Register-Machine Based Processes

JAN A. BERGSTRA

University of Amsterdam, The Netherlands; Utrecht University, The Netherlands

AND

ALBAN PONSE

University of Amsterdam, The Netherlands; *CWI, Amsterdam, The Netherlands*

Abstract. We study extensions of the process algebra axiom system ACP with two recursive operations: the *binary Kleene star* $*$, which is 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 it is easy to represent register machine computation, and an equational theory results that is not decidable. In order to increase the expressive power, abstraction is then added: with *rooted branching bisimulation equivalence* each computable process can be expressed, and with *rooted τ-bisimilarity* each semi-computable process that initially is finitely branching can be expressed. Moreover, with abstraction and a finite number of auxiliary actions these results can be obtained without binary Kleene star. Finally, we consider two alternatives for the push-down operation. Each of these gives rise to similar results.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Specification techniques*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*Decision problems*

General Terms: Theory

Additional Key Words and Phrases: Bisimulation equivalence, computability, concurrency, expressivity, iteration, Kleene star, process algebra, push-down operation

1. *Introduction*

In this paper we take as a point of departure the process algebra axiom system ACP, that is, the Algebra of Communicating Processes defined by Bergstra and Klop [1984] and overviewed in Baeten and Weijland [1990], Baeten and Verhoef [1995], and Fokkink [2000]. ACP is an algebraic approach to concurrency theory

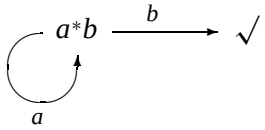
Authors' address: Programming Research Group, University of Amsterdam, Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands, e-mail: alban@science.uva.nl, Web: http://www. science.uva.nl/research/prog/.

that supports the interleaving hypothesis: concurrency can be modeled and analyzed in terms of the interleaving and synchronization of actions (elementary, indivisable processes) by a small number of primitive operations. (This is further explained in Section 2.) Although the syntax of ACP is suitable for the specification of finite processes, it is common practice to consider (potentially) infinite behavior, specified by means of recursive equations, and analyzed with help of the axioms of ACP and specific proof rules. Here we follow a different approach to the specification of infinite behavior and use recursive (or "iterative") *operations* instead (cf. Bergstra et al. [1993, 1994] and Bergstra and Ponse [2001]; for an overview see Bergstra et al. [2001]). This approach has lately attracted significant attention,[1] and provides a way to handle infinite processes as terms, hence supporting the equational founding of process algebra.

The purpose of this paper is to establish elementary computability and expressivity results of some particular extensions of ACP with recursive operations. We consider the *binary Kleene star* as the most basic recursive operation. This operation, notation $*$, stems from Kleene [1956] and is in process algebra defined by

$$x^*y = x \cdot (x^*y) + y$$

(cf. Bergstra et al. [1993, 1994]). Here, $+$ is the process algebra operation that models choice, and $\cdot$ (product) models sequential composition. (As usual, $\cdot$ binds stronger than $+$ and the symbol $\cdot$ is often omitted.) With the binary Kleene star one specifies *regular processes*, that is, finite state processes. As an example, for actions a and b, the process term a^*b characterizes the following behavior:



where the labeled arrows represent the execution of actions and $\surd$ expresses termination. The present paper can be seen as a follow-up of Bergstra et al. [1994], where we showed that ACP extended with abstraction and binary Kleene star is suitable to express each *regular* process if one adopts common behavioral semantics.

In this paper we first consider an extension of ACP with two recursive operations:

—the binary Kleene star $*$ as introduced above;

—the *push-down* operation \$, defined by $x^{\$}y = x((x^{\$}y)(x^{\$}y)) + y$ (in Bergstra and Ponse [2001]).

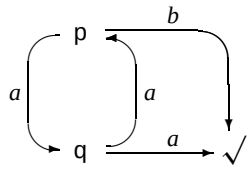
In Section 3 we describe a setting in which (unary) recursive functions are computed by "register-machine based processes." Register machine programs and registers, that is, the essential ingredients of register machine computation, are modeled by sequential, deterministic processes. A register machine computation is then

[1] The quest for axiomatizations of various behavioral equivalences for various forms of iteration turned out to be attractive: see, for example, Fokkink and Zantema [1994], Fokkink [1994, 1996, 1997], Aceto et al. [1996, 1998b, 1998c], Aceto and Ingólfsdóttir [1996], Aceto and Fokkink [1997], van Glabbeek [1997], and Aceto and Groote [1999].

specified as the parallel composition of these processes, and results in a sequence of synchronization actions that stem from communications between the "program" and the "registers." There is one distinct register that initially contains the input value, and upon termination of the program the (computed) output value. In the case of nondefinedness, the register machine computation diverges by performing an infinite sequence of synchronization actions. It easily follows that the resulting theory (the set of consequences provable from our extension of ACP) is undecidable.

In spite of 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 ACP extended with $*$ and \$ (its standard semantics—strong bisimilarity, see Park [1981]—being taken for granted). This is for instance the case for the process p recursively defined below with actions a and b:

$$\mathsf{p} = a\mathsf{q} + b,$$
$$\mathsf{q} = a\mathsf{p} + a, \quad \text{or} \quad \text{in a picture:}$$



This lack of expressivity is solved in Section 4, where we include *abstraction* as an additional feature. We consider two well-known approaches:

—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.

Moreover, with abstraction (and auxiliary actions) at hand, the use of the binary Kleene star can be avoided.

Finally, we consider in Section 5 two alternatives for the push-down operation, and argue that the results described above are preserved. The article ends with some conclusions (Section 6). We added an appendix on the uniform construction of register machines with two registers (based on Minsky [1967]).

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

In this section we briefly recall the process algebra axiom system ACP and consider its extension with the recursive operations $*$ and \$ in detail. Then we provide a (standard) operational semantics. Finally, we show that the second example process described in the Introduction cannot be expressed in the present extension.

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*:

$$\gamma(a, b) = \gamma(b, a) \qquad \text{if } \gamma(a, b) \downarrow \text{ (i.e., } \gamma(a, b) \textit{ defined}),$$
$$\gamma(a, \gamma(b, c)) = \gamma(\gamma(a, b), c) \qquad \text{if } \gamma(a, \gamma(b, c)) \downarrow.$$

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 while

γ is not fully undefined on $A \times A$, we speak of *handshaking* (two-party communication, see Bergstra and Tucker [1984]). The action set A and the communication function γ can be regarded as the parameters of the axiom system ACP defined by Bergstra and Klop [1984]. Henceforth we shall write $\mathsf{ACP}(A, \gamma)$. The signature of $\mathsf{ACP}(A, \gamma)$ is as follows:

sorts:	A	(a given, finite set of actions),
	$\mathcal{P}$	(the set of process terms; $A \subseteq \mathcal{P}$),
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),
	$\parallel : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(parallel composition or merge),
	$\between : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(left merge),
	$\mid : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(communication merge, given $\gamma : A \times A \to A$),
	$\partial_H : \mathcal{P} \to \mathcal{P}$	(encapsulation, $H \subseteq A$),
constants:	$\delta \in \mathcal{P} \setminus A$	(deadlock or inaction).

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. The left merge and the communication merge are auxiliary operations (allowing a finite axiomatization of the merge): $x \between y$ is as $x \parallel y$ with the restriction that the first action must stem from x, and $x \mid y$ is as $x \parallel y$, except that the first action must be a communication between x and y. Finally, encapsulation can be used to enforce communications between parallel components (this is illustrated by some examples in the sequel). Closed terms are further called *process terms*, in order to stress that these represent processes.

In Table I, the axioms of the system $\mathsf{ACP}(A, \gamma)$ are collected, where a ranges over $A_\delta = A \cup \{\delta\}$. Although the $\parallel$-operation is not axiomatized as an associative and commutative operation, it has these properties for all process terms (this can be proved with structural induction).

Example 2.1.1. As an example, assume $\gamma(a, b) = c$. Then one can derive in $\mathsf{ACP}(A, \gamma)$ that

$$\begin{aligned} a^2 \parallel b^2 &= a(a \parallel b^2) + b(b \parallel a^2) + c(a \parallel b) \\ &= a(a \cdot b^2 + b(b \parallel a) + cb) \\ &\quad + b(b \cdot a^2 + a(a \parallel b) + ca) \\ &\quad + c(ab + ba + c) \\ &= a(a \cdot b^2 + b(ba + ab + c) + cb) \\ &\quad + b(b \cdot a^2 + a(ab + ba + c) + ca) \\ &\quad + c(ab + ba + c). \end{aligned}$$

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 $\mathsf{BPA}_\delta(A)$. For a detailed introduction to $\mathsf{BPA}(A)$—$\mathsf{ACP}(A, \gamma)$ and an intuitive account see, for example, Baeten and Weijland [1990] and Fokkink [2000].

The binary Kleene star was added to process algebra by Bergstra et al. [1993] with the axioms given in Table II (see also Bergstra et al. [1994]). Bergstra and Ponse [2001] defined the recursive operation *push-down*, notation \$, by the single

TABLE I. THE AXIOMS OF $\mathsf{ACP}(A, \gamma)$ WHERE $a, b \in A_\delta$, $H \subseteq A$

(A1)	$x + y = y + x$	(CF1)	$a \mid b = \gamma(a, b)$	if $\gamma(a, b) \downarrow$
(A2)	$x + (y + z) = (x + y) + z$	(CF2)	$a \mid b = \delta$	otherwise
(A3)	$x + x = x$			
(A4)	$(x + y)z = xz + yz$	(CM1)	$x \parallel y = (x \between y + y \between x) + x \mid y$	
(A5)	$(xy)z = x(yz)$	(CM2)	$a \between x = ax$	
		(CM3)	$ax \between y = a(x \parallel y)$	
(A6)	$x + \delta = x$	(CM4)	$(x + y) \between z = x \between z + y \between z$	
(A7)	$\delta x = \delta$	(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)	$\partial_H(a) = a \quad \delta$ if $a \notin H$	
		(D2)	$\partial_H(a) = \delta \quad a$ 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 II. ADDITIONAL AXIOMS FOR BINARY KLEENE STAR AND PUSH-DOWN

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

axiom also displayed in Table II. Extension of one of the systems mentioned above with $*$ or \$ and relevant axioms is denoted by adding the appropriate symbol as a superscript, for example, $\mathsf{BPA}^*(A)$ 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 \$. We note that associativity and commutativity of $\parallel$ in process terms cannot be proved from the given axioms if $*$ or \$ is involved. However, since it is convenient to omit parentheses in (large) $\parallel$-expressions we further adopt the *notational convention* that $\parallel$ associates to the left, thus $x \parallel y \parallel z = (x \parallel y) \parallel z$.

2.2. OPERATIONAL SEMANTICS. We relate process terms to labeled 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, thus obtaining an operational semantics that captures process behavior in terms of the actions that can be executed.

A *labeled transition system* is a tuple $\langle S, L, \rightarrow, s\rangle$, where S is a set of *states*, L is a set of *labels*, $\rightarrow$ is a *transition relation*, and $s \in S$ is the *initial state* or *root*. Consider one of the process algebra axiom systems $\mathsf{BPA}(A) - \mathsf{ACP}^{*\$}(A, \gamma)$, and let $\mathcal{P}$ represent all process terms 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 actions from A. The transition relation $\rightarrow$ contains transitions

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

TABLE III. TRANSITION RULES FOR $\mathsf{BPA}(A) - \mathsf{ACP}^{*\$}(A, \gamma)$ WHERE $a, b \in A$, $H \subseteq A$

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

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

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

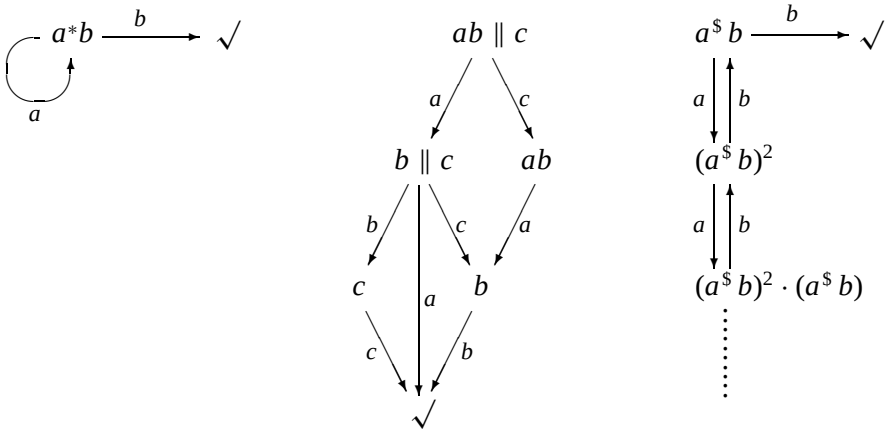
$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{\begin{array}{l} x \parallel y \xrightarrow{\gamma(a,b)} y' \text{ if } \gamma(a,b) \downarrow \\ x \mid y \xrightarrow{\gamma(a,b)} y' \text{ if } \gamma(a,b) \downarrow \end{array}} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b}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

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

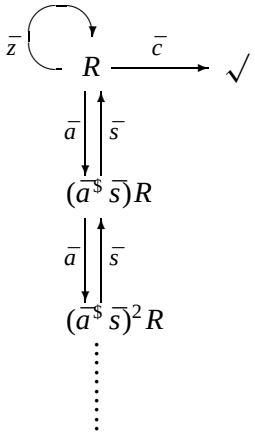
$$\_ \xrightarrow{\cdot} \surd \subseteq \mathcal{P} \times A. \quad (\surd \text{ is pronounced "tick".})$$

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 III define the transition relation $\rightarrow$, 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. If $\mathcal{P}$ is fixed and no confusion can arise, we often write P for $\langle \mathcal{P}, L, \rightarrow, P \rangle$, so the labeled transition system related to a process term P has P itself as initial state, and each state that can be reached from P via a sequence of transitions is called *a substate of* P. The *transition system of* P consists of P and all transitions that can be reached from P.

Example 2.2.1. Consider actions a, b, c. As a first example, the transition system of $a^* b$ as defined by the rules in Table III is displayed below (and also in the Introduction). Also transition systems of $ab \parallel c$ and $a^{\$} b$ are displayed below, where it is assumed that $\gamma(b, c) = a$ is the only communication defined.



The following process plays a major role in the sequel of the paper. Consider actions $\{\overline{a}, \overline{s}, \overline{z}, \overline{c}\}$ and process term $(\overline{a}(\overline{a}^{\$}\,\overline{s}) + \overline{z})^*\overline{c}$ abbreviated by R. The process term R can be recognized as a *register*, that is, a memory location for a natural number with unbounded capacity and restricted access as modeled by the specific actions: $\overline{a}$ for "add one," $\overline{s}$ for "subtract one," $\overline{z}$ for "test zero," and $\overline{c}$ for "clear, terminate the process." The transition system of R is visualized below.



Labeled transition systems are too concrete to represent processes. For example, process terms $a^*\delta$ and $(aa)^*\delta$ clearly represent the same process, that is, the process that repeatedly executes action a, but their transition systems are different (nonisomorphic). Therefore we consider bisimulation equivalence [Park 1981] over transition systems, which is the largest equivalence relation that respects all behavioral properties captured by process terms: two bisimilar processes cannot be distinguished in terms of observability.

Definition 2.2.2. A *bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{P}$ that satisfies the following conditions:

—if $P\mathcal{R}Q$ and $P \xrightarrow{a} P'$ for some $a \in A$ and $P' \in \mathcal{P}$, then there exists $Q' \in \mathcal{P}$ such that $Q \xrightarrow{a} Q'$ and $P'\mathcal{R}Q'$,

—if $P\mathcal{R}Q$ and $Q \xrightarrow{a} Q'$ for some $a \in A$ and $Q' \in \mathcal{P}$, then there exists $P' \in \mathcal{P}$ such that $P \xrightarrow{a} P'$ and $P'\mathcal{R}Q'$,
—if $P\mathcal{R}Q$ then for all $a \in A$, $P \xrightarrow{a} \surd$ if and only if $Q \xrightarrow{a} \surd$.

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

Note that $\leftrightarroweq$ is an equivalence relation. Now if we take $\mathcal{P}$ as the set of $\mathsf{ACP}^{*\$}(A, \gamma)$ terms, it follows that $\leftrightarroweq$ is a congruence relation for all operations involved [Baeten and Verhoef 1993; Groote and Vaandrager 1992]. We write $\mathsf{ACP}^{*\$}(A, \gamma)/\underset{\leftrightarroweq}{} \models P = Q$ whenever $P \leftrightarroweq Q$ according to the notions just defined, and for variable sequence $\vec{x} = x_1, \ldots, x_n$ we write

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

if for all $\vec{P} = P_1, \ldots, P_n$ it holds that $\mathsf{ACP}^{*\$}(A, \gamma)/\underset{\leftrightarroweq}{} \models t_1(\vec{P}) = t_2(\vec{P})$. It is not difficult to establish that in the bisimulation model thus obtained all equations of Tables I and II are true. So we have the following result:

LEMMA 2.2.3. *The system* $\mathsf{ACP}^{*\$}(A, \gamma)$ *is sound with respect to bisimulation equivalence*: *if* $\mathsf{ACP}^{*\$}(A, \gamma) \vdash t_1(\vec{x}) = t_2(\vec{x})$, *then* $\mathsf{ACP}^{*\$}(A, \gamma)/\underset{\leftrightarroweq}{} \models t_1(\vec{x}) = t_2(\vec{x})$.

Finally, the axioms of $\mathsf{ACP}(A, \gamma)$ completely characterize bisimilarity between the processes that can be expressed [Bergstra and Klop 1984; Baeten and Weijland 1990]. Moreover, bisimilarity over $\mathsf{BPA}^*(A)$ is completely axiomatized by the axioms of $\mathsf{BPA}(A)$ (i.e., (A1)–(A5)) and (BKS1)–(BKS3), as was first proved by Fokkink and Zantema [1994]. For an interesting decidability result on bisimulation equivalence, see Baeten et al. [1993].

2.3. ON THE EXPRESSIVENESS OF $\mathsf{ACP}^{*\$}(A, \gamma)$. Bergstra et al. [1994] showed that the expressiveness of systems with binary Kleene star can be analyzed using properties of cycles in labeled transition systems (these results were strengthened by Boselie [1995]). In order to show a negative expressivity result for $\mathsf{ACP}^{*\$}(A, \gamma)$, we adapt some of these results. A state $Q \in \mathcal{P}$ is a *successor* of state $P \in \mathcal{P}$ if $P \xrightarrow{a} Q$ for some $a \in A$. A *cycle* is a sequence of distinct states $(P_0, \ldots, P_n)$ such that P_{i+1} is a successor of P_i for $i = 0, \ldots, n-1$ and P_0 is a successor of P_n. An action a is an *exit action* of state P if $P \xrightarrow{a} \surd$.

LEMMA 2.3.1. *Let* C *be a cycle in a labeled transition system associated to a process term over* $\mathsf{ACP}^{*\$}(A, \gamma)$. *Then* C *has one of the following forms, for* $n \in \mathbb{N}$, *where* $\equiv$ *denotes syntactic equivalence*:

(i) $C \equiv (P_0 Q, P_1 Q, \ldots, P_n Q)$;
(ii) $C \equiv (P^* Q, P_1(P^* Q), \ldots, P_n(P^* Q))$, *or any cyclic permutation thereof*;
(iii) $C \equiv (P^{\$} Q, P_1(P^{\$} Q), \ldots, P_n(P^{\$} Q))$, *or any cyclic permutation thereof*;
(iv) $C \equiv (P_0 \parallel Q_0, P_1 \parallel Q_1, \ldots, P_n \parallel Q_n)$;
(v) $C \equiv (\partial_H(P_0), \partial_H(P_1), \ldots, \partial_H(P_n))$.

PROOF. Let $C \equiv (C_0, \ldots, C_n)$. We apply case distinction on C_0. Clearly C_0 is not a single action, and because $+$, $\between$, $|$ do not occur as the first operation in

right-hand sides of conclusions of transition rules, it follows that C_0 can not be a successor, so $C_0 \not\equiv P \diamond Q$ for $\diamond \in \{+, \between, |\}$ and five cases remain:

—$C_0 \equiv RS$. If S is not a state in C, then $C \equiv (RS, R_1S, \ldots, R_nS)$, which corresponds to case (i). If S is a state in C, then there is a sequence of transitions $S \xrightarrow{a_1} \cdots \xrightarrow{a_n} RS$. Observe that there are only three transition rules that can give rise to a transition $T \xrightarrow{a} T'$ where T is a proper subterm of T':

$$\frac{x \xrightarrow{a} \surd}{x^{\$}y \xrightarrow{a} (x^{\$}y)(x^{\$}y)} \quad \frac{x \xrightarrow{a} x'}{x^*y \xrightarrow{a} x'(x^*y)} \quad \frac{x \xrightarrow{a} x'}{x^{\$}y \xrightarrow{a} x'((x^{\$}y)(x^{\$}y))}$$

This implies that S is of the form P^*Q or $P^{\$}Q$, and that C must be of the form (ii) or (iii).

—$C_0 \equiv R^*S$. Analogous to the case $C_0 \equiv RS$, we see that C is of form (ii).

—$C_0 \equiv R^{\$}S$. Analogous to the case $C_0 \equiv RS$, we see that C is of form (iii).

—$C_0 \equiv R \between S$. As $R \between S$ is not a substate of R or S, it follows from the transition rules for the merge that C must be of form (iv).

—$C_0 \equiv \partial_H(R)$. Since

$$\frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x') \text{ if } a \notin H}$$

is the only transition rule for ∂_H that can have been used, it follows that C is of form (v). □

Lemma 2.3.1 can be used to derive further properties of cycles.

LEMMA 2.3.2. *Let C be a cycle in a labeled transition system associated to a process term over $\mathsf{ACP}^{*\$}(A, \gamma)$. Then there is at most one state in C with an exit action.*

PROOF. Cycle $C = (C_0, \ldots, C_n)$ must be of one of the forms (i)–(v) from Lemma 2.3.1. We apply induction with respect to the size of C.

—$C = (P_0Q, \ldots, P_nQ)$. Then none of the states in C has an exit action.

—$C = (P^*Q, P_1(P^*Q), \ldots, P_n(P^*Q))$, or any cyclic permutation thereof. Then P^*Q is the only state in C that may have an exit action.

—$C = (P^{\$}Q, P_1(P^{\$}Q), \ldots, P_n(P^{\$}Q))$, or any cyclic permutation thereof. Then $P^{\$}Q$ is the only state in C that may have an exit action.

—$C = (P_0 \between Q_0, \ldots, P_n \between Q_n)$. By induction, there is at most one $i \in \{0, \ldots, n\}$ such that both P_i and Q_i have an exit action (note that only one of $(P_0, \ldots, P_n)$, $(Q_0, \ldots, Q_n)$ necessarily is a cycle). So $P_i \between Q_i$ is the only state in C that may have an exit action.

—$C = (\partial_H(P_0), \ldots, \partial_H(P_n))$. By induction, the cycle $(P_0, \ldots, P_n)$ contains at most one state P_i that has an exit action. So $\partial_H(P_i)$ is the only state in C that may have an exit action. □

Let $a, b \in A$. We now argue that the regular process p recusively defined by

$$\mathsf{p} = a\mathsf{q} + b,$$
$$\mathsf{q} = a\mathsf{p} + a,$$

(also considered in the Introduction) cannot be defined in $\mathsf{ACP}^{*\$}(A,\gamma)$ modulo strong bisimulation equivalence. More precisely, given transitions $\mathsf{p} \xrightarrow{a} \mathsf{q}$, $\mathsf{p} \xrightarrow{b} \surd$, $\mathsf{q} \xrightarrow{a} \mathsf{p}$ and $\mathsf{q} \xrightarrow{a} \surd$ there does not exist a process term R in $\mathsf{ACP}^{*\$}(A,\gamma)$ that satisfies $R \leftrightarrows \mathsf{p}$, where $\leftrightarrows$ is defined over $\mathcal{P} \cup \{\mathsf{p}, \mathsf{q}\}$ and $\mathcal{P}$ is the set of $\mathsf{ACP}^{*\$}(A,\gamma)$ process terms. For assume the contrary: according to Lemma 2.3.2, each process term yielding a cycle is not a candidate because each cycle with a state bisimilar to p contains at least two states, and has at least two exit actions. So, the transition system associated to process term R necessarily has an infinite number of states. (Cf. $a^{\$}\,\delta$ that has an infinite number of states and no cycles, and which is bisimilar to $a^*\delta$.) This implies that R contains an occurrence of \$, which contributes to R's transition system by a transition, say $T \xrightarrow{a} T'$, that is derived with one of the rules introducing \$ via its left-argument. Because $+$, $\between$,$|$ do not occur as the first operation in right-hand sides of conclusions of transition rules, this implies that T' cannot have an exit action, which contradicts $R \leftrightarrows \mathsf{p}$. Hence, p cannot be defined in $\mathsf{ACP}^{*\$}(A,\gamma)$.

Remark 2.3.3. With a little more effort we can show that the regular process r defined by $\mathsf{r} = aa\mathsf{s} + a$, $\mathsf{s} = a\mathsf{r} + a$ cannot be specified in $\mathsf{ACP}^{*\$}(A,\gamma)$ for any choice of $A \supseteq \{a\}$ (cf. Boselie [1995]).

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

In this section we turn to register machines, and establish a process algebraic representation of register machine computation for a particular repertoire of actions and handshake communications. Having this, it easily follows that the resulting theory is undecidable.

3.1. Alphabets, Registers, and Register Machine Programs.

We define a setting in which register machine computation is straightforwardly modeled in $\mathsf{ACP}^{*\$}(A,\gamma)$ for a particular choice of A and γ. We consider registers modeled by process terms as in Example 2.2.1: a register named $i \in \mathbb{N}$ is modeled as a process R_i over alphabet

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

by $R_i = (\overline{a}_i(\overline{a}_i{}^{\$}\,\overline{s}_i) + \overline{z}_i)^*\overline{c}_i$. Furthermore, we define for $j \in \mathbb{N}$ the following abbreviations:

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

So $R_i(j)$ represents register i containing value j. Rather than viewing registers as autonomous processes, we want them to be controlled by a *register machine iterative program*. To this end, we define for $i \in \mathbb{N}$ the alphabet

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

containing actions that represent instructions to register i. Starting from alphabets α_i, we define a class of process terms representing structured register machine programs.

Definition 3.1.1. For $i \in \mathbb{N}$ the Register Machine Iterative Programs using registers $0, 1, \ldots, i-1$, notation

$$\mathsf{RMI}(i),$$

is a collection of process terms with alphabet in $\cup_{k<i}\alpha_k$. The class $\mathsf{RMI}(i)$ is inductively defined by the following clauses:

(1) $a_k \in \mathsf{RMI}(i)$ if $k < i$,
(2) $s_k{}^* z_k \in \mathsf{RMI}(i)$ if $k < i$,
(3) $s_k{}^* c_k \in \mathsf{RMI}(i)$ if $k < i$,
(4) if $P \in \mathsf{RMI}(i)$ and $k < i$, then $(s_k P)^* z_k \in \mathsf{RMI}(i)$,
(5) if $P, Q \in \mathsf{RMI}(i)$, then $PQ \in \mathsf{RMI}(i)$,
(6) if $P, Q \in \mathsf{RMI}(i)$ and $k < i$, then $s_k P + z_k Q \in \mathsf{RMI}(i)$.

Note that if P is an element of $\mathsf{RMI}(i)$, then also each successor of P. Furthermore, $\mathsf{RMI}(i)$ is closed under associativity of sequential composition (and we will omit brackets in repeated applications). Finally, note that each $P \in \mathsf{RMI}(i)$ specifies a *deterministic* process (i.e., in the case that there are two outgoing transitions, these have different labels).

Let t be an action disjoint from $\cup_i(\alpha_i \cup \overline{\alpha}_i)$. For $n \in \mathbb{N}$, we distinguish the following sets of actions:

$$A_n = \{t\} \cup H_n,$$
$$H_n = \bigcup_{j<n}(\alpha_j \cup \overline{\alpha}_j).$$

The sets H_n will be used for encapsulation, thus enforcing communications between parallel components. Communication on A_n is defined by $\gamma(a, b) = t$ 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 $a = \overline{b}$.

Encapsulated parallel composition of a register machine iterative program together with the registers it addresses will be used to model register machine computation by synchronization: the actions of an RMI process term perform handshaking communications with the registers addressed. For example we can derive in $\mathsf{ACP}^{*\$}(A_1, \gamma)$ that

$$\begin{aligned}\partial_{H_1}(s_0{}^* z_0 \parallel R_0(j)) &= \partial_{H_1}(s_0{}^* z_0 \between R_0(j)) + \partial_{H_1}(R_0(j) \between s_0{}^* z_0) \\ &\quad + \partial_{H_1}(s_0{}^* z_0 \mid R_0(j)) \\ &= \partial_{H_1}((s_0(s_0{}^* z_0) + z_0) \between R_0(j)) + \partial_{H_1}(R_0(j) \between s_0{}^* z_0) \\ &\quad + \partial_{H_1}(s_0(s_0{}^* z_0) \mid R_0(j)) + \partial_{H_1}(z_0 \mid R_0(j)).\end{aligned}$$

By encapsulation the first two summands equal δ. In case $j = 0$, the third summand equals δ and the fourth equals $t \cdot \partial_{H_1}(R_0(0))$ (because $\gamma(z_0, \overline{z}_0) = t$). In the case that $j > 0$, the third summand equals $t \cdot \partial_{H_1}(s_0{}^* z_0 \parallel R_0(j-1))$ (by an $s_0 \mid \overline{s}_0$ communication), and the last summand equals δ. So in general,

$$\partial_{H_1}(s_0{}^* z_0 \parallel R_0(j)) = t^{j+1} \cdot \partial_{H_1}(R_0(0)).$$

Because we often consider a number of registers operating in parallel, we introduce for $k > 0$ the abbreviation

$$\mathcal{R}_k \text{ defined by } \mathcal{R}_1 = R_0,\ \mathcal{R}_{k+1} = \mathcal{R}_k \parallel R_k.$$

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

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

As it turns out, $\mathsf{ACP}^{*\$}(A_k, \gamma)$, in which $\mathcal{R}_k$ and $\mathsf{RMI}(k)$ processes are specifiable, 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 $k \geq 4$. The cases $k = 1, 2, 3$ may feature various anomalies due to a lack of expressive power, and are of no concern to us.

3.2. REGISTER MACHINE PROGRAMMING IN $\mathsf{ACP}^{*\$}(A, \gamma)$. In the sequel we will often consider expressions of the form

$$\partial_{H_k}(Px \parallel \mathcal{R}_k(n_0, \ldots, n_{k-1}))$$

abbreviating $\partial_{H_k}(Px \parallel [\cdots(R_0(n_0) \parallel R_1(n_1)) \ldots \parallel R_{k-1}(n_{k-1})])$ with P in $\mathsf{RMI}(k)$ and x a process variable. The following basic result states that we can 'implement' each computable function in $\mathsf{ACP}^{*\$}(A_4, \gamma)$.

THEOREM 3.2.1. *Let $f : \mathbb{N} \to \mathbb{N}$ be computable* (*not necessarily total*). *There exist $P \in \mathsf{RMI}(4)$ and computable $g : \mathbb{N} \to \mathbb{N}\backslash\{0\}$ such that if $f(n)$ is defined, then $g(n)$ is defined and*

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

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

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

PROOF. Consider a register machine programming language with instructions 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 holds value zero, then go to instruction l', otherwise subtract 1 from register i and go to instruction l.

Let $\bar{P}$ be a register machine program that computes f using three registers 1, 2, 3, and instructions numbered $1, \ldots, k$: if the tuple $\langle x_1, x_2, x_3 \rangle$ represents the values of registers 1, 2, 3, respectively, and $\bar{P}$ started with instruction 1 on machine state $\langle n, 0, 0 \rangle$ terminates (i.e., has reached a halt-instruction), then this termination state is $\langle f(n), 0, \underline{0} \rangle$. (This is possible; see e.g., Minsky [1967] or Appendix A.)

We turn $\bar{P}$ into a process P in $\mathsf{RMI}(4)$, taking an extra register process R_0 to store "the next instruction number." We set

$$P = L_1 Q, \quad Q = (s_0 P_1)^* z_0,$$

and for $line(m, \bar{P})$ denoting the m_{th} instruction of $\bar{P}$,

$$P_m = \begin{cases} L_m & \text{if } k = 1, \\ s_0 P_{m+1} + z_0 L_m & \text{if } k > 1 \text{ and } m+1 < k, \\ s_0 L_{m+1} + z_0 L_m & \text{if } k > 1 \text{ and } m+1 = k, \end{cases}$$

$$L_m = \begin{cases} (s_3{}^* z_3)(s_2{}^* z_2)(s_0{}^* z_0) & \text{if } line(m, \bar{P}) = \mathsf{halt}, \\ a_i \cdot a_0^l & \text{if } line(m, \bar{P}) = (a_i, l), \\ s_i \cdot a_0^l + z_i \cdot a_0^{l'} & \text{if } line(m, \bar{P}) = (s_i, l, l'). \end{cases}$$

Starting from the initial state

$$\partial_{H_4}(L_1 Qx \parallel \mathcal{R}_4(0, n, 0, 0)), \text{ that is, } \partial_{H_4}(L_1 Qx \parallel (R_0 \parallel R_1(n) \parallel R_2 \parallel R_3)),$$

we show that each 'program state' $L_m Qx$, that is, the state that models execution of instruction m, occurs in a pattern of the form

$$\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)),$$

and results in an appropriate update of the register values and the "program state." We apply case distinction on $line(m, \bar{P})$:

—$line(m, \bar{P}) = \mathsf{halt}$. In this case the registers R_0, R_2 and R_3 must be emptied (set to value 0) and $L_m Q$ must be terminated. We derive

$$\begin{aligned} &\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)) \\ &\quad = \partial_{H_4}((s_3{}^* z_3)(s_2{}^* z_2)(s_0{}^* z_0) Qx \parallel (R_0 \parallel R_1(x_1) \parallel R_2(x_2) \parallel R_3(x_3))) \\ &\quad = t^{x_3+1} \cdot \partial_{H_4}((s_2{}^* z_2)(s_0{}^* z_0) Qx \parallel (R_3 \parallel (R_0 \parallel R_1(x_1) \parallel R_2(x_2)))) \\ &\quad = t^{x_2+x_3+2} \cdot \partial_{H_4}((s_0{}^* z_0) Qx \parallel ((R_2 \parallel (R_0 \parallel R_1(x_1))) \parallel R_3)) \\ &\quad = t^{x_2+x_3+3} \cdot \partial_{H_4}(Qx \parallel (R_0 \parallel R_1(x_1) \parallel R_2 \parallel R_3)) \\ &\quad = t^{x_2+x_3+4} \cdot \partial_{H_4}(x \parallel (R_0 \parallel R_1(x_1) \parallel R_2 \parallel R_3)) \\ &\quad = t^{x_2+x_3+4} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0, x_1, 0, 0)). \end{aligned}$$

—$line(m, \bar{P}) = (a_1, l)$ (recall $1 \le l \le k$). In this case we derive

$$\begin{aligned} &\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)) \\ &\quad = \partial_{H_4}\big(a_1 a_0^l Qx \parallel (R_0 \parallel R_1(x_1) \parallel R_2(x_2) \parallel R_3(x_3))\big) \\ &\quad = t \cdot \partial_{H_4}\big(a_0^l Qx \parallel (R_1(x_1+1) \parallel R_0 \parallel R_2(x_2) \parallel R_3(x_3))\big) \\ &\quad = t^{l+1} \cdot \partial_{H_4}(Qx \parallel (R_0(l) \parallel R_1(x_1+1) \parallel R_2(x_2) \parallel R_3(x_3))) \\ &\quad = t^{2l+2} \cdot \partial_{H_4}(L_l Qx \parallel (R_0 \parallel R_1(x_1+1) \parallel R_2(x_2) \parallel R_3(x_3))) \\ &\quad = t^{2l+2} \cdot \partial_{H_4}(L_l Qx \parallel \mathcal{R}_4(0, x_1+1, x_2, x_3)). \end{aligned}$$

In case $line(m, \bar{P}) = (a_i, l)$ for $i = 2, 3$ it follows in a similar way that

$$\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)) = t^{2l+2} \cdot \partial_{H_4}(L_l Qx \parallel \mathcal{R}_4(0, x_1, y_2, y_3)),$$

where $y_i = x_i + 1$ and $y_{5-i} = x_{5-i}$.

—$line(m, \bar{P}) = (s_i, l, l')$. Now there are two cases to distinguish depending on the value x_i of the current machine configuration $\langle x_1, x_2, x_3 \rangle$. If $x_i = 0$ it follows that

$$\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)) = t^{2l'+2} \cdot \partial_{H_4}(L_l Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)).$$

If $x_i > 0$ and s_i modifies the machine configuration $\langle x_1, x_2, x_3\rangle$ into $\langle y_1, y_2, y_3\rangle$ where $y_i = x_i - 1$ and $y_j = x_j$ for $j \neq i$, then

$$\partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, x_1, x_2, x_3)) = t^{2l+2} \cdot \partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, y_1, y_2, y_3)).$$

Based upon the number of t-steps computed above we now provide a definition of function g, applying induction on the length of terminating computations. Let the auxiliary functions $F_m(x_1, x_2, x_3)$ for $m = 1, \ldots, k$ be defined as follows:

—if $L_m = (s_3{}^*z_3)(s_2{}^*z_2)(s_0{}^*z_0)$, i.e., the encoding of halt, then $F_m(x_1, x_2, x_3) = x_2 + x_3 + 4$,

—if $L_m = a_i a_0^l$, i.e., the encoding of instruction (a_i, l), then $F_m(x_1, x_2, x_3) = F_l(y_1, y_2, y_3) + 2l + 2$, where $y_i = x_i + 1$ and $y_j = x_j$ for $j \neq i$,

—if $L_m = s_i a_0^l + z_i a_0^{l'}$, i.e., the encoding of instruction (s_i, l, l'), then

$$F_m(x_1, x_2, x_3) = \begin{cases} F_l(y_1, y_2, y_3) + 2l + 2 & \text{if } x_i > 0,\ y_i = x_i - 1 \text{ and} \\ & y_j = x_j \text{ for } j \neq i, \\ F_{l'}(x_1, x_2, x_3) + 2l' + 2 & \text{otherwise.} \end{cases}$$

Observe that the F_m are not necessarily *total*, even if f is. However, if $\bar{P}$ started at line m with register values $n, m2, m3$ computes to halt with register values $n', 0, 0$ then

$$\begin{aligned} &\mathsf{ACP}^{*\$}(A_4, \gamma) \vdash \partial_{H_4}(L_m Qx \parallel \mathcal{R}_4(0, n, m2, m3)) \\ &\quad = t^{F_m(n,m2,m3)} \cdot \partial_{H_4}(x \parallel \mathcal{R}_4(0, n', 0, 0)). \end{aligned}$$

This follows by induction on the length of (terminating) computations, say h. If $h = 0$ then $line(m, \bar{P}) = \text{halt}$ and $L_m = (s_3{}^*z_3)(s_2{}^*z_2)(s_0{}^*z_0)$. If $h = h' + 1$, then L_m is either $a_i a_0^l$ or $s_i a_0^l + z_i a_0^{l'}$ for some i, l, l'. The identities above suffice to make the induction step. Taking $m = 1$ and $m2 = m3 = 0$ yields the required information: $g(n) = F_1(n, 0, 0)$. Clearly, if $f(n)$ is defined, then so is $g(n)$ and $g(n) > 0$.

In the case that P started at line m with certain register values does not halt (so the performance of successive instructions is perpetual), the second statement of the lemma follows immediately. □

In Section 4 we will use the following generalizations of this result.

COROLLARY 3.2.2. *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function (not necessarily total). Then there exist $P, Q \in \mathsf{RMI}(5)$ such that for some computable functions $g, h : \mathbb{N} \to \mathbb{N}\backslash\{0\}$ and for all n, if $f(n)$ is defined, then so are $g(n)$ and $h(n)$, and*

$$\begin{aligned} &\mathsf{ACP}^{*\$}(A_5, \gamma) \vdash \partial_{H_5}(Px \parallel \mathcal{R}_5(0, n, 0, 0, 0)) = t^{g(n)} \cdot \partial_{H_5}(x \parallel \mathcal{R}_5(0, f(n), 0, 0, 0)), \\ &\mathsf{ACP}^{*\$}(A_5, \gamma) \vdash \partial_{H_5}(Qx \parallel \mathcal{R}_5(0, n, 0, 0, 0)) = t^{h(n)} \cdot \partial_{H_5}(x \parallel \mathcal{R}_5(0, n, f(n), 0, 0)). \end{aligned}$$

PROOF. The first statement follows immediately from the previous proof. Furthermore, let $Q_1 = ((s_1 a_2 a_0)^* z_1)((s_0 a_1)^* z_0)$, so $Q_1 \in \mathsf{RMI}(5)$. It follows easily that

$$\mathsf{ACP}^{*\$}(A_5, \gamma) \vdash \partial_{H_5}(Q_1 x \parallel \mathcal{R}_4(0, n, 0, 0))) = t^{5n+2} \cdot \partial_{H_5}(x \parallel \mathcal{R}_5(0, n, n, 0, 0)).$$

Let $\bar{Q}_2$ be a register machine program that computes f using registers 2, 3 and 4. As shown in the proof of Theorem 3.2.1, there exist $Q_2 \in \mathsf{RMI}(5)$ and computable

function $h' : \mathbb{N} \to \mathbb{N}\backslash\{0\}$ such that if $f(n)$ is defined, then

$$\begin{aligned}\mathsf{ACP}^{*\$}(A_5, \gamma) \vdash \partial_{H_5}(Q_2 x \parallel (R_0 \parallel R_2(n) \parallel R_3 \parallel R_4)) \\ = t^{h'(n)} \cdot \partial_{H_5}(x \parallel (R_0 \parallel R_2(f(n)) \parallel R_3 \parallel R_4))\end{aligned}$$

if the modeling of the halt instruction is adapted to $(s_4{}^* z_4)(s_3{}^* z_3)(s_0{}^* z_0)$. It follows that if $f(n)$ is defined, then

$$\begin{aligned}\mathsf{ACP}^{*\$}(A_5, \gamma) \vdash \partial_{H_5}(Q_1 Q_2 x \parallel \mathcal{R}_4(0, n, 0, 0))) \\ = t^{5n+2+h'(n)} \cdot \partial_{H_5}(x \parallel \mathcal{R}_5(0, n, f(n), 0, 0)).\end{aligned}$$

So, setting $Q = Q_1 Q_2$ and $h(n) = 5n + 2 + h'(n)$ proves the second statement of the lemma. □

3.3. UNDECIDABILITY OF $\mathsf{ACP}^{*\$}(A, \gamma)$. It being possible to represent each computable function in $\mathsf{ACP}^{*\$}(A_4, \gamma)$, it is not difficult to prove that $\mathsf{ACP}^{*\$}(A_4, \gamma)$ has an undecidable theory (initial algebra). We provide a family of process terms U_n, V_n such that $\mathsf{ACP}^{*\$}(A_4, \gamma) \vdash U_n = V_n$ is not decidable.

THEOREM 3.3.1. $\mathsf{ACP}^{*\$}(A_4, \gamma) \vdash u = v$ *is not 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 Theorem 3.2.1 there are $P \in \mathsf{RMI}(4)$ and computable function g such that if $f(n)$ is defined, then

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

Now let $U, V \in \mathsf{RMI}(4)$ and U_n, V_n be defined by

$$\begin{aligned} U &= P(s_3{}^* c_3)(s_2{}^* c_2)(s_0{}^* c_0)(s_1{}^* c_1), \\ V &= P(s_3{}^* c_3)(s_2{}^* c_2)(s_0{}^* c_0)((s_1(s_1{}^* z_1))^* c_1), \\ U_n &= \partial_{H_4}(U \parallel \mathcal{R}_4(0, n, 0, 0)), \\ V_n &= \partial_{H_4}(V \parallel \mathcal{R}_4(0, n, 0, 0)). \end{aligned}$$

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

$$\begin{aligned} n \in W_{e_1} \Rightarrow f(n) = 0 \Rightarrow \vdash U_n = V_n \qquad & \left(= t^{g(n)+4}\right), \\ n \in W_{e_2} \Rightarrow f(n) = 1 \Rightarrow \nvdash U_n = V_n \quad & \left(t^{g(n)+5} \neq t^{g(n)+6}\right). \end{aligned}$$

As to the latter implication: assume otherwise, that is, $\mathsf{ACP}^{*\$}(A_4, \gamma) \vdash t^k = t^{k+1}$ for some $k > 0$. Then by Lemma 2.2.3, $t^k \leftrightarrows t^{k+1}$, which clearly is a contradiction.

Thus, decidability of $\mathsf{ACP}^{*\$}(A_4, \gamma) \vdash U_n = V_n$ 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 $\mathsf{ACP}^{*\$}(A, \gamma)$ *left cancelling* or *trace consistent* if for all $a, b \in A$,

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

TABLE IV. 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)	$\tau_I(xy) = \tau_I(x) \cdot \tau_I(y)$	

The following corollary is a straightforward consequence from the proof given above:

COROLLARY 3.3.2. *Let $A_4 \subseteq A$ and let $\mathcal{M}$ be a model of $\mathsf{ACP}^{*\$}(A,\gamma)$ that is left canceling. Then the word problem of $\mathcal{M}$ is undecidable.*

Remark 3.3.3. In the undecidability results above, we use 33 actions ($|A_4| + |\{t\}|$) and the restriction to handshaking communication. Retaining this last restriction, we can reduce this number: replace all c_i actions by z_i in U, V and use $U\delta$, $V\delta$ instead, and replace all $\overline{c_i}$ by δ. So for A containing at least 25 actions it follows that $\mathsf{ACP}^{*\$}(A,\gamma)$ has an undecidable theory.

4. *Adding Abstraction*

A basic ingredient of concurrency theory is the *silent* or *internal* action or *hidden move,* notation τ, which dates back to Milner [1980]. 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, that is, 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, serving both verification styles and expressive power. In this section we show that the addition of abstraction yields a substantial increase of expressive power. Furthermore, we show that the use of $*$ in all our results can be avoided.

4.1. AXIOM SYSTEMS AND OPERATIONAL SEMANTICS. Bergstra and Klop [1985] defined the system $\mathsf{ACP}_\tau(A,\gamma)$. This system extends $\mathsf{ACP}(A,\gamma)$ with a constant τ and abstraction operators $\tau_I(\_)$ renaming 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 IV. These axioms characterize *rooted τ-bisimilarity* (explained below). 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 Baeten and Weijland [1990], based on van Glabbeek and Weijland [1989]). Its additional axioms are given in Table V. 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, for example, Baeten and Weijland [1990] and Fokkink [2000].

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

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

Extension of one of the systems mentioned above with $*$ or \$ and relevant axioms is denoted by adding the appropriate symbol as a superscript, for example,

TABLE V. 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)	$\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$	

TABLE VI. ADDITIONAL TRANSITION RULES FOR $\mathsf{ACP}_{\tau}(A,\gamma)$ AND $\mathsf{ACP}^{\tau}(A,\gamma)$ WHERE $a \in A_{\tau}$

$\tau \xrightarrow{\tau} \surd$	$\dfrac{x \xrightarrow{a} \surd}{\begin{array}{ll}\tau_I(x) \xrightarrow{a} \surd & \text{if } a \notin I \\ \tau_I(x) \xrightarrow{\tau} \surd & \text{if } a \in I\end{array}}$	$\dfrac{x \xrightarrow{a} x'}{\begin{array}{ll}\tau_I(x) \xrightarrow{a} \tau_I(x') & \text{if } a \notin I \\ \tau_I(x) \xrightarrow{\tau} \tau_I(x') & \text{if } a \in I\end{array}}$

$\mathsf{ACP}^{\tau *\$}(A,\gamma)$ stands for $\mathsf{ACP}^{\tau}(A,\gamma)$ extended with $*$ and \$, the axioms (BKS1)–(BKS5), and the defining axiom for \$.

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 III where a now ranges over $A_{\tau} = A \cup \{\tau\}$, and those in Table VI.

In order to formulate general expressivity results, we define both rooted τ-bisimilarity and rooted branching bisimilarity for transition systems of which the states are not necessarily process terms. Let $\mathcal{Q} \cup \{\surd\}$ be the set of states under consideration with $\surd \notin \mathcal{Q}$ the only terminal state, and let a transition relation $\rightarrow \subseteq \mathcal{Q} \times A_{\tau} \times \mathcal{Q} \cup \{\surd\}$ be given. We first provide definitions and then 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 $P \stackrel{a}{\Rightarrow} P'$ if $P \Rightarrow Q \xrightarrow{a} Q' \Rightarrow P'$, and $P \stackrel{a}{\Rightarrow} \surd$ if either $P \Rightarrow Q \xrightarrow{a} Q' \Rightarrow \surd$ or $P \Rightarrow Q \xrightarrow{a} \surd$ for some $Q, Q' \in \mathcal{Q}$.

Definition 4.1.1. A *τ-bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{Q} \cup \{\surd\}$ that satisfies the following conditions:

—if $P\mathcal{R}Q$ and $P \xrightarrow{a} P'$ for some $a \in A_{\tau}$ and $P' \in \mathcal{Q} \cup \{\surd\}$, then either $a = \tau$ and $P'\mathcal{R}Q$, or there exists $Q' \in \mathcal{Q} \cup \{\surd\}$ such that $Q \stackrel{a}{\Rightarrow} Q'$ and $P'\mathcal{R}Q'$,

—if $P\mathcal{R}Q$ and $Q \xrightarrow{a} Q'$ for some $a \in A_{\tau}$ and $Q' \in \mathcal{Q} \cup \{\surd\}$, then either $a = \tau$ and $P\mathcal{R}Q'$, or there exists $P' \in \mathcal{Q} \cup \{\surd\}$ such that $P \stackrel{a}{\Rightarrow} P'$ and $P'\mathcal{R}Q'$,

—if $\surd\mathcal{R}Q$ then either $Q \equiv \surd$ or $Q \Rightarrow \surd$,

—if $P\mathcal{R}\surd$ then either $P \equiv \surd$ or $P \Rightarrow \surd$.

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

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

—if $P \xrightarrow{\tau} P'$ for some $P' \in \mathcal{Q} \cup \{\surd\}$, then there exists $Q' \in \mathcal{Q} \cup \{\surd\}$ such that $Q \xrightarrow{\tau} Q'$ and $P' \mathcal{R} Q'$, and

—if $Q \xrightarrow{\tau} Q'$ for some $Q' \in \mathcal{Q} \cup \{\surd\}$, then there exists $P' \in \mathcal{Q} \cup \{\surd\}$ such that $P \xrightarrow{\tau} P'$ and $P' \mathcal{R} Q'$.

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

Observe that the relations $\leftrightarrows_\tau$ and $\leftrightarrows_{r\tau}$ are equivalence relations.

Definition 4.1.2. A *branching bisimulation* is a binary relation $\mathcal{R}$ over $\mathcal{Q} \cup \{\surd\}$ that satisfies the following conditions:

—if $P \mathcal{R} Q$ and $P \xrightarrow{a} P'$ for some $a \in A_\tau$ and $P' \in \mathcal{Q} \cup \{\surd\}$, then either $a = \tau$ and $P' \mathcal{R} Q$, or there are $Q' \in \mathcal{Q} \cup \{\surd\}$ and $Q'' \in \mathcal{Q}$ such that $Q \Rightarrow Q'' \xrightarrow{a} Q'$, $P \mathcal{R} Q''$, and $P' \mathcal{R} Q'$,

—if $P \mathcal{R} Q$ and $Q \xrightarrow{a} Q'$ for some $a \in A_\tau$ and $Q' \in \mathcal{Q} \cup \{\surd\}$, then either $a = \tau$ and $P \mathcal{R} Q'$, or there are $P' \in \mathcal{Q} \cup \{\surd\}$ and $P'' \in \mathcal{Q}$ such that $P \Rightarrow P'' \xrightarrow{a} P'$, $P'' \mathcal{R} Q$, and $P' \mathcal{R} Q'$,

—if $\surd \mathcal{R} Q$ then either $Q \equiv \surd$ or $Q \Rightarrow \surd$,

—if $P \mathcal{R} \surd$ then either $P \equiv \surd$ or $P \Rightarrow \surd$.

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

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

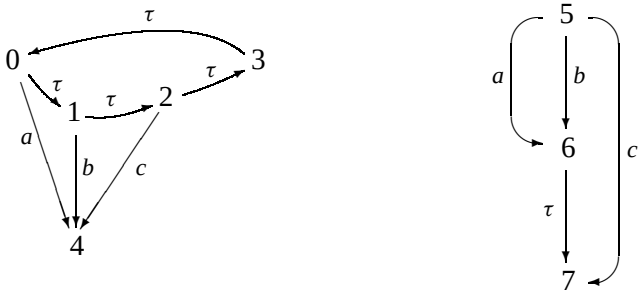
—if $P \xrightarrow{a} P'$ for some $a \in A_\tau$ and $P' \in \mathcal{Q} \cup \{\surd\}$, then there exists $Q' \in \mathcal{Q} \cup \{\surd\}$ such that $Q \xrightarrow{a} Q'$ and $P' \mathcal{R} Q'$,

—if $Q \xrightarrow{a} Q'$ for some $a \in A_\tau$ and $Q' \in \mathcal{Q} \cup \{\surd\}$, then there exists $P' \in \mathcal{Q} \cup \{\surd\}$ such that $P \xrightarrow{a} P'$ and $P' \mathcal{R} Q'$.

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

Observe that also $\leftrightarrows_b$ and $\leftrightarrows_{rb}$ are equivalence relations. Furthermore, note that $\leftrightarrows_b \subseteq \leftrightarrows_\tau$ and $\leftrightarrows_{rb} \subseteq \leftrightarrows_{r\tau}$, and that these inclusions are strict (cf. simple instances of axioms (T3) and (T2), respectively).

Example 4.1.3. As an example, consider the following transition systems:



where states 4 and 7 both represent deadlock. It follows that $0 \leftrightarrows_b 1 \leftrightarrows_b 2 \leftrightarrows_b 3 \leftrightarrows_b 5$, but that in none of these cases $\leftrightarrows_{r\tau}$ (and thus $\leftrightarrows_{rb}$) holds. Furthermore, $4 \leftrightarrows_b 6 \leftrightarrows_b 7$.

The restriction to the rooted versions serves to ensure congruence properties. $\mathcal{Q}$ being the set of $\mathsf{ACP}_\tau(A, \gamma)$ terms, the relation $\leftrightarrows_{r\tau}$ is a congruence (but $\leftrightarrows_\tau$ is not, e.g., $a \leftrightarrows_\tau \tau a$ and $a + b \not\leftrightarrows_\tau \tau a + b$). Moreover, $\mathsf{ACP}_\tau^{*\$}(A, \gamma)$ is sound with respect to rooted τ-bisimilarity, and its fragment $\mathsf{ACP}_\tau(A, \gamma)$ is complete with respect to its process terms [Bergstra and Klop 1985; Baeten and Weijland 1990]. For $\mathsf{ACP}^\tau(A, \gamma)$ and $\mathsf{ACP}^{\tau*\$}(A, \gamma)$ there are similar results with respect to rooted branching bisimilarity. More information on (rooted) branching bisimulation equivalence can be found in van Glabbeek [1993]; Glabbeek and Weijland [1996]. Finally, note that bisimilar transition systems also are rooted τ-bisimilar and rooted branching bisimilar.

4.2. EXPRESSIVENESS RESULTS. Let $A_n(B) = A_n \cup B$, where B is a finite set of actions disjoint from A_i for any $i \in \mathbb{N}$. In this section we first prove that each *computable process* over action set B (this notion is explained below) can be expressed in $\mathsf{ACP}^{\tau*\$}(A_5(B), \gamma)$, thus $\mathsf{ACP}^{*\$}(A_5(B), \gamma)$ equipped with abstraction and rooted branching bisimulation equivalence. This means that for any transition system T representing a computable process over B there exists a process term P over $\mathsf{ACP}^{\tau*\$}(A_5(B), \gamma)$ such that $T \leftrightarrows_{rb} P$. Then we prove that in $\mathsf{ACP}_\tau^{*\$}(A_5(B), \gamma)$, that is, $\mathsf{ACP}^{*\$}(A_5(B), \gamma)$ with abstraction and rooted τ-bisimilarity, a large class of semi-computable processes can be expressed: those that are initially finitely branching. Therefore one may say that $\mathsf{ACP}_\tau^{*\$}(A_5(B), \gamma)$ is *expressively complete*. 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.

We consider a transition system T with states in $\mathbb{N}$ and with label set B, in which 0 serves as initial state and 1 is the only terminal state (in particular, state 1 is assumed to have no outgoing transitions). Furthermore, $\{R_a \subseteq \mathbb{N} \times \mathbb{N} \mid a \in B\}$ is the transition relation. Adapting our further exposition to prior notation conventions, we shall often write

$$n \xrightarrow{a} m$$

instead of $R_a(n, m)$. Sometimes we shall use other values than 0 and 1 for the root and terminal state of a transition system: the notation $n^\frown m T$ makes explicit that n is the root of T and m the terminal state, thus $T = 0^\frown 1 T$.

Definition 4.2.1. Let $S \subseteq \mathbb{N}$ be a recursive set of states with $0 \in S$, and $T = \langle S, B, \{R_a \subseteq S \times S \mid a \in B\}, 0\rangle$ a transition system.

Then T is *recursive* if for some injective function $h : B \to \mathbb{N}$ and bijective pairing function $\langle \_, \_ \rangle : \mathbb{N}^2 \to \mathbb{N}$, 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[2] (CI) of the finite set encoding all next steps from s:

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

[2]The canonical index of $\emptyset$ is 0, of $\{k_1, k_2, \ldots, k_l\}$ it is the number $2^{k_1} + 2^{k_2} + \cdots + 2^{k_l}$, and D_x is the finite set with canonical index x. Note that $y \in D_x \Rightarrow y < x$.

Furthermore, T is r.e. (*recursively enumerable*) if this is the case for all relations R_a, thus $R_a(n, m)$ if and only if $\exists k R'_a(n, m, k)$ with R'_a recursive.

Note that in a recursive transition system all possible transitions from any state are finite in number and can be computed. We call a process *computable* if it can be represented up to some behavioral equivalence by a recursive transition system, and *semi-computable* if this is the case for some r.e. transition system. In order to express recursive and r.e. transition systems over alphabet B, we define an extended class of register machine iterative programs.

Definition 4.2.2. Let $B_t = B \cup \{t\}$. Let the class of register machine iterative programs with actions in B_t, notation

$$\mathsf{RMI}(B_t, i),$$

be defined as $\mathsf{RMI}(i)$ (see Definition 3.1.1), but with the following two extra clauses:

(7) $a \in \mathsf{RMI}(B_t, i)$ if $a \in B_t$,

(8) if $P, Q \in \mathsf{RMI}(B_t, i)$ and $a, b \in B_t$, then $aP + bQ \in \mathsf{RMI}(B_t, i)$.

Note that $\mathsf{RMI}(B_t, i)$ is closed under successors and under associativity of sequential composition.

We first do not concern ourselves with rootedness, and formulate two basic lemmas. Using these, our main expressivity results follow in a straightforward manner.

LEMMA 4.2.3. *Let T be a recursive transition system with labels in B. Then T modulo branching bisimulation equivalence is expressible in* $\mathsf{ACP}^{\tau *\$}(A_5(B), \gamma)$.

PROOF. Without loss of generality we assume that $|B| > 1$. We use the bijective pairing function $\langle \_, \_ \rangle$ defined by $\langle n, m \rangle = \frac{1}{2}((n + m)^2 + 3m + n)$, with unpairing functions $(\_)_0$ and $(\_)_1$. So $\langle 0, 0 \rangle = 0$, $\langle 1, 0 \rangle = 1$, $n = \langle (n)_0, (n)_1 \rangle$, and $(n)_0 \leq n \geq (n)_1$.

Let $T = \langle S, B, \{R_a \mid a \in B\}, 0 \rangle$ be characterized by the functions h and $next$ (cf. Definition 4.2.1), and let $B_\tau = B \cup \{\tau\}$. We transform T into $\overline{T} = \langle \overline{S}, B_\tau, \{R_a \mid a \in B_\tau\}, 0 \rangle$ by replacing each state with a τ-loop of appropriate size, of which each state has at most one outgoing B-transition:

(1) $\overline{S} = \{\langle s, j \rangle \mid s \in S, j \leq next(s)\}$,

(2) $\forall s \in S \setminus \{1\} \begin{pmatrix} \langle s, j+1 \rangle \xrightarrow{\tau} \langle s, j \rangle & \text{if } j < next(s) \\ \langle s, 0 \rangle \xrightarrow{\tau} \langle s, next(s) \rangle, & \end{pmatrix}$

(3) $\langle s, j \rangle \xrightarrow{a} \langle s', 0 \rangle \in \overline{T}$ if $s \xrightarrow{a} s'$ in T and $\langle h(a), s' \rangle = j$.

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

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

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

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

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

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

For process term P and $n, k, l \in \mathbb{N}$ we write $P : n, 0 \to k, l$ if there is an $m \in \mathbb{N}$ such that $\partial_{H_5}(Px \parallel \mathcal{R}_5(0, n, 0, 0, 0)) = t^m \cdot \partial_{H_5}(x \parallel \mathcal{R}_5(0, k, l, 0, 0))$. According to Corollary 3.2.2 we can choose $P_1, P_2, P_3, P_4 \in \mathsf{RMI}(5)$ such that $\mathsf{ACP}^{*\$}(A_5(\emptyset), \gamma) \vdash$

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

For the case $|B| = 4$ consider the program P schematically depicted in Figure 1, suggesting how the program should be adapted for other values of $|B| > 1$. In this scheme a dotted arrow stands for a t-step, and a bold arrow starting from Q_i $(i = 1, \ldots, 4)$ marks the initial and terminal state associated with process term P_i. We first argue that P (and therefore also each of its states) is expressible in $\mathsf{RMI}(B_t, 5)$ (modulo strong bisimulation). Let S be the process that starts in state Q_1 and terminates at state Q. Then

$$S = P_1 \left(\begin{array}{l} s_2 P_2 \left(\begin{array}{l} s_2 \left(\begin{array}{l} s_2(z_2(tP_3 + g(2)P_4) + s_2(tP_3 + g(3)P_4)) \\ + \\ z_2(tP_3 + g(1)P_4) \end{array} \right) \\ + \\ z_2(tP_3 + g(0)P_4) \end{array} \right) \\ + \\ z_2 P_3 \end{array} \right)$$

We define

$$\begin{aligned}
R &= s_1 a_1 a_1 S((z_1 S)^* s_1), \\
\mathsf{Exit} &= (s_0{}^* c_0)(s_2{}^* c_2)(s_3{}^* c_3)(s_4{}^* c_4),
\end{aligned}$$

and conclude that $P \leftrightarrows R^* c_1 \cdot \mathsf{Exit}$. (Note that $R^* c_1 \in \mathsf{RMI}(B_t, 5)$.)

Now $\bar{T} \leftrightarrows_b \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5)$ where the terminal state 1 plays the role of $\surd$. We shall not define a witnessing bisimulation relation in detail but sketch the idea. Each state $\langle s, j \rangle$ in $\bar{T}$ with $s \neq 1$ is related to each state in the τ-loop containing $\tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5(0, \langle s, 0 \rangle, 0, 0, 0))$. The different states $\langle s, \ldots \rangle$ of this τ-loop are maintained in register R_1 by P_1 and (if executed) P_2, and updated by P_3. Furthermore,

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

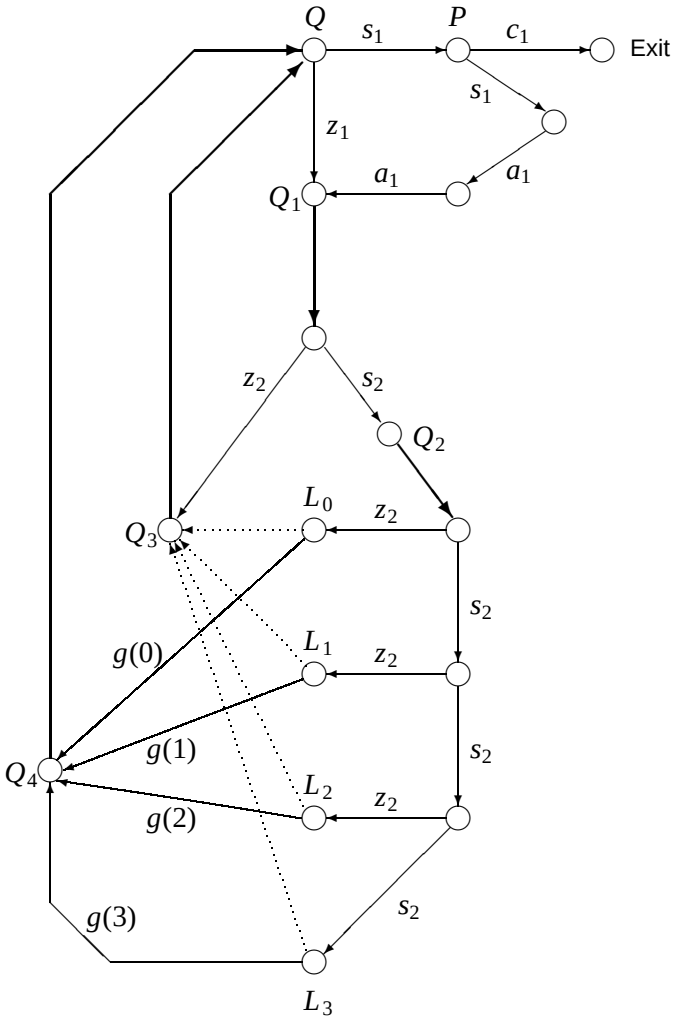


FIG. 1. A register program scheme.

if and only if

$$\tau_{\{t\}} \circ \partial_{H_5}(L_i \parallel \mathcal{R}_5(0, \langle s, j\rangle, 0, 0, 0)) \xrightarrow{g(i)} \tau_{\{t\}} \circ \partial_{H_5}(Q_4 \parallel \mathcal{R}_5(0, \langle s, j\rangle, 0, 0, 0))$$
$$\stackrel{\tau}{\Rightarrow} \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5(0, \langle s', 0\rangle, 0, 0, 0)).$$

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

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

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

Because $\leftrightarrows_b$ is an equivalence relation, we conclude $T \leftrightarrows_b \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5)$, so T can be expressed in $\mathsf{ACP}^{\tau * \$}(A_5(B), \gamma)$. □

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 4.2.4. *Let T be an r.e. transition system with labels in B. Then T modulo τ-bisimulation equivalence is expressible in* $\mathsf{ACP}_\tau^{*\$}(A_5(B), \gamma)$.

PROOF. Assume $|B| > 1$. We use the bijective pairing function $\langle \_, \_ \rangle$ from the previous proof. Let $T = \langle S, B, \{R_a \mid a \in B\}, 0\rangle$ with $R_a(n, m) \Leftrightarrow \exists k R'_a(n, m, k)$ for some decidable R'_a, and with function $\underline{h}$ encoding the actions of B. We transform T into the recursive transition system $\overline{T} = \langle \mathbb{N}, B_\tau, \{\overline{R}_a \mid a \in B_\tau\}, 0\rangle$ by defining the following transitions:

(1) $\forall n \in \mathbb{N}\setminus\{1\}(n \xrightarrow{\tau} \langle (n)_0, (n)_1 + 1\rangle)$,

(2) $\forall a \in B$ and $\forall n, i, m \in \mathbb{N}$, $\overline{R}_a(\langle n, i\rangle, \langle m, 0\rangle))$ if and only if $R'_a(n, m, k)$, $(i)_0 = h(a)$, $((i)_1)_0 = m$, and $(((i)_1)_1)_0 = k$ (thus, $i = \langle h(a), \langle m, \langle k, l\rangle\rangle\rangle$ for some l).

So, each state $n \in \mathbb{N} \setminus \{1\}$ has exactly one outgoing τ-transition $n \xrightarrow{\tau} \langle (n)_0, (n)_1 + 1\rangle$ and at most one outgoing a-step for at most one $a \in B$.

Moreover, $\overline{T} = \langle 0, 0\rangle^\frown \langle 1, 0\rangle \overline{T} \leftrightarrows_\tau T$ by the following τ-bisimulation TB:

$$\begin{array}{l} \mathrm{TB}(0, 0), \\ \mathrm{TB}(\langle k, n\rangle, k) \text{ for all } n \in \mathbb{N} \text{ and } k \in \mathbb{N}\setminus\{1\}, \text{ and} \\ \mathrm{TB}(1, 1). \end{array}$$

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

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

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

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

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

Now $\overline{T} \leftrightarrows_\tau \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5)$ 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 $\tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5(0, \langle s, 0\rangle, 0, 0, 0))$. The different states $\langle s, \ldots\rangle$ of this τ-sequence are maintained in register process R_1 by P_1 and (if executed) P_2, and updated by P_3. Furthermore,

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

if and only if

$$\tau_{\{t\}} \circ \partial_{H_5}(L_i \parallel \mathcal{R}_5(0, \langle s, j\rangle, 0, 0, 0)) \xrightarrow{g(i)} \tau_{\{t\}} \circ \partial_{H_5}(Q_4 \parallel \mathcal{R}_5(0, \langle s, j\rangle, 0, 0, 0))$$
$$\stackrel{\tau}{\Rightarrow} \quad \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5(0, \langle s', 0\rangle, 0, 0, 0)).$$

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

Because $\leftrightarrows_\tau$ is an equivalence relation, we conclude $T \leftrightarrows_\tau \tau_{\{t\}} \circ \partial_{H_5}(Q \parallel \mathcal{R}_5)$. □

With these two lemmas our main expressivity results follow immediately.

THEOREM 4.2.5. *Let T be a recursive transition system with labels in B. Then T modulo rooted branching bisimulation equivalence is expressible in* $\mathsf{ACP}^{\tau *\$}(A_5(B), \gamma)$.

PROOF. Transform T into $\tilde{T}$ 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 $\tilde{T}$ 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 $0^\frown 1T \leftrightarrows 0^\frown 1\tilde{T}$, and

$$0^\frown 1\tilde{T} \leftrightarrows \begin{cases} \left(\sum_{\{a \in B \mid R_a(0,1)\}} a + \sum_{\{a \in B, m \in \mathbb{N} \mid m \neq 1,\ R_a(0,m)\}} a \cdot m^\frown 1\tilde{T} \right) & \text{if } \bigcup_{a \in B,\, j \in \mathbb{N}} R_a(0, j) \neq \emptyset, \\ \delta & \text{otherwise.} \end{cases}$$

Because $\tilde{T}$ has no incoming transitions in its root state, it is sufficient to express $m^\frown 1\tilde{T}$ modulo branching bisimulation for each appropriate value of $m \in \mathbb{N} \setminus \{0\}$. As proved in Lemma 4.2.3, this is possible. □

In exactly the same way our next expressivity result follows from Lemma 4.2.4.

THEOREM 4.2.6. *Let T be an r.e. transition system with labels in B that initially is finitely branching. Then T modulo rooted τ-bisimulation equivalence is expressible in* $\mathsf{ACP}_\tau^{*\$}(A_5(B), \gamma)$.

We note that Theorem 4.2.6 strengthens an expressivity result of Baeten et al. [1987] which states that each recursive transition system over a finite set of labels can be expressed in $\mathsf{ACP}_\tau(A, \gamma)$ with finite, guarded recursive specifications.

4.3. AVOIDING BINARY KLEENE STAR. In the present setting we do not need binary Kleene star: its use can be encoded with the help of auxiliary actions, communication and abstraction. This is the case for our results on $\leftrightarrows_{rb}$ as well as on $\leftrightarrows_{r\tau}$. In order to prove this, let

$$\simeq \in \{ \leftrightarrows_{rb}, \leftrightarrows_{r\tau} \}.$$

We shall only use that $x\tau \simeq x$, and we repeat below two basic results of Bergstra and Ponse [2001]. The first of these states that for each finite state transition system with labels in A (and thus each *regular process*) there is a finite extension A^{fe}

of A (i.e., $A^{fe} \setminus A$ is finite) such that it can be expressed in $\mathsf{ACP}^{\tau\$}(A^{fe}, \gamma)$ or $\mathsf{ACP}^{\$}_{\tau}(A^{fe}, \gamma)$. So, in particular our modeling of *register machine programs* does not depend on the use of $*$.

THEOREM 4.3.1. *For each finite state transition system T with labels in A there is a finite extension A^{fe} of A such that T can be expressed in $\mathsf{ACP}^{\tau\$}(A^{fe}, \gamma)$ or in $\mathsf{ACP}^{\$}_{\tau}(A^{fe}, \gamma)$ with handshaking only, and the actions in A not subject to communication.*

PROOF. Let T be a finite state transition system with labels in A. Then, for some $n \in \mathbb{N}$, T can be characterized by p_1 in the linear system

$$\mathsf{p}_i = \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot \mathsf{p}_j\right) + \beta_i \ (i = 1, \ldots, n)$$

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

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

and let the only communications over A^{fe} be defined by $\gamma(r_j, s_j) = t$ $(j = 0, \ldots, n)$. Consider the following processes:

$$\left(\sum_{j=1}^{n} \alpha_{i,j} \cdot s_j\right) + \beta_i \quad \text{abbreviated by} \quad F_i \text{ for } i = 1, \ldots, n,$$

$$\left(\sum_{j=1}^{n} r_j \cdot F_j\right) {}^{\$} r_0 \qquad \text{abbreviated by} \quad K,$$

$$\left(\sum_{j=1}^{n} r_j \cdot s_j\right) {}^{\$} s_0 \qquad \text{abbreviated by} \quad L.$$

Then $\mathsf{p}_1 \simeq \tau_{\{t\}} \circ \partial_H(F_1 \cdot K \parallel L)$. This can be shown with help of the infinite transition system characterized by

$$\mathsf{q}_i(k) = \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot \mathsf{q}_j(k+1)\right) + \beta_i, \ i = 1, \ldots, n \text{ and } k \in \mathbb{N}.$$

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

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

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

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

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

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

and in particular $\mathsf{p}_1 \simeq \mathsf{q}_1(0) \simeq \tau_{\{t\}} \circ \partial_H(F_i \cdot K \parallel L)$. □

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

THEOREM 4.3.2. *A* $\mathsf{BPA}^{*\$}(A)$ *register process* $(\overline{a}(\overline{a}^{\$}\,\overline{s}) + \overline{z})^*\overline{c}$ *can be defined in* $\mathsf{ACP}^{\tau\$}(A^{fe}, \gamma)$ *or in* $\mathsf{ACP}^{\$}_{\tau}(A^{fe}, \gamma)$ *with handshaking only if* $|A^{fe}\backslash A| \geq 5$.

PROOF. Let $A^{fe}\backslash A = \{t\} \cup H$ with $H = \{r_i, s_i \mid i = 0, 1\}$, and let $\gamma(r_0, s_0) = \gamma(r_1, s_1)$ be the only communications defined. Consider

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

Then it follows in a similar way as shown above that $(\overline{a}(\overline{a}^{\$}\,\overline{s}) + \overline{z})^*\overline{c} \simeq \tau_{\{t\}} \circ \partial_H(P\,Q \parallel R)$. □

5. *Alternatives*

In this section we first show that we can reduce the number of registers used in Section 3.2 for the modeling of register machine computation. Then we consider two alternatives for the push-down, each of which can be used to obtain similar results as proved before. Typically, both these operations can be used to define some form of counting.

5.1. BOUNDED REGISTERS. Recall the crucial Theorem 3.2.1, which involves a particular modeling of register machine computations in $\mathsf{ACP}^{*\$}(A_4, \gamma)$. In its proof, one of the registers is used to keep track of the "current instruction number" during computation. Therefore, this register can be replaced by a process that mimics a register up to a finite depth (namely, the number of instructions of the "current program"). Such processes can be defined in $\mathsf{BPA}^*(\overline{\alpha}_i)$ (where $\overline{\alpha}_i$ contains the specific register actions, see Section 3.1).

LEMMA 5.1.1. *A bounded register over* $\overline{\alpha}_i$ *can be defined in* $\mathsf{BPA}^*(\overline{\alpha}_i)$.

PROOF. First, let a *subcounter* $S_{i,n}$ over alphabet $\overline{\alpha}_i = \{\overline{a}_i, \overline{s}_i, \overline{z}_i, \overline{c}_i\}$ be defined by induction to its *depth* $n \in \mathbb{N}$ in the following way:

$$\begin{aligned} S_{i,0} &= \overline{s}_i, \\ S_{i,k+1} &= (\overline{a}_i \cdot S_{i,k})^*\overline{s}_i. \end{aligned}$$

Then a bounded register $R_{i,n}$ that can hold values $0, 1, \ldots n$ can be defined as follows: $R_{i,0} = \overline{z}_i{}^* \overline{c}_i$ and $R_{i,n+1} = (\overline{a}_i \cdot S_{i,n} + \overline{z}_i)^* \overline{c}_i$. □

As a consequence, Theorem 3.2.1 can be formulated as a "three-$-statement," that is, with the use of only three registers, and all previous results can be obtained with one register less. A perhaps more appealing use of bounded registers comes up in Section 5.3.

5.2. TWO ALTERNATIVES FOR PUSH-DOWN. In addition to push-down, we have defined two other recursive operations that are non-regular, and that can be seen as variations on the binary Kleene star. Bergstra et al. [1994] introduced the non-regular *nesting* operation $\_^\sharp\_$, which is defined by

$$x^\sharp y = x \cdot ((x^\sharp y) \cdot x) + y.$$

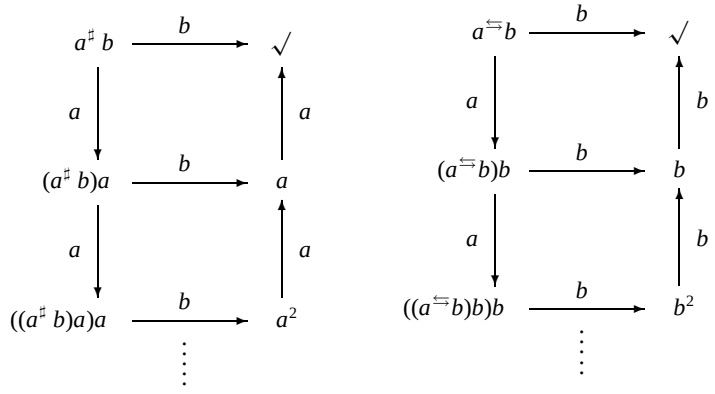
More recently, Bergstra and Ponse [2001] introduced the *back and forth* operation $\_^{\leftrightarrows}\_$, which is defined by

$$x^{\leftrightarrows} y = x \cdot ((x^{\leftrightarrows} y) \cdot y) + y.$$

Transition rules for $\sharp$ and $\leftrightarrows$ are

$$\frac{x \xrightarrow{a} \surd}{\begin{array}{l} x^\sharp y \xrightarrow{a} (x^\sharp y)x \\ y^\sharp x \xrightarrow{a} \surd \\ x^{\leftrightarrows} y \xrightarrow{a} (x^{\leftrightarrows} y)y \\ y^{\leftrightarrows} x \xrightarrow{a} \surd \end{array}} \quad \text{and} \quad \frac{x \xrightarrow{a} x'}{\begin{array}{l} x^\sharp y \xrightarrow{a} x'((x^\sharp y)x) \\ y^\sharp x \xrightarrow{a} x' \\ x^{\leftrightarrows} y \xrightarrow{a} x'((x^{\leftrightarrows} y)y) \\ y^{\leftrightarrows} x \xrightarrow{a} x' \end{array}}$$

As an example, consider for actions a and b the transition systems of $a^\sharp b$ and $a^{\leftrightarrows} b$:



It is easily seen that # and $\leftrightarrows$ are non-regular, and it can be argued that these together with $ are the most simple candidates for obtaining a binary, non-regular recursive operation. Let $\diamond \in \{\sharp, \leftrightarrows\}$. Adding $\diamond$ to the signature of $\mathsf{ACP}(A, \gamma)$, and its defining axiom to those of $\mathsf{ACP}(A, \gamma)$ yields the system which we denote by

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

In the same way, we define $\mathsf{ACP}^{*\diamond}(A, \gamma)$ as the extension of $\mathsf{ACP}^*(A, \gamma)$ with $\diamond$.

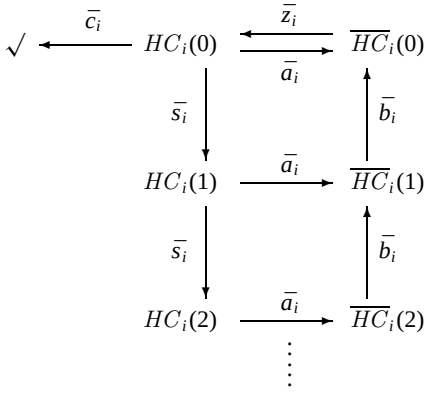
5.3. RELATED RESULTS. All previous results have their counterpart in $\mathsf{ACP}^{*\diamond}(A,\gamma)$. In order to show this we introduce the auxiliary notion of a "$\diamond$-half-counter" (cf. Bergstra et al. [1994]).

$$HC_i = ((\overline{s_i^{\diamond}}\,\overline{a}_i)\cdot \overline{z}_i)^*\overline{c}_i,$$

where we stick to the alphabet $\overline{\alpha}_i$, although most actions lose the intuition previously given.

Let $\overline{b} = \overline{s}$ if $\diamond = \#$, and $\overline{b} = \overline{a}$ if $\diamond = \leftrightarrows$. We use the following, perhaps more convenient characterization of HC_i:

$$\begin{aligned}
HC_i &= HC_i(0)\\
HC_i(0) &= \overline{s}_i HC_i(1) + \overline{a}_i\overline{HC}_i(0) + \overline{c}_i\\
HC_i(n+1) &= \overline{s}_i HC_i(n+2) + \overline{a}_i\overline{HC}_i(n+1)\\
\\
\overline{HC}_i &= \overline{HC}_i(0)\\
\overline{HC}_i(0) &= \overline{z}_i HC_i(0)\\
\overline{HC}_i(n+1) &= \overline{b}_i\overline{HC}_i(n)
\end{aligned}$$



We now provide a result that relates to Theorem 3.2.1. To keep things simple, we will use a bounded register $R_{0,k}$ (as defined in the proof of Lemma 5.1.1) when implementing a register machine program with k instructions. Furthermore, we will use process term $\overline{HC}_i(n)$ to model register i having value n. Finally, we will use an extra $\diamond$-half-counter HC_4 for "shifting," in order to model instructions (a_i, l).

THEOREM 5.3.1. *Let $f:\mathbb{N}\to\mathbb{N}$ be computable (not necessarily total). There exist $P\in\mathsf{BPA}^*(\overline{\alpha}_5)$, $k\in\mathbb{N}$, and computable $g:\mathbb{N}\to\mathbb{N}\backslash\{0\}$ such that if $f(n)$ is defined, then $g(n)$ is defined and*

$$\begin{aligned}
&\mathsf{ACP}^{*\diamond}(A_5,\gamma)\vdash \partial_{H_5}(Px \parallel (R_{0,k}\parallel \overline{HC}_1(n)\parallel \overline{HC}_2\parallel \overline{HC}_3\parallel HC_4))\\
&\quad = t^{g(n)}\cdot \partial_{H_5}(x \parallel (R_{0,k}\parallel \overline{HC}_1(f(n))\parallel \overline{HC}_2\parallel \overline{HC}_3\parallel HC_4)),
\end{aligned}$$

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

$$\mathsf{ACP}^{*\diamond}(A_5,\gamma)\vdash \partial_{H_5}(Px \parallel (R_{0,k}\parallel \overline{HC}_1(n)\parallel \overline{HC}_2\parallel \overline{HC}_3\parallel HC_4)) = t^i\cdot M_i.$$

PROOF. We first consider the #-case. Let $\overline{P}$ be a register machine program that computes f using registers 1, 2 and 3, and numbered instructions $1, 2, \ldots, k$. Our modeling of register machine program expressions is as follows: let

$$P = L_1' Q,\quad Q = (s_0 P_1)^* z_0,$$

and

$$P_m = \begin{cases} L'_m & \text{if } k = 1, \\ s_0 P_{m+1} + z_0 L'_m & \text{if } k > 1 \text{ and } m+1 < k, \\ s_0 L'_{m+1} + z_0 L'_m & \text{if } k > 1 \text{ and } m+1 = k, \end{cases}$$

$$L'_m = \begin{cases} (s_3{}^* z_3 a_3)(s_2{}^* z_2 a_2)(s_0{}^* z_0) & \text{if } \mathit{line}(m, \overline{P}) = \mathsf{halt}, \\ ((s_i s_4)^* z_i s_4 a_4)((s_4 s_i)^* z_4 a_i) \cdot a_0^l & \text{if } \mathit{line}(m, \overline{P}) = (a_i, l), \\ s_i \cdot a_0^l + z_i a_i \cdot a_0^{l'} & \text{if } \mathit{line}(m, \overline{P}) = (s_i, l, l'). \end{cases}$$

We do not provide a precise description of register machine program process terms in this case, but it is apparent that these are deterministic and reside in $\mathsf{BPA}^*(\overline{\alpha}_5)$. Now the pattern

$$\partial_{H_5}(L'_m Q x \parallel (R_{0,k} \parallel \overline{HC}_1(x_1) \parallel \overline{HC}_2(x_2) \parallel \overline{HC}_3(x_3) \parallel HC_4)),$$

represents the register machine that is about to perform instruction m on machine configuration $\langle x_1, x_2, x_3 \rangle$. For example, in the case that L'_m models the instruction (a_2, l), we obtain in $4x_2 + 2l + 8$ t-steps the next (expected) pattern in which L'_m is updated to L'_l and x_2 to $x_2 + 1$. The remaining part of the proof is similar to that of Theorem 3.2.1.

It should be clear how to adapt the above proof to the $\leftrightarrows$-case (occasionally replacing s_i actions by a_i, and $\overline{s_i}$ actions by $\overline{a_i}$). □

In a similar way Corollary 3.2.2 can be adapted to the $\diamond$-cases. Therefore we have the following results:

COROLLARY 5.3.2

(1) In both $\mathsf{ACP}^{*\sharp}(A_5, \gamma)$ and $\mathsf{ACP}^{*\leftrightarrows}(A_5, \gamma)$ provable equality between process terms is not decidable. (Cf. Theorem 3.3.1).

(2) Let T be a recursive transition system with labels in B. Then T modulo rooted branching bisimulation equivalence is expressible in $\mathsf{ACP}^{\tau *\sharp}(A_6(B), \gamma)$ and in $\mathsf{ACP}^{\tau *\leftrightarrows}(A_6(B), \gamma)$. (Cf. Theorem 4.2.5).

(3) Let T be an r.e. transition system with labels in B that initially is finitely branching. Then T modulo rooted τ-bisimulation equivalence is expressible in $\mathsf{ACP}^{*\sharp}_\tau(A_6, (B)\gamma)$ and in $\mathsf{ACP}^{*\leftrightarrows}_\tau(A_6, (B)\gamma)$. (Cf. Theorem 4.2.6).

(4) Results 2 and 3 hold without the $*$ operation if a finite number of auxiliary actions (and handshaking communication over these) is included. Result 1 holds without the $*$ operation if abstraction and a finite number of auxiliary actions (and handshaking communication over these) are included. (Cf. Theorems 4.3.1 and 4.3.2, and Bergstra and Ponse [2001] for some proofs).

6. *Conclusions*

We have shown that $\mathsf{ACP}^{*\$}(A, \gamma)$ (i.e., $\mathsf{ACP}(A, \gamma)$ with binary Kleene star and push-down) allows for a straightforward modeling of register machine computation. Adding abstraction to this setting (either axiomatizing rooted branching bisimilarity or rooted τ-bisimilarity) yields a substantial increase in expressivity: (at least) each computable process can be specified with help of auxiliary actions. Furthermore,

with abstraction the use of the binary Kleene star can be avoided. In our presentation the use of push-down has paved the way to corresponding results using $\sharp$ (the nesting operation) or $\leftrightarrows$ (the back and forth operation) instead. It should be noticed that the systems $\mathsf{ACP}^{*\sharp}(A,\gamma)$ and $\mathsf{ACP}^{*\leftrightarrows}(A,\gamma)$ (and their versions with abstraction) have their own strong points. In particular, a cycle cannot be defined in $\mathsf{ACP}^{*\sharp}(A,\gamma)$ or $\mathsf{ACP}^{*\leftrightarrows}(A,\gamma)$ without use of $*$, so both $\sharp$ and $\leftrightarrows$ are not "iterative" in the most strict sense, and thus may be judged "more primitive."

These results establish various equational foundings of process algebra: each computable process can simply be represented by a term in each of $\mathsf{ACP}^{\tau\$}(A,\gamma)$, $\mathsf{ACP}^{\tau\sharp}(A,\gamma)$ and $\mathsf{ACP}^{\tau\leftrightarrows}(A,\gamma)$ or the associated rooted τ version. Adding binary Kleene star as well yields a more flexible and natural format for the specification of concurrent processes. Straightforward definitions of typical processes such as stacks, bags (multi-sets) and queues with these recursive operations are given in our companion paper [Bergstra and Ponse 2001].

In the modeling of computability provided here, both register machine *programs* and *registers* are captured by process terms, and their sequential interaction is specified in a concurrent fashion. Of course, many alternatives are conceivable. We mention the approach of Bergstra and Loots [1999], where typically a register is viewed as a *coprogram*, i.e., a data type that provides a service to a program. This view emphasizes that programs in RMI can be defined as finite state objects, whereas registers (coprograms) necessarily have unbounded capacity. A process algebraic approach that explicitly incorporates data is μCRL (micro Common Representation Language) defined by Groote and Ponse [1995]), an ACP-based language in which processes can be parameterized with data via data-parametric actions and recursive specifications. Furthermore, μCRL contains conditional composition and data-parametric forms of communication and summation. Ponse [1996] proved that each computable process can be specified in a BPA-oriented fragment of μCRL.

In this paper we showed undecidability of ACP with one of $\$$, $\sharp$, $\leftrightarrows$ and at least one of abstraction or binary Kleene star. We did not further address the issue of proof theory. Some interesting conditional proof rules are the following variants of RSP, the Recursive Specification Principle (cf. e.g., Baeten and Weijland [1990] and Fokkink [2000]):

$$(\mathrm{RSP}^{*})\ \frac{\partial_A(y)=\delta \quad x=yx+z}{x=y^{*}z}, \qquad (\mathrm{RSP}^{\$})\ \frac{\partial_A(y)=\delta \quad x=yxx+z}{x=y^{\$}z},$$

$$(\mathrm{RSP}^{\leftrightarrows})\ \frac{\partial_A(y)=\delta \quad x=yxz+z}{x=y^{\leftrightarrows}z}, \quad (\mathrm{RSP}^{\sharp})\ \frac{\partial_A(y)=\delta \quad x=yxy+z}{x=y^{\sharp}z}.$$

Here the condition $\partial_A(y)=\delta$ (the formulation of which stems from Kamsteeg [1999]) is only relevant for settings with abstraction and rules out processes with an initial τ-step, in order to exclude undesirable identities like $\tau a=\tau^{*}\delta$. It is an open question whether the various ACP extensions when equipped with the appropriate RSP variant(s) *characterize* the associated type of bisimulation equivalence (strong, rooted branching or rooted τ). This is a topic of ongoing research. A positive result in this vein is the completeness of $\mathsf{BPA}(A)+(\mathrm{BKS1})+(\mathrm{RSP}^{*})$ for strong bisimulation equivalence (which follows from the equational axiomatization of Fokkink and Zantema [1994]: (BKS2) and (BKS3) are derivable). Furthermore, Aceto et al. [1998b] proved that a whole range of process semantics coarser than strong

bisimulation do *not* allow a finite equational characterization of the binary Kleene star (see also Aceto et al. [1998a]). Moreover, Sewell [1997] showed that there does not exist a finite equational characterization of the binary Kleene star modulo strong bisimulation in the presence of δ, due to the fact that $(a^k)^*\delta$ is strongly bisimilar to $a^*\delta$ for positive integers k. Fokkink [1997] defined the *perpetual loop*, a restricted form of the binary Kleene star: $x^\omega = x(x^\omega)$ (in a setting with δ and $*$ this yields $x^\omega = x^*\delta$), and provided an RSP-based complete axiomatization of bisimulation equivalence for BPA_δ with perpetual loop. Finally, equational axiomatizations of bisimilarity for other BPA-oriented systems with some form of iteration were given in Fokkink [1994]; Aceto et al. [1996]; Aceto and Ingólfsdóttir [1996]; Fokkink [1996]; Aceto and Fokkink [1997]; van Glabbeek [1997]; Aceto et al. [1998c]; and Aceto and Groote [1999] (for an overview, see Bergstra et al. [2001]).

Appendix A. Universal Register Machines with Two Registers

It is a standard result that the class of recursive functions is characterized by *register machine computability*. Here we recall a particular approach, in which a register machine program is a (finite) set of instructions numbered $1, \ldots, k$ 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 holds value zero, then go to instruction l', otherwise subtract 1 from register i and go to instruction l.

Let the m-tuple $\langle x_1, x_2, \ldots, x_m\rangle$ describe the contents of registers $1, 2, \ldots, m$. 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 $1, \ldots, m$, in such a way that $f(n)$ is defined if and only if P started with the instruction numbered 1 on register machine configuration $\langle n, 0, \ldots, 0\rangle$ computes to a halt instruction. We adopt as output convention that in this case the contents of the register machine is $\langle f(n), 0, \ldots, 0\rangle$, and we represent this situation graphically as:

$$\begin{array}{c}\langle n, 0, \ldots, 0\rangle \\ \Downarrow P \\ \langle f(n), 0, \ldots, 0\rangle.\end{array}$$

If $f(n)$ is not defined, the computation programmed by P on state $\langle n, 0, \ldots, 0\rangle$ 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 (based on Minsky [1967]) for 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

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

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

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

where p_m is the mth 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 $2^{x_1} \cdot 3^{x_2} \cdot \cdots \cdot p_m^{x_m}$ into

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

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 $\langle 2^{x_1} \cdot 3^{x_2} \cdot \cdots \cdot p_m^{x_m}, 0\rangle \Downarrow (A_i, L) \langle p_i \cdot 2^{x_1} \cdot 3^{x_2} \cdot \cdots \cdot p_i^{x_i} \cdot \cdots \cdot p_m^{x_m}, 0\rangle$ with L 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 subtracting 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, for example, $\langle 2, 0\rangle \Downarrow (S_2, L, L') \langle 2, 0\rangle$ with L' the current instruction no. and $\langle 3, 0\rangle \Downarrow (S_2, L, L') \langle 1, 0\rangle$ with L 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 $\bar{P}$ that satisfies

$$\begin{array}{c}\langle n, 0, \ldots, 0\rangle \\ \Downarrow P \\ \langle f(n), 0, \ldots, 0\rangle\end{array} \quad \text{if and only if} \quad \begin{array}{c}\langle 2^n, 0\rangle \\ \Downarrow \bar{P} \\ \langle 2^{f(n)}, 0\rangle.\end{array}$$

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

Finally, $\bar{P}$ 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

$$\begin{array}{c}\langle n, 0, 0\rangle \\ \Downarrow \text{pre} \\ \langle 0, 2^n, 0\rangle \\ \Downarrow \bar{P} \\ \langle 0, 2^{f(n)}, 0\rangle \\ \Downarrow \text{post} \\ \langle f(n), 0, 0\rangle\end{array}$$

where

$$\begin{array}{l}\text{pre} = 1: (a_1, 2) \\ \qquad 2: (s_0, 3, 8) \\ \qquad 3: (s_1, 4, 6) \\ \qquad 4: (a_2, 5) \\ \qquad 5: (a_2, 3) \\ \qquad 6: (s_2, 7, 2) \\ \qquad 7: (a_1, 6) \\ \qquad 8: \text{halt}\end{array} \quad \text{and} \quad \begin{array}{l}\text{post} = 1: (s_1, 2, 4) \\ \qquad 2: (s_1, 3, 7) \\ \qquad 3: (a_2, 1) \\ \qquad 4: (s_2, 5, 6) \\ \qquad 5: (a_1, 4) \\ \qquad 6: (a_0, 1) \\ \qquad 7: \text{halt}\end{array}$$

So pre; $\bar{P}$; post simulates the computation of f on a three register machine, just as is claimed in the proof of Theorem 3.2.1. It is straightforward how concatenation ";" of programs can be defined in this case, e.g., pre; $\bar{P}$ can be obtained by removing instruction 8 in pre and adding 7 to all instruction numbers and references occurring in P.

ACKNOWLEDGMENTS. We thank the referees for useful comments.

REFERENCES

ACETO, L., AND FOKKINK, W. J. 1997. An equational axiomatization for multi-exit iteration. *Inf. Comput. 137*, 2, 121–158.

ACETO, L., FOKKINK, W. J., AND INGÓLFSDÓTTIR, A. 1998a. On a question of A. Salomaa: the equational theory of regular expressions over a singleton alphabet is not finitely based. *Theoret. Comput. Sci. 209*, 1/2, 163–178.

ACETO, L., FOKKINK, W. J., AND INGÓLFSDÓTTIR, A. 1998b. A menagerie of non-finitely based process semantics over BPA*: from ready simulation to completed traces. *Math. Struct. Comput. Sci. 8*, 3, 193–230.

ACETO, L., FOKKINK, W. J., AND INGÓLFSDÓTTIR, A. 1998c. A Cook's tour of equational axiomatizations for prefix iteration. In *Proceedings of the 1st Conference on Foundations of Software Science and Computation Structures* (*FoSSaCS'98*), (Lisbon, Portugal). M. Nivat, Ed. Lecture Notes in Computer Science, vol. 1378. Springer-Verlag, New York, pp. 20–34.

ACETO, L., FOKKINK, W. J., VAN GLABBEEK, R. J., AND INGÓLFSDÓTTIR, A. 1996. Axiomatizing prefix iteration with silent steps. *Inf. Comput. 127*, 1, 26–40.

ACETO, L., AND GROOTE, J. F. 1999. A complete equational axiomatization for MPA with string iteration. *Theoret. Comput. Sci. 211*, 1/2, 339–374.

ACETO, L., AND INGÓLFSDÓTTIR, A. 1996. An equational axiomatization of observation congruence for prefix iteration. In *Proceedings of the 5th Conference on Algebraic Methodology and Software Technology*

(*AMAST'96*) (Munich, Germany). M. Wirsing and M. Nivat, Eds. Lecture Notes in Computer Science, vol. 1101. Springer-Verlag, New York, pp. 195–209.

BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. 1987. On the consistency of Koomen's fair abstraction rule. *Theoret. Comput. Sci. 51*, 1/2, 129–176.

BAETEN, J. C. M., BERGSTRA, J. A., AND KLOP, J. W. 1993. Decidability of bisimulation equivalence for processes generating context-free languages. *J. ACM 40*, 3, 653–682.

BAETEN, J. C. M., AND VERHOEF, C. 1993. A congruence theorem for structured operational semantics with predicates. In *Proceedings CONCUR 93*, (Hildesheim, Germany). E. Best, Ed. Lecture Notes in Computer Science, vol. 715. Springer-Verlag, New York, pp. 477–492.

BAETEN, J. C. M., AND VERHOEF, C. 1995. Concrete process algebra. In *Handbook of Logic in Computer Science*, *Volume IV*, *Syntactical Methods*. S. Abramsky, D. Gabbay, and T. Maibaum, Eds., Oxford University Press, Oxford, England, pp. 149–268.

BAETEN, J. C. M., AND WEIJLAND, W. P. 1990. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press.

BERGSTRA, J. A., BETHKE, I., AND PONSE, A. 1993. Process algebra with iteration. Report P9314 (June), Programming Research Group, University of Amsterdam. (See `http://www.science.uva.nl/research/prog/reports/reports.html`).

BERGSTRA, J. A., BETHKE, I., AND PONSE, A. 1994. Process algebra with iteration and nesting. *Comput. J. 37*, 4, 243–258.

BERGSTRA, J. A., FOKKINK, W. J., AND PONSE, A. 2001. Process algebra with recursive operations. In *Handbook of Process Algebra*. J. A. Bergstra, A. Ponse, and S. A. Smolka, Eds., Elsevier, North Holland, Amsterdam, The Netherlands, pp. 333–389.

BERGSTRA, J. A., AND KLOP, J. W. 1984. Process algebra for synchronous communication. *Inform. Cont. 60*, 1/3, 109–137.

BERGSTRA, J. A., AND KLOP, J. W. 1985. Algebra of communicating processes with abstraction. *Theoret. Comput. Sci. 37*, 1, 77–121.

BERGSTRA, J. A., AND LOOTS, M. E. 1999. Programs, interfaces and components, report 006 (July). Artificial Intelligence Preprint Series, CKI, Department of Philosophy, Utrecht University. (See `http://www.phil.uu.nl/ckipreprints.html`).

BERGSTRA, J. A., AND PONSE, A. 2001. Non-requlars iterators in process algebra. *Theoret. Comput. Sci. 269*, 1–2, 203–229. (This is an extended version of two recursive generalizations of iteration process algebra. Report P9808 (June 1998), Programming Research Group, University of Amsterdam,) (See `http://www.science.uva.nl/research/prog/reports/reports.html`).

BERGSTRA, J. A., AND TUCKER, J. V. 1984. Top down design and the algebra of communicating processes. *Sci. Comput. Prog. 5*, 2, 171–199.

BOSELIE, J. 1995. Expressiveness results for process algebra with iteration. Master's dissertation Computer Science, University of Amsterdam, Amsterdam, The Netherlands.

FOKKINK, W. J. 1994. A complete equational axiomatization for prefix iteration. *Inf. Proc. Lett. 52*, 333–337.

FOKKINK, W. J. 1996. A complete axiomatization for prefix iteration in branching bisimulation. *Fund. Inf.* 26, 2, 103–113.

FOKKINK, W. J. 1997. Axiomatizations for the perpetual loop in process algebra. In *Proceedings of the 24th Annual ICALP*. P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds. Springer-Verlag, New York, pp. 571–581.

FOKKINK, W. J. 2000. *Introduction to Process Algebra*. Springer-Verlag, New York.

FOKKINK, W. J., AND ZANTEMA, H. 1994. Basic process algebra with iteration: Completeness of its equational axioms. *Comput. J. 37*, 4, 259–267.

GROOTE, J. F., AND PONSE, A. 1995. The syntax and semantics of μCRL. In *Algebra of Communicating Processes* (Utrecht 1994), *Workshops in Computing*. A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, Eds., Springer-Verlag, New York, pp. 26–62.

GROOTE, J. F., AND VAANDRAGER, F. W. 1992. Structured operational semantics and bisimulation as a congruence. *Inf. Comput. 100*, 2, 202–260.

KAMSTEEG, G. 1999. Formalization of Process Algebra with Data in the Calculus of Constructions with Inductive Types. Ph.D. dissertation, Computer Science, Leiden University, Leiden.

KLEENE, S. C. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, Princeton University Press, Princeton, N.J., pp. 3–41.

MILNER, R. 1980. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, vol. 92. Springer-Verlag, New York.

MINSKY, M. L. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N.J.

PARK, D. M. R. 1981. Concurrency and automata on infinite sequences. In *Proceedings of the* 5*th GI Conference*. P. Deussen, Ed. Lecture Notes in Computer Science, vol. 104. Springer-Verlag, New York, pp. 167–183.

PONSE, A. 1996. Computable processes and bisimulation equivalence. *Form. Asp. Comput. 8*, 6, 648–678.

SEWELL, P. M. 1997. Nonaxiomatisability of equivalences over finite state processes. *Ann. Pure Appl. Logic 90*, 1/3, 163–191.

VAN GLABBEEK, R. J. 1993. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proceedings of the 18th Symposium on Mathematical Foundations of Computer Science* (*MFCS'93*) (Gdansk, Poland). A. Borzyszrowski and S. Sokolowski, Eds. Lecture Notes in Computer Science, vol. 711. Springer-Verlag, New York, pp. 473–484.

VAN GLABBEEK, R. J. 1997. Axiomatizing flat iteration. In *Proceedings of the 8th Conference on Concurrency Theory* (*CONCUR'98*) (Warsaw, Poland). A. Mazurkiewicz and J. Winkowski, Eds. Lecture Notes in Computer Science, vol. 1243, Springer-Verlag, New York, pp. 228–242.

VAN GLABBEEK, R. J., AND WEIJLAND, W. 1989. Branching time and abstraction in bisimulation semantics (extended abstract). In G. RITTER Ed., *Information Processing 89*, Proceedings of the IFIP 11th World Computer Congress, San Fransisco 1989, pp. 613–618. North-Holland. Full version in *J. ACM 43*, 3, 1996, 555–600.

VAN GLABBEEK, R. J., AND WEIJLAND, W. 1996. Branching time and abstraction in bisimulation semantics. *J. ACM 43*, 3, 555–600.

RECEIVED DECEMBER 1999; REVISED SEPTEMBER 2001; ACCEPTED SEPTEMBER 2001