{0\}$. As proved in Lemma 3.2.2, this is possible. $\square$

In exactly the same way our next expressivity result follows from Lemma 3.2.3.

Theorem 3.2.5. *Let T be an r.e. transition system over B that initially is finitely branching. Then $0^\frown 1T$ modulo rooted τ-bisimulation equivalence is expressible in* $\mathsf{ACP}_\tau^{*\$}(A_5(\{B\}),\gamma)$.

We note that this results strengthens the expressivity result in [BBK87], in which expressibility of recursive transition systems in $\mathsf{ACP}_\tau(A,\gamma)$ with finite, guarded recursive specifications is proved.

3.3 Strengthening Results

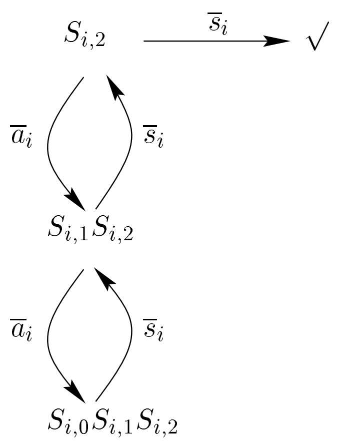
First, observe that in the proof of the basic Lemma 2.4.2 the register R_3 is 'used' up to a uniformly bounded depth (depending on the number of instructions of register program $\overline{P}$). So a bounded counter up to depth $N+1$, say $R_{3,N+1}$, would suffice. Such a process is definable with use of only $*$ and its specific actions, as shown below. As a consequence, Lemma 2.4.2 can be formulated as a "three-\$-statement", i.e., with the use of only three registers, and all previous results can be obtained with one register less.

Definition 3.3.1. A *subcounter* $S_{i,N}$ is defined by induction to its *depth* N $N \in \mathbb{N}$ in the following way:

$$S_{i,0} = \overline{s}_i$$
$$S_{i,n+1} = (\overline{a}_i \cdot S_{i,n})^* \overline{s}_i.$$

□

As an example, the subcounter $S_{i,2}$ can be visualized as follows:



Lemma 3.3.2. *A register with finite depth can be defined in* $\mathsf{ACP}^*(A, \gamma)$.

Proof. $R_{i,0} = \overline{z}_i{}^* \overline{c}_i$ and $R_{i,n+1} = (\overline{a}_i \cdot S_{i,n} + \overline{z}_i)^* \overline{c}_i$. □

Next, we show that we do not need binary Kleene star: its use can be encoded with help of auxiliary actions and abstraction. This is both the case for our results on $\underline{\leftrightarrow}_{rb}$ and on $\underline{\leftrightarrow}_{r\tau}$. In order to prove this, let

$$\simeq \;\; \in \{ \underline{\leftrightarrow}_{rb} \,,\, \underline{\leftrightarrow}_{r\tau} \}.$$

We shall only use that $aX \simeq a\tau X$, and we repeat below the two basic results (cf. [BP98]). The first of these states that each finite state transition system can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ or $\mathsf{ACP}^{\$}_{\tau}(A^{ext}, \gamma)$ where A^{ext} is a finite extension of A, i.e., $A^{ext} \setminus A$ is finite. So, in particular our modeling of *register machine programs* does not depend on the use of $*$.

Theorem 3.3.3. *For each finite state transition system with labels in A there is a finite extension A^{ext} of A such that P can be expressed in* $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ *or in* $\mathsf{ACP}^{\$}_{\tau}(A^{ext}, \gamma)$ *with handshaking only, and the actions in A not subject to communication.*

Proof. Let T be a finite state transition system over A. Then T can be characterized by P_1, where P_1 is defined by the linear system

$$P_i = \Big(\sum_{j=1}^{n} \alpha_{i,j} \cdot P_j \Big) + \beta_i \qquad (i = 1, ..., n)$$

with all $\alpha_{i,j}$ and β_i finite sums of actions or δ. Define A^{ext} as the extension of A with the following $2n+3$ actions:

$$\{t\} \cup H, \text{ where } H = \{r_j, s_j \mid j = 0, ..., n\},$$

and consider the following processes:

$$\begin{array}{lll} \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot s_j\right) + \beta_i & \text{abbreviated by} & F_i \text{ for } i = 1, ..., n \\ \left(\sum_{j=1}^{n} r_j \cdot F_j\right)^{\$} r_0 & \text{abbreviated by} & K \\ \left(\sum_{j=1}^{n} r_j \cdot s_j\right)^{\$} s_0 & \text{abbreviated by} & L. \end{array}$$

Then $P_1 \simeq \tau_{\{t\}} \circ \partial_H(F_1 \cdot K \parallel L)$. This can be shown with help of the infinite transition system

$$Y_i(k) = \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot Y_j(k+1)\right) + \beta_i, \; i = 1, ..., n, \; n \geq 1, \text{ and } k \in \mathbb{N}.$$

Obviously, $P_i \leftrightarrows Y_i(k)$ $(k \in \mathbb{N})$. So it suffices to show that $\tau_{\{t\}} \circ \partial_H(F_i \cdot K \parallel L) \simeq Y_i(0)$. We show this by first omitting the $\tau_{\{t\}}$-application: for $k \in \mathbb{N}$,

$$\begin{aligned} \partial_H(F_i \cdot K^{k+1} \parallel L^{k+1}) &= \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot \partial_H(s_j \cdot K^{k+1} \| L^{k+1})\right) + \beta_i \cdot \partial_H(K^{k+1} \| L^{k+1}) \\ &= \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot t \cdot \partial_H(K^{k+1} \parallel s_j \cdot L^{k+2})\right) + \beta_i \cdot t^{k+1} \\ &= \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot t \cdot t \cdot \partial_H(F_j \cdot K^{k+2} \parallel L^{k+2})\right) + \beta_i \cdot t^{k+1}. \end{aligned}$$

Hence, applying $\tau_{\{t\}}$ and axiom $x \cdot \tau = x$ we find for each k

$$\tau_{\{t\}} \circ \partial_H(F_i \cdot K^{k+1} \| L^{k+1}) = \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot \tau_{\{t\}} \circ \partial_H(F_j \cdot K^{k+2} \| L^{k+2})\right) + \beta_i.$$

So $\tau_{\{t\}} \circ \partial_H(F_i \cdot K^{k+1} \parallel L^{k+1})$ satisfies the equation characterizing state $Y_i(k)$, and hence

$$\tau_{\{t\}} \circ \partial_H(F_i \cdot K^{k+1} \parallel L^{k+1}) \simeq Y_i(k),$$

and in particular $P_1 \simeq Y_1(0) \simeq \tau_{\{t\}} \circ \partial_H(F_i \cdot K \parallel L)$. □

We are done if we show that also a *register process* can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ or $\mathsf{ACP}_{\tau}^{\$}(A^{ext}, \gamma)$, thus without the use of $*$.

Theorem 3.3.4. *A register process* $(a(a^{\$} b) + z)^{*} c$ *can be defined in* $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ *or in* $\mathsf{ACP}_{\tau}^{\$}(A^{ext}, \gamma)$ *if* $|A^{ext} \setminus A| \geq 5$.

Proof. Let $A^{ext} \setminus A = \{t\} \cup \{r_i, s_i \mid i = 0, 1\}$ and consider

$$\begin{array}{rll} (a(a^{\$} b) + z) s_1 + c & \text{abbreviated by} & P \\ (r_1 \cdot P)^{\$} r_0 & \text{abbreviated by} & Q \\ (r_1 \cdot s_1)^{\$} s_0 & \text{abbreviated by} & R. \end{array}$$

Now $(a(a^{\$} b) + z)^{*} c \simeq \tau_{\{t\}} \circ \partial_{\{r_i, s_i \mid i=0,1\}}(P \cdot Q \parallel R)$ follows in a similar way as explained in the previous proof. □

4 Conclusions

We showed that an expressive format is obtained by adding \$ as a primitive to ACP with abstraction. Adding binary Kleene star as well yields a more flexible format for the specification of concurrent processes, in which register machine computation can be modeled in a detailed and straightforward manner. These results provide a purely equational founding of process algebra: any process of interest can simply be represented by a term.

References

[BBK87] J.C.M. Baeten, J.A. Bergstra and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[BBP94] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *Computer Journal*, 37(4):243–258, 1994.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109-137, 1984.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[BP98] J.A. Bergstra and A. Ponse. Two Recursive Generalizations of Iteration in Process Algebra. Technical Report P9808, Programming Research Group, University of Amsterdam, 1998.
(See also `http://www.wins.uva.nl/research/prog/reports/reports.html`.)

[BT84] J.A. Bergstra and J.V. Tucker. Top down design and the algebra of communicating processes. *Science of Computer Programming*, 5(2):171–199, 1984.

[BV93] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, ed., *Proceedings 4th Conference on Concurrency Theory (CONCUR 93)*, Hildesheim, LNCS 715, pp. 477–492. Springer-Verlag, 1993.

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

[FZ94] W.J. Fokkink and H. Zantema. Basic process algebra with iteration: completeness of its equational axioms. *Computer Journal*, 37(4):259–267, 1994.

[GV92] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[GW96] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, Princeton NJ, 1956.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Min67] M.L. Minsky. *Computation: finite and infinite machines*, Prentice-Hall, Englewood Cliffs N.J., 1967.

A Universal Register Machines with Two Registers

It is a standard fact that the class of recursive functions is characterized by *register machine computability*. Here we recall some basic facts. A register machine program is a (finite) set of instructions numbered $1, ..., k$ of the following form:

halt	halt,
(a_i, l)	add 1 to register i and go to instruction l,
(s_i, l, l')	if possible substract 1 from register i and go to instruction l, otherwise go to l'.

A unary function $f : \mathbb{N} \to \mathbb{N}$ is register machine computable if there exists a register program P that operates on a finite number of registers, say $1, ..., m$, in such a way that $f(n)$ is defined if and only if

$$\langle n, 0, ..., 0\rangle$$

$$\Downarrow P$$

$$\langle f(n), 0, ..., 0\rangle$$

Here $\langle x_1, x_2, ..., x_m\rangle$ gives the contents of the registers $1, 2, ..., m$, P is started with the instruction numbered 1, and P terminates by executing a halt instruction resulting in machine state $\langle f(n), 0, ..., 0\rangle$. If $f(n)$ is not defined, the computation programmed by P on state $\langle n, 0, ..., 0\rangle$ is perpetual ('diverges').

Let P be some fixed register program that uses m registers and that computes unary function f. We sketch a uniform construction transforming P into a register program that uses only two registers. To this end we assume $m > 2$ (otherwise there is nothing to show) and use *prime factorization*. Computation of P can be simulated by encoding each m-tuple

$$\langle x_1, x_2, ..., x_m\rangle$$

(representing some state of the m registers) as the number

$$2^{x_1} \cdot 3^{x_2} \cdot 5^{x_3} \cdot ... \cdot p_m^{x_m},$$

where p_m is the m_{th} prime number. It suffices to show how the effect of instructions (a_i, l) and (s_i, l, l') of P can be simulated on these encoded states. The translation of (a_i, l), say register machine program fragment (A_i, L) for some appropriate instruction number L, should be such that it transforms $2^{x_1} \cdot 3^{x_2} \cdot ... \cdot p_m^{x_m}$ into

$$2^{x_1} \cdot 3^{x_2} \cdot ... \cdot p_i^{x_i+1} \cdot ... \cdot p_m^{x_m} = p_i \cdot 2^{x_1} \cdot 3^{x_2} \cdot ... \cdot p_i^{x_i} \cdot ... \cdot p_m^{x_m}.$$

This can be easily defined in terms of (a_j, l) and (s_j, l, l') for $j \in \{1,2\}$, for instance as follows:

$$\begin{array}{rl}
1: & (s_1, 2, p_i+2) \\
2: & (a_2, 3) \\
3: & (a_2, 4) \\
 & \vdots \\
p_i: & (a_2, p_i+1) \\
p_i+1: & (a_2, 1) \\
p_i+2: & (s_2, p_i+3, L) \\
p_i+3: & (a_1, p_i+2).
\end{array}$$

so

$$\langle 2^{x_1} \cdot 3^{x_2} \cdot \ldots \cdot p_m^{x_m}, 0 \rangle$$

$$\Downarrow (A_i, L)$$

$$\langle p_i \cdot 2^{x_1} \cdot 3^{x_2} \cdot \ldots \cdot p_i^{x_i} \cdot \ldots \cdot p_m^{x_m}, 0 \rangle$$

where L is the current instruction no.

Simulating (s_i, l, l') is slightly more complex: the problem is to decide whether our code x has p_i as a divisor, i.e., to decide whether x_i is zero or not. This can be done by repeatedly substracting p_i from x in register 1, while counting upwards in register 2 (which initially is empty). If this leaves no remainder in register 1, then the quotient can be copied back into register 1. If this leaves a remainder (p_i is not a divisor), this remainder is stored by a position in the program, upon which the remaining part of the program should copy back the p_i-fold of the contents of register 2 in register 1. As an example assume $p_i = 3$ (thus $i = 2$). The simulation (S_2, L, L') for appropriate instruction numbers L, L' can be programmed as follows:

$$\begin{array}{rl}
1: & (s_1, 2, 5) \\
2: & (s_1, 3, 8) \\
3: & (s_1, 4, 7) \\
4: & (a_2, 1) \\
5: & (s_2, 6, L) \\
6: & (a_1, 5) \\
7: & (a_1, 8) \\
8: & (a_1, 9) \\
9: & (s_2, 10, L') \\
10: & (a_1, 11) \\
11: & (a_1, 12) \\
12: & (a_1, 9)
\end{array}$$

So, e.g.,

$\langle 2, 0 \rangle$	$\langle 3, 0 \rangle$
$\Downarrow (S_2, L, L')$	$\Downarrow (S_2, L, L')$
$\langle 2, 0 \rangle$	$\langle 1, 0 \rangle$
where L' is the current instruction no.	where L is the current instruction no.

So each instruction (a_i, l) and (s_i, l, l') occurring in P can be systematically translated into a program fragment that operates on a two register machine and transforms the encoded states appropriately. Modifying the instruction numbers of the program fragments thus obtained, this yields a two register machine program $\overline{P}$ that satisfies

$$\begin{array}{ccc}
\langle n, 0, \ldots, 0 \rangle & \text{if and only if} & \langle 2^n, 0 \rangle \\
\Downarrow P & & \Downarrow \overline{P} \\
\langle f(n), 0, \ldots, 0 \rangle & & \langle 2^{f(n)}, 0 \rangle
\end{array}$$

Of course, if P diverges on some input configuration $\langle x_1, ..., x_m\rangle$, then $\overline{P}$ diverges on $\langle 2^{x_1} \cdot ... \cdot p_m^{x_m}, 0\rangle$.

Finally, $\overline{P}$ can easily be used to simulate each terminating computation programmed by P, *including* the (de)coding of input/output values: add a register 0 for input/output, and let pre be a register program that transforms $\langle n,0,0\rangle$ into $\langle 0,2^n,0\rangle$ and post a program with the reverse effect. Then

$$\begin{array}{c}
\langle n,0,0\rangle \\
\Downarrow \mathsf{pre} \\
\langle 0,2^n,0\rangle \\
\Downarrow \overline{P} \\
\langle 0,2^{f(n)},0\rangle \\
\Downarrow \mathsf{post} \\
\langle f(n),0,0\rangle
\end{array}
\qquad
\text{where }\ \mathsf{pre} = \begin{array}{rl}
1: & (a_1,2)\\
2: & (s_0,3,8)\\
3: & (s_1,4,6)\\
4: & (a_2,5)\\
5: & (a_2,3)\\
6: & (s_2,7,2)\\
7: & (a_1,6)\\
8: & \mathsf{halt}
\end{array}
\quad \text{and }\ \mathsf{post} = \begin{array}{rl}
1: & (s_1,2,4)\\
2: & (s_1,3,7)\\
3: & (a_2,1)\\
4: & (s_2,5,6)\\
5: & (a_1,4)\\
6: & (a_0,1)\\
7: & \mathsf{halt}
\end{array}$$

So pre; $\overline{P}$; post simulates the computation of f on a three register machine, just as is claimed in Lemma 2.4.2. It is straightforward how concatenation ";" of programs can be defined in this case, e.g., pre; $\overline{P}$ can be obtained by removing instruction 8 in pre and adding 7 to all instruction numbers and references occurring in $\overline{P}$.