ELSEVIER

Theoretical Computer Science 269 (2001) 203–229

Theoretical Computer Science

www.elsevier.com/locate/tcs

Non-regular iterators in process algebra

Jan A. Bergstra[a,b], Alban Ponse[a,c,*]

[a] *University of Amsterdam, Programming Research Group, Kruislaan 403, 1098 SJ Amsterdam, Netherlands*

[b] *Utrecht University, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, Netherlands*

[c] *CWI, Department of Software Engineering, Kruislaan 413, 1098 SJ Amsterdam, Netherlands*

Received December 1995; revised September 2000; accepted October 2000
Communicated by U. Montanari

Abstract

We consider three forms of non-regular iteration in process algebra: the *push-down* operation \$, defined by $x^{\$}y = x((x^{\$}y)(x^{\$}y)) + y$, the *nesting* operation $\sharp$, defined by $x^{\sharp}y = x((x^{\sharp}y)x) + y$, and the *back and forth* operation $\leftrightarrows$, defined by $x^{\leftrightarrows}y = x((x^{\leftrightarrows}y)y) + y$. In the process algebraic framework ACP with abstraction and one of \$, $\sharp$ or $\leftrightarrows$ we provide definitions of the following standard processes: *stack*, *context-free process*, *bag*, and *queue*. These definitions apply to all standard behavioural equivalences (we only use $x\tau = x$, where τ is the silent step). Moreover, these results yield the expressive power to express computable processes modulo rooted branching bisimulation equivalence, and hence support the equational founding of process algebra: standard processes can be represented as terms. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Concurrency; Process algebra; Iteration; Kleene star; Nesting operation; Push-down operation; Back and forth operation; Expressiveness

1. Introduction

A 'classic' quotation of Milner [35] states that for a proper understanding of the basic issues concerning the behavior of concurrent systems it could be helpful to look for a simple language with "*as few operators or combinators as possible*, *each of which embodies some distinct and intuitive idea*, *and which together give completely general expressive power*".

* Correspondence address: University of Amsterdam, Programming Research Group, Kruislaan 403, 1098 SJ Amsterdam, Netherlands. Tel.: 31-20-525-7592; fax.: 31-20-525-7490.

E-mail address: alban@wins.uva.nl (A. Ponse).

0304-3975/01/$ - see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S0304-3975(00)00413-8

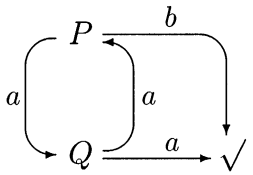
The addition of a single recursive operation to the process algebraic framework ACP (algebra of communicating processes) with abstraction [18, 11] provides 'general expressive power': each computable process can be expressed up to rooted branching bisimulation equivalence [29] and this cannot be done without using 'abstraction'. We consider three candidates for such a recursive operation. The 'distinct and intuitive idea' embodied by each of these is a simple way of counting (by repeated or nested recursion). Rather than focusing on a particular machine model for computability [1] we show how to specify standard processes as one finds in a text book on process theory, such as the stack [32, 37, 11]. Our motivation for this approach is that the 'general expressive power' of the proposed framework then follows from a standard result in [8], where it is shown how a Turing machine can be implemented with two stacks and a regular control process.

In [13] we introduced the *binary Kleene star*, also called *iteration*, in process algebra. The binary Kleene star stems from Kleene [33] and is – in our notation – defined by

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

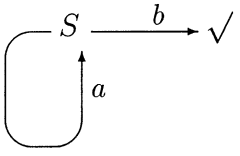
In ACP, the operation $\cdot$ models sequential composition, and $+$ models choice. As common in algebra, the symbol $\cdot$ is often omitted and $+$ binds weakest. So $x^* y$ expresses "either do x and repeat $x^* y$, or do y". The combination of Kleene's $*$-operation and the operations from ACP with abstraction (shortly recalled in the next section) gives the expressive power to describe *regular processes* up to rooted branching bisimilarity: given a set A of actions, a process is *regular over* A if it can be characterized by a finite state system in which labeled transitions model the execution of actions from A. Typically, a regular process can be specified by a *linear specification* (roughly: a right-linear regular grammar). Some examples with actions a and b are

$P = aQ + b$ or in a picture:
$Q = aP + a$



and

$S = aS + b$ or in a picture:



In these pictures, $\surd$ is a symbol expressing (successful) termination. Of course, another specification of S is $a^* b$.

[1] In our setting, register machine programming would be a natural candidate. In Section 10 (Conclusions) we return to this issue.

The focus on iteration in process algebra raised interest in variations of the Kleene star operation (cf. [12, 13, 21, 2, 1, 23]). Here we introduce the binary *push-down* operation $\_^{\$}\_$ as a form of non-regular iteration. This operation is defined by

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

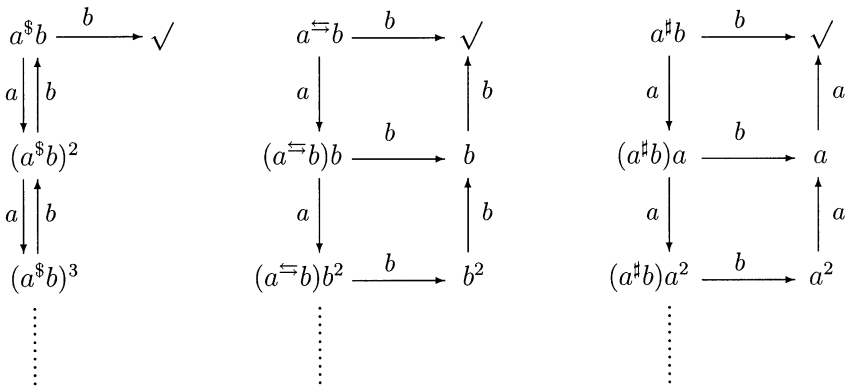
(Either do x and repeat $(x^{\$}y)(x^{\$}y)$, or do y.) A second type of non-regular iteration that we introduce here is the *back and forth* operation $\_^{\leftrightarrows}\_$, defined by

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

(Either do x and repeat $(x^{\leftrightarrows}y)y$, or do y.) Finally, in [13] we introduced the binary *nesting* operation $\_^{\sharp}\_$, defined by

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

(Either do x and repeat $(x^{\sharp}y)x$, or do y.) The operations \$, $\leftrightarrows$ and $\sharp$ are 'non-regular': with each of these a non-regular process can be defined. As an example consider processes $a^{\$}b$, $a^{\leftrightarrows}b$ and $a^{\sharp}b$ with actions a, b illustrated below:



Here the states that are process terms determine possible further behaviour, i.e., outgoing transitions. None of $a^{\$}b$, $a^{\leftrightarrows}b$ and $a^{\sharp}b$ is regular in any common semantics for process algebra.

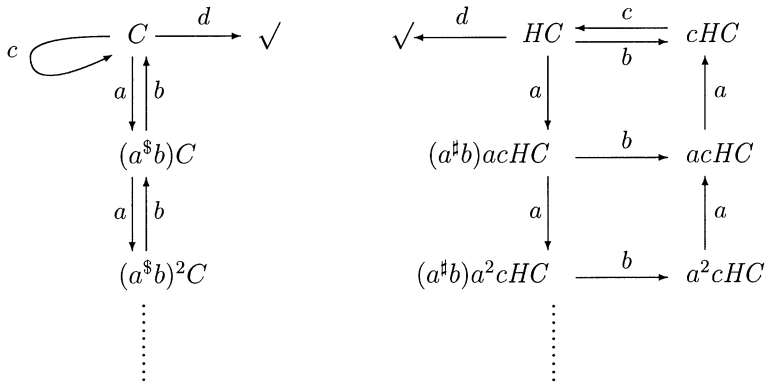
In this paper we show by direct, algebraic proofs that the following standard processes can be defined with one of \$, $\leftrightarrows$ or $\sharp$, the operations from ACP with abstraction, auxiliary actions, and two-party communication (handshaking) between the auxiliary actions:

- A *stack* over a finite data type,
- A *context-free* process (roughly: a process specifiable by a context-free grammar in Greibach normal form),
- A *bag* or *multiset* over a finite data type,
- A *queue* over a finite data type.

These results concern a whole range of process equivalences that respect the axiom $x\tau = x$. Among these are rooted branching bisimulation [29], rooted delay bisimulation

[34], rooted η-bisimulation [37], and rooted weak bisimulation [10]. We give detailed proofs for the settings with \$ and $\sharp$, after which we argue that our results also hold for the case with $\leftrightarrows$.

Two basic, auxiliary processes used in our proofs are the *counter* C and the *half-counter* HC (which stems from [13]), both displayed below:



Using Kleene star and push-down, the counter C can be defined by

$$C = (a(a^{\$}b) + c)^* d$$

with actions a (add one), b (subtract one), c (test zero), and d (terminate). This process can be recognized as a register, i.e., a memory location for a natural number with unbounded capacity and restricted access as modeled by the specific actions. Of course we shall give a definition of C without $*$.

Using iteration and nesting, the half-counter HC can be defined by

$$HC = ((a^{\sharp}b)c)^* d.$$

Initially HC is in 'add-mode', from which it can terminate by executing d, or in which it can stay by performing a-actions (steps). Furthermore, HC can evolve from add-mode into 'subtract-mode' by performing b. In subtract-mode, the half-counter can only 'count back to zero' by performing a-actions after which it can re-enter its initial state by performing a zero-test action c. We provide a definition of HC using $\sharp$ as the only recursive operation.

The structure of the paper is as follows: first (in Section 2) we recall some process algebra and introduce some notation. Then, in Section 3, we discuss regularity and the particular proof rule RSP. In Section 4 we show how to define an arbitrary regular process and a (half-)counter in ACP with abstraction and push-down (nesting) only. With these we specify in Section 5 a stack, which is a particular context-free process that generalizes a counter. In Section 6 we show how an arbitrary context-free process can be defined with help of a regular control process and a stack. In Section 7 we provide specifications of a bag (multiset) and a queue,

processes that both are not context-free. In Section 8 we consider the operation $\leftrightarrows$ and argue that the above-mentioned expressiveness results have their counterparts in this setting. In Section 9 we discuss some general expressiveness issues concerning the various recursive operations. We finish the paper with some conclusions in Section 10.

2. Process algebra, axioms and notation

First we shortly recall some process algebra. The process algebraic framework ACP (algebra of communicating processes, see, e.g., [17, 11]) has two parameters: a finite set A of constants modeling atomic actions, and a (partial) binary, commutative and associative *communication function* γ on A, defining which actions communicate. In order to highlight these parameters we henceforth write $\mathsf{ACP}(A,\gamma)$. Furthermore there is a constant $\delta \notin A$ (deadlock or inaction), and we write A_δ for $A \cup \{\delta\}$. Process operations of $\mathsf{ACP}(A,\gamma)$ are alternative composition or choice ($+$), sequential composition ($\cdot$), parallel composition or merge ($\|$), left and communication merge ($\lfloor\!\lfloor$ and $|$, used for the axiomatization of $\|$), and encapsulation (∂_H, renaming actions in $H \subseteq A$ into δ). We mostly suppress the $\cdot$ in terms, and brackets according to the following precedences: $\cdot > \{\|, \lfloor\!\lfloor, |\} > +$.

$\mathsf{ACP}(A,\gamma)$ is further extended to $\mathsf{ACP}^\tau(A,\gamma)$ by adding the constant $\tau \notin A_\delta$ (the silent step) and hiding τ_I (or 'abstraction', i.e., renaming actions in $I \subseteq A$ into τ). We write $A_{\delta\tau}$ for $A_\delta \cup \{\tau\}$. The axioms of $\mathsf{ACP}^\tau(A,\gamma)$ are displayed in Table 1. These axioms characterize *rooted branching bisimulation* equivalence (see [29]) for the closed terms, further called *process terms*, over $\mathsf{ACP}^\tau(A,\gamma)$. The axioms of $\mathsf{ACP}(A,\gamma)$ are obtained by omitting τ (so in Table 1, a and b then range over A_δ) and τ_I, and the axioms (B1), (B2), and (TI1)–(TI4). The axioms of $\mathsf{ACP}(A,\gamma)$ characterize (*strong*) *bisimulation* equivalence. Note that $+$ and $\cdot$ are associative, and that $+$ is also commutative and idempotent. In this paper we only use two-party communication or *handshaking*, which can be axiomatized by $x \mid y \mid z = \delta$ (see [19]). For a detailed introduction to $\mathsf{ACP}(A,\gamma)$ and $\mathsf{ACP}^\tau(A,\gamma)$ we refer to [11, 24].

Axioms for the $*$-operation are included in Table 1. In [25], Fokkink and Zantema prove that strong bisimilarity for process terms built from A and the operations $+, \cdot$ and $*$ is axiomatized by $\mathsf{BPA}^*(A)$, i.e., the axioms (A1)–(A5) and (BKS1)–(BKS3). We write $\mathsf{ACP}^*(A,\gamma)$ ($\mathsf{ACP}^{\tau*}(A,\gamma)$) for the extension with $*$ and (BKS1)–(BKS4) (all BKS axioms, respectively).

Let for $\diamond \in \{\$, \sharp, \leftrightarrows\}$, $\mathsf{ACP}^{\tau\diamond}(A,\gamma)$ be the extension of $\mathsf{ACP}^\tau(A,\gamma)$ with the concerning recursive operation and its defining axiom. The results in the remainder of the paper involve supersets of A, the set of atomic actions. We require that any such superset A^{ext} satisfies

$$(A^{\mathrm{ext}} \backslash A) \supseteq \{t\} \cup \{r_i, s_i \mid i \in I,\ I \subseteq \mathbb{N} \text{ some index set}\}.$$

Table 1
Axioms of $\mathsf{ACP}^{\tau}(A,\gamma)$ ($a,b\in A_{\delta\tau}$ and $H,I\subseteq A$). BKS axioms for binary Kleene star ($H,I\subseteq A$)

(A1)	$x+y = y+x$	(CM6)	$a\mid bx = (a\mid b)x$
(A2)	$x+(y+z) = (x+y)+z$	(CM7)	$ax\mid by = (a\mid b)(x\parallel y)$
(A3)	$x+x = x$	(CM8)	$(x+y)\mid z = x\mid z+y\mid z$
(A4)	$(x+y)z = xz+yz$	(CM9)	$x\mid(y+z) = x\mid y+x\mid z$
(A5)	$(xy)z = x(yz)$		
		(D1)	$\partial_H(a) = a$ if $a\notin H$
(A6)	$x+\delta = x$	(D2)	$\partial_H(a) = \delta$ if $a\in H$
(A7)	$\delta x = \delta$	(D3)	$\partial_H(x+y) = \partial_H(x)+\partial_H(y)$
		(D4)	$\partial_H(xy) = \partial_H(x)\cdot\partial_H(y)$
(CF1)	$a\mid b=\gamma(a,b)$ if $\gamma(a,b)\downarrow$		
(CF2)	$a\mid b=\delta$ otherwise	(B1)	$x\tau = x$
		(B2)	$x(\tau(y+z)+y)=x(y+z)$
(CM1)	$x\parallel y=(x\between y+y\between x)+x\mid y$		
(CM2)	$a\between x = ax$	(TI1)	$\tau_I(a) = a$ if $a\notin I$
(CM3)	$ax\between y = a(x\parallel y)$	(TI2)	$\tau_I(a) = \tau$ if $a\in I$
(CM4)	$(x+y)\between z = x\between z+y\between z$	(TI3)	$\tau_I(x+y) = \tau_I(x)+\tau_I(y)$
(CM5)	$ax\mid b = (a\mid b)x$	(TI4)	$\tau_I(xy) = \tau_I(x)\cdot\tau_I(y)$

(BKS1)	$x^*y = x(x^*y)+y$	(BKS4)	$\partial_H(x^*y) = \partial_H(x)^*\partial_H(y)$
(BKS2)	$x^*(yz) = (x^*y)z$	(BKS5)	$\tau_I(x^*y) = \tau_I(x)^*\tau_I(y)$
(BKS3)	$(x+y)^*z=x^*(y((x+y)^*z)+z)$		

An extension A^{ext} is called *finite* if $A^{\mathrm{ext}}\backslash A$ is. On $A^{\mathrm{ext}}\backslash A$ we employ the following version of handshaking:

$$\gamma(r_i,s_i)=t$$

for each pair of actions r_i, s_i (thus, $r_i\mid s_i=t$). Furthermore, we often use Σ-notation for finite sums: the expression

$$\sum_{j=1}^{n}P_j$$

abbreviates $P_1+P_2+\cdots+P_n$. If $n=0$, this expression denotes δ. We adopt the convention that $+$ binds weaker, and all other process operations bind stronger than $\sum$. Finally, as in the previous pictures, we use exponentiation: $x^1=x$ and $x^{n+2}=x(x^{n+1})$ ($n\in\mathbb{N}$).

3. Regularity, linear specifications, and RSP

In this section we first give a precise characterization of regularity based on *linear specifications*. Then we discuss data-parametric linear specifications and the proof rule RSP, the recursive specification principle [11, 24]. This principle plays a central role in the proofs of our expressiveness results.

Given a model $\mathscr{M}$ that satisfies the axioms of $\mathsf{ACP}^{\tau\diamond}(A,\gamma)$ for $\diamond \in \{*,\$,\sharp,\leftrightarrows\}$, *process* $\mathsf{p}_1 \in \mathscr{M}$ is *regular over* A_δ *in* $\mathscr{M}$ if for some $n \geqslant 1$ there exist processes $\mathsf{p}_1,\mathsf{p}_2,\ldots,\mathsf{p}_n \in \mathscr{M}$ such that

$$\mathscr{M} \models \mathsf{p}_i = \sum_{j=1}^{n} \alpha_{i,j}\mathsf{p}_j + \beta_i$$

for $i=1,\ldots,n$, where the $\alpha_{i,j}$ and β_i are finite sums of actions or δ. In this case, the processes p_i are said to be *solutions* for the variables X_i in the *linear specification*

$$X_i = \sum_{j=1}^{n} \alpha_{i,j}X_j + \beta_i. \tag{1}$$

So, using linear specifications we can characterize regularity in a model independent way and speak about a 'regular process' without making explicit which model we have in mind. Observe that each process term over $\mathsf{ACP}^*(A,\gamma)$ represents a regular process over A_δ. E.g., a^*b solves $X_1 = aX_1 + b$, and the same holds for $a(a^*b)+b$. This is not the case for $\mathsf{ACP}^{\tau*}(A,\gamma)$, e.g., τ^*b is not regular over A_δ (however, it is the abstraction of a regular process over A_δ: $\tau^*b = \tau_{\{a\}}(a^*b)$ follows from (TI1), (TI2), and (BKS5)). To enhance readability, we often omit summands δ and δX_j in linear specifications (cf. axioms (A6) and (A7)). E.g., ab is regular because it solves X_1 in the linear specification $X_1 = aX_2,\ X_2 = b$ (abbreviating $X_1 = \delta X_1 + aX_2 + \delta,\ X_2 = \delta X_1 + \delta X_2 + b$).

We shall also use *data-parametric* linear specifications, i.e., linear specifications of which the equations are parametric in (some encoding of) $\mathbb{N}$. More precisely, a data-parametric linear specification (over A_δ) consists of a finite number $n \geqslant 1$ of equations of the form $(i=1,\ldots,n)$

$$X_i(k) = \sum_{j=1}^{n} \alpha_{i,j}(k) \cdot X_j(f_{i,j}(k)) + \beta_i(k), \tag{2}$$

where k ranges over (some encoding of) $\mathbb{N}$, $\alpha_{i,j}(k)$ and $\beta_i(k)$ are finite sums of actions or δ for each k, and all $f_{i,j} : \mathbb{N} \to \mathbb{N}$ are primitive recursive. For each $m \in \mathbb{N}$, $X_i(m)$ is considered to define a process that can evolve into $X_j(f_{i,j}(m))$ if $\alpha_{i,j}(m) \neq \delta$ by performing an action of $\alpha_{i,j}(m)$, or that can terminate if $\beta_i(m) \neq \delta$ by performing an action of $\beta_i(m)$. A simple example of a data-parametric linear specification, omitting δ-summands, with k ranging over $\mathbb{N}$ is

$$X_1(0) = aX_1(1) + b,$$

$$X_1(k+1) = aX_1(k+2) + bX_2(k),$$

$$X_2(0) = a,$$

$$X_2(k+1) = aX_2(k).$$

Observe that $a^\sharp b$, $(a^\sharp b)a^{m+1}$ and a^{m+1} solve $X_1(0)$, $X_1(m+1)$ and $X_2(m)$, respectively. If we replace the action b by δ in the above specification, and omit the resulting δ-summands we end up with $X_1'(k) = aX_1'(k+1)$ and the X_2-equations remain the same.

In this case, $a^\sharp\delta$ and $(a^\sharp\delta)a^{m+1}$ solve $X_1'(0)$ and $X_1'(m+1)$, respectively. Notice that in this case $X_1'(k)$ and $X_2(k)$ are independent, and that $a^*\delta$ also solves $X_1'(m)$ for each $m\in\mathbb{N}$.

Remark 1. Data-parametric linear specifications can be seen as a (preferred) notation for linear specifications consisting of a denumerably infinity of equations. For example, a conventional notation for $X(k)=aX(k+1)$ (with k ranging over $\mathbb{N}$) is $X_k=aX_{k+1}$, emphasizing that each X_k is a variable (cf., e.g., [11]).

Linear specifications of the form (1) and (2) are called *guarded* (each occurrence of a variable in the right-hand side is guarded by δ or a sum of atomic actions). The conditional rule RSP (the *recursive specification principle*, see, e.g., [11, 24]) states that each guarded recursive specification has at most one solution per variable (and parameter value). In common process algebra semantics, RSP is a sound proof rule. In this paper we shall use RSP to equate different process terms containing recursive operations, or to establish definitions of (certain) standard processes. As an example, RSP implies that

$$a^*\delta=(aa)^*\delta=a((aa)^*\delta).$$

This can be seen as follows: consider the linear specification

$$X_1=aX_2,$$

$$X_2=aX_1.$$

Then both $\langle a^*\delta,a^*\delta\rangle$ and $\langle(aa)^*\delta,a((aa)^*\delta)\rangle$ solve $\langle X_1,X_2\rangle$. (For the latter pair of solutions, derive $(aa)^*\delta=a(a((aa)^*\delta))$.) By RSP the two identities $a^*\delta=(aa)^*\delta$ and $a^*\delta=a((aa)^*\delta)$ follow. For another example, using the data-parametric linear specification $X_1'(k)=aX_1'(k+1)$ (see above) it follows with RSP that $a^*\delta=a^\sharp\delta=(a^\sharp\delta)a^{m+1}$ for all $m\in\mathbb{N}$. Because $a^*\delta$ is regular, $a^\sharp\delta$ and $(a^\sharp\delta)a^{m+1}$ are regular as well. Without proof we state:

Proposition 2. *Let $\diamond\in\{*,\$,\sharp,\leftrightarrows\}$. Each process term over $\mathsf{ACP}^\diamond(A,\gamma)$ is definable as a solution in a (data-parametric) linear specification.*

A useful consequence of RSP is the *commutativity* and *associativity* of $\|$ and $|$ for processes definable by (data-parametric) linear recursive specifications (see [20]), so in particular for process terms over $\mathsf{ACP}^\diamond(A,\gamma)$ for $\diamond\in\{*,\$,\sharp,\leftrightarrows\}$. (These properties are not derivable from the axioms of $\mathsf{ACP}^\diamond(A,\gamma)$.) Therefore it is permitted that we use commutativity and associativity of $\|$ and $|$ in our proofs whenever convenient. In particular, we need not use brackets in process terms containing repeated applications of $\|$ (cf., e.g., the proof of Theorem 6).

For ease of specification we sometimes consider finite (recursive) specifications of the form (1) in which the $\alpha_{i,j}$ and β_i may be either δ or process terms built from

A with $+, \cdot$ and $*$. Such a specification is considered guarded and still characterizes regularity: with the axioms provided it can always be unfolded into a linear one that defines the same solution for X_1 (the first equation). E.g., X_1 has the same solution in $X_1 = abX_1$ as in the linear specification $X_1 = aX_2,\ X_2 = b$, or for a less trivial example, X_1 has the same solution in

$$X_1 = (a^*b)X_1 + c$$

as in $X_1 = bX_1 + aX_2 + c,\ X_2 = aX_2 + bX_1$ (cf. [16, Theorem 2.1.1]).

4. Expressing regular processes and (half)-counters

In this section we prove that for each regular process P over A_δ there is a *finite* extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}}, \gamma)$ with $\diamond \in \{\$, \sharp\}$. Also, we provide a definition of a counter in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$ and of a half-counter in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}}, \gamma)$ for suitable, finite extensions A^{ext}.

Theorem 3. *For each regular process P over A_δ there exists a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, and the actions in A not subject to communication.*

Proof. Let P_1 be a regular process over A_δ given by $P_i = \sum_{j=1}^{n} \alpha_{i,j}P_j + \beta_i$ $(i = 1, \ldots, n)$ where $\alpha_{i,j}$ and β_i are finite sums of actions or δ. Define A^{ext} as the extension of A with the following $2n + 3$ actions:

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

and consider the following process terms over $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$:

$$\sum_{j=1}^{n} \alpha_{i,j}s_j + \beta_i \text{ abbreviated by } F_i \text{ for } i = 1, \ldots, n,$$

$$\left(\sum_{j=1}^{n} r_jF_j\right)^{\$} r_0 \text{ abbreviated by } K,$$

$$\left(\sum_{j=1}^{n} r_js_j\right)^{\$} s_0 \text{ abbreviated by } L.$$

Then $P_1 = \tau_{\{t\}} \circ \partial_H(F_1K \parallel L)$. This can be shown with help of RSP and the data-parametric linear specification

$$Y_i(k) = \sum_{j=1}^{n} \alpha_{i,j}Y_j(k+1) + \beta_i, \quad i = 1, \ldots, n,\ n \geqslant 1 \quad \text{and } k \in \mathbb{N}.$$

Obviously, P_i is a solution for each $Y_i(k)$ $(i=1,\ldots,n,\ k\in\mathbb{N})$. So it suffices to show that $\tau_{\{t\}}\circ\partial_H(F_iK \parallel L)$ is also a solution for $Y_i(0)$. We show this by first omitting the $\tau_{\{t\}}$-application:

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

Hence, applying $\tau_{\{t\}}$ and axiom $x\tau=x$ (B1), we find for each k

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

So $\tau_{\{t\}}\circ\partial_H(F_i\cdot K^{k+1} \parallel L^{k+1})$ satisfies the equation for $Y_i(k)$. □

Theorem 4. *For each regular process P over A_δ there exists a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$ with handshaking only, and the actions in A not subject to communication.*

Proof. Let the regular process P_1 over A_δ be given by $P_i=\sum_{j=1}^{n}\alpha_{i,j}P_j+\beta_i$ $(i=1,\ldots,n)$ where $\alpha_{i,j}$ and β_i are finite sums of actions or δ. Define A^{ext} as the extension of A with the following $2n+5$ actions:

$$\{t\}\cup H,\ \text{with } H=\{r_j,s_j \mid j=0,\ldots,n+1\},$$

where the s_{n+1},r_{n+1} communications are used to remove remnants of $\sharp$-recursion. Consider the following process terms over $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$:

$$\sum_{j=1}^{n}\alpha_{i,j}s_j+\beta_is_0 \text{ abbreviated by } G_i \text{ for } i=1,\ldots,n,$$

$$\left(\sum_{j=1}^{n}r_jG_j+s_{n+1}r_{n+1}\right)^{\sharp}r_0 \text{ abbreviated by } M,$$

$$\left(\sum_{j=1}^{n+1}r_js_j\right)^{\sharp}(r_0s_0) \text{ abbreviated by } N.$$

Then $P_i=\tau_{\{t\}}\circ\partial_H(G_iM \parallel N)$ for $i=1,\ldots,n$ follows with RSP in a similar way as above. □

A standard recursive specification of a *counter* defines this process as the solution for $X(0)$ in the data-parametric linear specification (k ranging over $\mathbb{N}$):

$$X(0) = a \cdot X(1) + c \cdot X(0) + d,$$

$$X(k+1) = a \cdot X(k+2) + b \cdot X(k).$$

Abbreviate $(a(a^{\$}b)+c)^*d$ to C. It easily follows that the process terms C and $(a^{\$}b)^{k+1}C$ solve $X(0)$ and $X(k+1)$, respectively. By RSP this implies that C *defines* a counter. As regards the *half-counter*, we simply adopt $((a^{\sharp}b)c)^*d$ as its definition (cf. [13]). In the following we show that both the counter and the half-counter can be defined without $*$.

Theorem 5. *Let* $A \supseteq \{a,b,c,d\}$.

1. *A counter* $(a(a^{\$}b)+c)^*d$ *can be defined in* $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}},\gamma)$ *with handshaking only and the actions in A not subject to communication if* $|A^{\mathrm{ext}}\backslash A| = 5$.
2. *A half-counter* $((a^{\sharp}b)c)^*d$ *can be defined in* $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$ *with handshaking and the actions in A not subject to communication if* $|A^{\mathrm{ext}}\backslash A| = 7$.

Proof. As for 1: let $A^{\mathrm{ext}}\backslash A = \{t\} \cup \{r_i, s_i \mid i = 0,1\}$, and consider

$(a(a^{\$}b)+c)s_1 + d$ abbreviated by P,

$(r_1P)^{\$}r_0$ abbreviated by Q,

$(r_1s_1)^{\$}s_0$ abbreviated by R.

Then it follows with RSP that $(a(a^{\$}b)+c)^*d = \tau_{\{t\}} \circ \partial_{\{r_i,s_i \mid i=0,1\}}(PQ \,\|\, R)$.

As for 2: let $A^{\mathrm{ext}}\backslash A = \{t\} \cup H$, where $H = \{r_i, s_i \mid i = 0,1,2\}$, and consider

$(a^{\sharp}b)cs_1 + ds_0$ abbreviated by S,

$(r_1S + s_2r_2)^{\sharp}r_0$ abbreviated by T,

$(r_1s_1 + r_2s_2)^{\sharp}(r_0s_0)$ abbreviated by U.

With an application of RSP it follows that $((a^{\sharp}b)c)^*d = \tau_{\{t\}} \circ \partial_H(ST \,\|\, U)$. □

5. Expressing a stack

We provide specifications of a *stack* over a finite data type in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}},\gamma)$ for $\diamond \in \{\$,\sharp\}$ with help of a regular control process and two (half-)counters.

Let $D = \{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ for some $N \in \mathbb{N}\backslash\{0\}$ be a finite set of data elements, ranged over by d. We assume that the values *empty* and *stop* are not in D. Let furthermore D^* be the set of finite strings over D, ranged over by w, and let ε denote the empty string. A standard recursive specification of a stack over D with empty-testing and

termination option defines this process as the solution for $X(\varepsilon)$ in the data-parametric linear specification

$$X(\varepsilon) = \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot X(\mathsf{d}_j) + s(\mathit{empty}) \cdot X(\varepsilon) + r(\mathit{stop}),$$

$$X(\mathsf{d}w) = \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot X(\mathsf{d}_j\mathsf{d}w) + s(\mathsf{d}) \cdot X(w).$$

Here the contents of the stack is modeled by the parameter value: $X(\mathsf{d}w)$ represents the stack that contains $\mathsf{d}w$ with d on top. Action $r(\mathsf{d})$ (receive d) models the push of d onto the stack, and action $s(\mathsf{d})$ (send d) represents deletion of d from the stack. Action $s(\mathit{empty})$ models empty-testing of the (empty) stack, and action $r(\mathit{stop})$ models termination of the (empty) stack. A non-terminating or non-empty-testing stack over D can be obtained by leaving out the concerning summands. In case $N=1$ ($D=\{\mathsf{d}_1\}$), the equations above specify a counter: the stack contents then models the counter value.

Theorem 6. *A stack over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication.*

Proof. Let a stack be defined as above. Without loss of generality we assume that $N>1$ in $D=\{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ (otherwise a counter does the job). Our approach is to encode the contents of the stack, i.e., elements from D^*, by natural numbers according to the following Gödel numbering $\ulcorner\_\urcorner : D^* \to \mathbb{N}$:

$$\ulcorner\varepsilon\urcorner = 0,$$
$$\ulcorner\mathsf{d}_j w\urcorner = j + N \cdot \ulcorner w\urcorner.$$

This encoding is a bijection with inverse $\mathit{deco}(\_)$ (let $\bigstar$ denote concatenation of strings):

$$\mathit{deco}(n) = \begin{cases} \varepsilon & \text{if } n = 0, \\ \mathsf{d}_N \bigstar \mathit{deco}(\frac{n-N}{N}) & \text{if } n \neq 0 \text{ and } n \bmod N = 0, \\ \mathsf{d}_{(n \bmod N)} \bigstar \mathit{deco}(\frac{n-(n \bmod N)}{N}) & \text{otherwise.} \end{cases}$$

So in case $N=3$, e.g., $\ulcorner\mathsf{d}_3\mathsf{d}_1\mathsf{d}_2\urcorner = 24$, and $\mathit{deco}(32) = \mathsf{d}_2\mathsf{d}_1\mathsf{d}_3 \in \{\mathsf{d}_1,\mathsf{d}_2,\mathsf{d}_3\}^*$.

For $j=1,2$, let C_j be a counter with add action $\bar{a}_j$, subtract action $\bar{b}_j$, zero testing action $\bar{c}_j$, and termination action $\bar{d}_j$ all in $A^{\mathrm{ext}}\backslash A$ (cf. Theorem 5.1). Let furthermore process terms $C_j(k)$ be such that they represent the typical 'counter states' in the following way ($k \in \mathbb{N}$):

$$C_j = C_j(0),$$
$$C_j(0) = \bar{a}_j C_j(1) + \bar{c}_j C_j(0) + \bar{d}_j,$$
$$C_j(k+1) = \bar{a}_j C_j(k+2) + \bar{b}_j C_j(k).$$

We further define a regular control process R_ε with actions $a_j,\ b_j,\ c_j,\ d_j \in A^{\mathrm{ext}}\backslash A$ and those of the stack. In combination with the counters C_1 and C_2, we use the process R_ε to define the stack. Note that the coding discussed above does not occur explicitly in R_ε.

$$R_\varepsilon = \sum_{j=1}^{N} r(\mathsf{d}_j)\cdot a_1^j \cdot R_j + s(empty)\cdot R_\varepsilon + r(stop)\cdot d_1 \cdot d_2$$

and for $k = 1,\ldots,N$:

$$\begin{aligned}
&R_k = \sum_{j=1}^{N} r(\mathsf{d}_j)\cdot \mathrm{Push}_j + s(\mathsf{d}_k)\cdot \mathrm{Pop}_k\\
&\mathrm{Push}_k = S_2^1 \cdot a_1^k \cdot S_1^{N_2} \cdot R_k\\
&\mathrm{Pop}_k = b_1^k \cdot S_2^{\frac{1}{N}} \cdot T_\varepsilon\\
&S_2^1 = (b_1 a_2)^* c_1 && \text{(shift the 'value' of } C_1 \text{ to } C_2),\\
&S_1^{N2} = (b_2(a_1^N))^* c_2 && \text{(shift the } N\text{-fold of } C_2 \text{ to } C_1),\\
&S_2^{1/N} = ((b_1^N)a_2)^* c_1 && \text{(shift the number of } N\text{-folds of } C_1 \text{ to } C_2),\\
&T_\varepsilon = b_2 a_1 T_1 + c_2 R_\varepsilon && \text{(test whether the stack is empty,}\\
&T_1 = b_2 a_1 T_2 + c_2 R_1 && \text{or which } D\text{-element is on top)},\\
&T_2 = b_2 a_1 T_3 + c_2 R_2\\
&\vdots\\
&T_N = b_2 a_1 T_1 + c_2 R_N.
\end{aligned}$$

Let γ for $j = 1,2$ be defined on $(A^{\mathrm{ext}}\backslash A)^2$ by $\gamma(x_j,\bar{x}_j) = t$ for $x \in \{a,b,c,d\}$, and let $H = \{x_j,\bar{x}_j \mid x \in \{a,b,c,d\}\}$. We show that

$$\tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1(0) \| C_2(0)),$$

solves the equation for $X(\varepsilon)$, and thus defines the stack:

$$\begin{aligned}
&\tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1(0) \| C_2(0))\\
&\quad = \sum_{j=1}^{N} r(\mathsf{d}_j)\cdot \tau_{\{t\}} \circ \partial_H(a_1^j \cdot R_j \| C_1(0) \| C_2(0))\\
&\qquad + s(empty)\cdot \tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1(0) \| C_2(0))\\
&\qquad + r(stop)\cdot \tau_{\{t\}} \circ \partial_H(d_1 \cdot d_2 \| C_1(0) \| C_2(0))
\end{aligned}$$

$$= \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot \tau^{j+1} \cdot \tau_{\{t\}} \circ \partial_H(R_j \| C_1(j) \| C_2(0))$$

$$+ s(empty) \cdot \tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1(0) \| C_2(0)) + r(stop) \cdot \tau \cdot \tau$$

$$= \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot \tau_{\{t\}} \circ \partial_H(R_j \| C_1(\ulcorner d_j \urcorner) \| C_2(0))$$

$$+ s(empty) \cdot \tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1(0) \| C_2(0)) + r(stop).$$

We are done if $\tau_{\{t\}} \circ \partial_H(R_j \parallel C_1(\ulcorner \mathsf{d}_j w \urcorner) \parallel C_2(0))$ solves the equation for $X(\mathsf{d}_j w)$ for each $j = 1, \ldots, N$ and $w \in D^*$ (and thus defines the stack with contents $\mathsf{d}_j w$). We prove this by first omitting the $\tau_{\{t\}}$-operation, and analyzing $\partial_H(R_j \parallel C_1(\ulcorner \mathsf{d}_j w \urcorner) \parallel C_2(0))$. This analysis is arranged in a graphical style in Fig. 1, where $P \to^a Q$ represents the statement $P = a \cdot Q$ for some $a \in A$, $P \twoheadrightarrow^\sigma Q$ represents $P = \sigma \cdot Q$ for σ some sequential process term, and branching from expressions represents application of $+$. So the uppermost process term in Fig. 1 with its arrows and resulting process terms represents the obviously derivable equation

$$\partial_H(R_j \| C_1(\ulcorner \mathsf{d}_j w \urcorner) \| C_2(0))$$

$$= \sum_{k=1}^{N} r(\mathsf{d}_k) \cdot \partial_H(\mathrm{Push}_k \| C_1(\ulcorner \mathsf{d}_j w \urcorner) \| C_2(0))$$

$$+ s(\mathsf{d}_j) \cdot \partial_H(\mathrm{Pop}_j \| C_1(\ulcorner d_j w \urcorner) \| C_2(0)).$$

From the derivation displayed in Fig. 1 and the axiom $x = x\tau$ (B1), it follows that

$$\tau_{\{t\}} \circ \partial_H(R_j \| C_1(\ulcorner \mathsf{d}_j w \urcorner) \| C_2(0)),$$

solves the equation for $X(\mathsf{d}_j w)$ ($j = 1, \ldots, N$ and $w \in D^*$). So by RSP it follows that

$$\tau_{\{t\}} \circ \partial_H(R_\varepsilon \| C_1 \| C_2),$$

defines the stack over D. By Theorem 3 it follows that once D is fixed, R_ε and hence the stack can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$ for some $A^{\mathrm{ext}} \supseteq A$, with handshaking only defined on $A^{\mathrm{ext}} \backslash A$. □

Theorem 7. *A stack over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication.*

Proof. In [13] it is proved that a stack over a finite data type can be defined in $\mathsf{ACP}^{\tau * \sharp}(A^{\mathrm{ext}}, \gamma)$ (so involving binary Kleene star) by means of the parallel composition of two half-counters and a regular control process. In that proof the empty-testing option does not occur, but this facility can be added in exactly the same way (in the

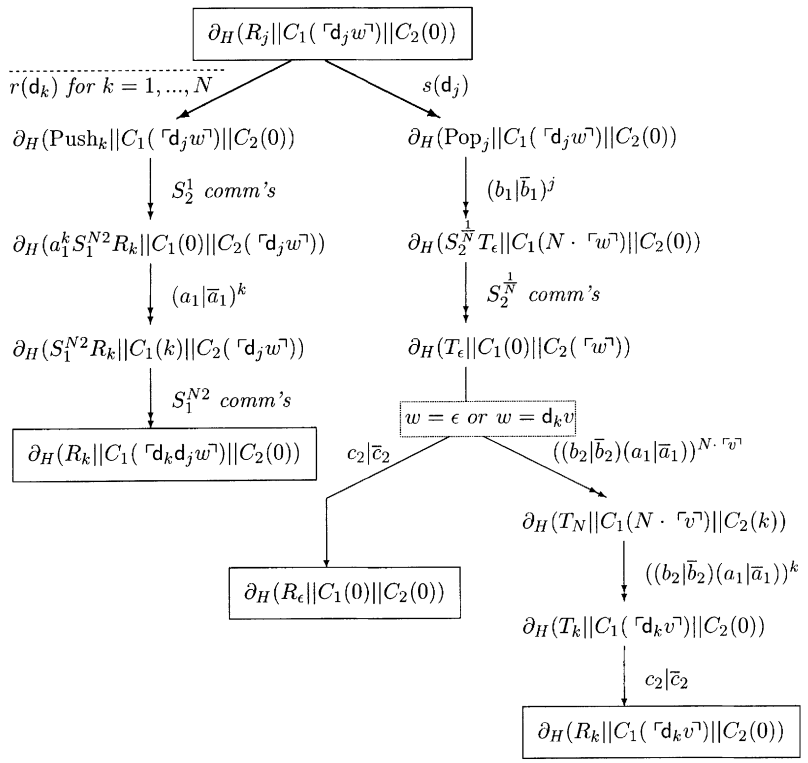


Fig. 1. Calculations with $\partial_H(R_j \parallel C_1(\ulcorner \mathsf{d}_j w\urcorner) \parallel C_2(0))$.

concerning regular control process) as above. So, it remains to be argued that all $*$-occurrences in the proof in [13] can be avoided. For the regular control process this follows immediately from Theorem 4, and for a half-counter from Theorem 5.2. □

6. Expressing context-free processes

A process is *context-free over* A_δ if it is a solution for a specification in *restricted Greibach normal form* (rGNF): a specification is in rGNF (over A_δ) if it is of the form

$$X_i = \sum_{j=1}^{n} \left(\sum_{k=1}^{n} \alpha_{i,j,k} X_j X_k + \beta_{i,j} X_j \right) + \gamma_i \tag{3}$$

for $i=1,\ldots,n$ $(n \geqslant 1)$, where $\alpha_{i,j,k}$, $\beta_{i,j}$ and γ_i are finite sums of actions or δ. Note that a regular process is context-free. (See [9] for more information on rGNFs, and for an interesting decidability result.) In particular, rGNF specifications are guarded and RSP can be applied. As an example, $a^\sharp b$ is context-free because it is a solution for X_1 in

$X_1 = aX_1X_2 + b$, $X_2 = a$. Also, a stack over $\{\mathsf{d}_1, \ldots, \mathsf{d}_N\}$ is context-free: each solution for $X(\varepsilon)$ in the standard specification of a stack (see the previous section) is also a solution for X_0 in the rGNF specification

$$X_0 = \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot X_j X_0 + s(\mathit{empty}) \cdot X_0 + r(\mathit{stop}),$$

$$X_k = \sum_{j=1}^{N} r(\mathsf{d}_j) \cdot X_j X_k + s(\mathsf{d}_k) \quad (k = 1, \ldots, N).$$

In this section we prove that each context-free process can be expressed in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}}, \gamma)$ for $\diamond \in \{\$, \sharp\}$ with help of a stack.

Theorem 8. *Let P be a context-free process over $A_{\delta\tau}$. Then P can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$ and in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication.*

Proof. Let the context-free process P_1 over A_δ be given as a solution for X_1 in specification (3) above. Because this specification contains $n > 0$ variables (and equations), we use a data type containing n values, say $D = \{\mathsf{d}_1, \ldots, \mathsf{d}_n\}$. Let process term $S(\varepsilon)$ define a stack over D without empty-testing and with termination action $r(\mathit{stop})$ (cf. Theorem 6). Furthermore, let the following process terms represent the typical states of the stack ($\mathsf{d} \in D$ and $w \in D^*$):

$$S(\varepsilon) = \sum_{j=1}^{n} r(\mathsf{d}_j) \cdot S(\mathsf{d}_j) + r(\mathit{stop}),$$

$$S(\mathsf{d}w) = \sum_{j=1}^{n} r(\mathsf{d}_j) \cdot S(\mathsf{d}_j \mathsf{d} w) + s(\mathsf{d}) \cdot S(w).$$

Now consider for $i = 1, \ldots, n$ the process terms

$$\sum_{j=1}^{n} \left(\sum_{k=1}^{n} \alpha_{i,j,k} \cdot s(\mathsf{d}_k) \cdot s(\mathsf{d}_j) + \beta_{i,j} \cdot s(\mathsf{d}_j) \right) + \gamma_i \quad \text{abbreviated by } F_i$$

and the regular process R defined by

$$R = \left(\sum_{j=1}^{n} r(\mathsf{d}_j) \cdot F_j \right)^* s(\mathit{stop}).$$

By Theorems 6 and 3 there exists a finite extension A^{ext} of A such that $S(\varepsilon)$ and R are specifiable in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}}, \gamma)$, and for $\mathsf{e} \in E = D \cup \{\mathit{stop}\}$, $(A^{\mathrm{ext}} \backslash A) \supseteq H = \{r(\mathsf{e}), s(\mathsf{e}) | \mathsf{e} \in E\}$ and $\gamma(r(\mathsf{e}), s(\mathsf{e})) = t$. By Theorems 7 and 4, there exists a finite extension A^{ext} of A such that the same holds for $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}}, \gamma)$. By RSP it follows that $P_i = \tau_{\{t\}} \circ \partial_H(F_i \cdot R \parallel S(\varepsilon))$. $\square$

7. Expressing bags and queues

In this section we consider two typical processes that are not context-free: a *bag* and a *queue*.[2] With help of stacks we provide specifications for both these processes in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}},\gamma)$ for $\diamond\in\{\$,\sharp\}$.

First we consider a bag or *multiset*. Let again $D=\{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ for some $N\in\mathbb{N}\backslash\{0\}$ be a finite set of data elements, ranged over by d, and let ρ range over $\mathbb{N}^N$, where ρ_k denotes the kth component of ρ for $k=1,\ldots,N$. We use ρ to encode a bag with ρ_k occurrences of d_k. In order to give a straightforward specification of the bag over D, we define the function $\Uparrow:\mathbb{N}^N\times\{1,\ldots,N\}\to\mathbb{N}^N$ for modeling insertion in a component-wise fashion:

$$(\rho\Uparrow j)_k=\begin{cases}\rho_k & \text{if } k\neq j,\\ \rho_k+1 & \text{otherwise } (1\leqslant k\leqslant N).\end{cases}$$

The function $\Downarrow:\mathbb{N}^N\times\{1,\ldots,N\}\to\mathbb{N}^N$ for modeling deletion is defined by

$$(\rho\Downarrow j)_k=\begin{cases}\rho_k & \text{if } k\neq j \text{ or } (k=j \text{ and } \rho_k=0),\\ \rho_k-1 & \text{otherwise } (1\leqslant k\leqslant N).\end{cases}$$

Let $\lambda\in\mathbb{N}^N$ denote the sequence of N zero's. Then for $\rho\in\mathbb{N}^N\backslash\{\lambda\}$, the bag over D with empty-testing and termination option is defined as the solution for $X(\lambda)$ in the data-parametric linear specification

$$X(\lambda)=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot X(\lambda\Uparrow j)+s(\mathit{empty})\cdot X(\lambda)+r(\mathit{stop}),$$

$$X(\rho)=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot X(\rho\Uparrow j)+\sum_{\{i\mid\rho_i>0\}} s(\mathsf{d}_i)\cdot X(\rho\Downarrow i).$$

Here the contents of the bag is modeled by the parameter value: $X(\rho)$ represents the bag that contains ρ_i occurrences of d_i. Action $r(\mathsf{d})$ (receive d) models insertion of d in the bag, and action $s(\mathsf{d})$ (send d) models deletion of d from the bag. As with the stack, action $s(\mathit{empty})$ models empty-testing, and action $r(\mathit{stop})$ models termination of the (empty) bag. A non-terminating or non-empty-testing bag over D can be obtained by leaving out the concerning summands. In case $N=1$, the equations above again specify a counter: the bag contents then models the counter value.

Theorem 9. *A bag or multiset over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}},\gamma)$ and in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication.*

[2] A bag (queue) over more than one data element is not context-free, (see [16] [7], respectively).

Proof. Let a bag over $D=\{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ be defined as above, and let $\sigma\in\{0,1\}^N$. We define similar functions $\uparrow$ and $\downarrow$ as $\Uparrow$ and $\Downarrow$ above (note that $|\{0,1\}^N|=2^N$ is finite), again in a component-wise fashion:

$$(\sigma\uparrow j)_k=\begin{cases}\sigma_k & \text{if } k\neq j,\\ 1 & \text{otherwise } (1\leqslant k\leqslant N),\end{cases}$$

$$(\sigma\downarrow j)_k=\begin{cases}\sigma_k & \text{if } k\neq j,\\ 0 & \text{otherwise } (1\leqslant k\leqslant N).\end{cases}$$

Now an element of $\{0,1\}^N$ encodes per component k whether or not d_k is present in the bag (by value 1 or 0, respectively). Consider counters $C_1,\ldots,C_N$ for occurrence counting of $\mathsf{d}_1,\ldots,\mathsf{d}_N$, where C_j has add-action $\bar{a}_j$, subtract-action $\bar{b}_j$, zero-testing $\bar{c}_j$, and stop-action $\bar{d}_j$, and the regular control process R_λ defined by the following 2^N equations ($\sigma\in\{0,1\}^N\backslash\{\lambda\}$):

$$R_\lambda=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot a_j\cdot R_{\lambda\uparrow j}+s(\mathit{empty})\cdot R_\lambda+r(\mathit{stop})\cdot d_1\cdot\ldots\cdot d_N,$$

$$R_\sigma=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot a_j\cdot R_{\sigma\uparrow j}+\sum_{\{i\mid\sigma_i=1\}} s(\mathsf{d}_i)\cdot b_i\cdot(c_i\cdot R_{\sigma\downarrow i}+b_i\cdot a_i\cdot R_\sigma).$$

By Theorems 3 and 5.1 (4 and 7), the process R_λ and the C_j can be specified in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}},\gamma)$ ($\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$, respectively: recall that a counter is a stack over a singleton data type) for a suitable extension A^{ext}. Let $H=\{x_j,\bar{x}_j\mid x\in\{a,b,c,d\},\ j=1,\ldots,N\}$ and $\gamma(x_j,\bar{x}_j)=t$ for $x\in\{a,b,c,d\},\ j=1,\ldots,N$. It is not hard to show that

$$\tau_{\{t\}}\circ\partial_H(R_\lambda\|C_1\|\ldots\|C_N),$$

solves the equation for $X(\lambda)$. By RSP this yields the definition of the bag over D. □

We finish this section with the specification of a queue over a finite data type $D=\{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ ($N>0$) using only either \$ or ♯. The queue over D with empty-testing and termination option is defined as the solution for $X(\varepsilon)$ in the data-parametric linear specification

$$X(\varepsilon)=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot X(\mathsf{d}_j)+s(\mathit{empty})\cdot X(\varepsilon)+r(\mathit{stop}),$$

$$X(w\mathsf{d})=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot X(\mathsf{d}_j w\mathsf{d})+s(\mathsf{d})\cdot X(w).$$

Here the contents of the queue is modeled by the parameter value: $X(w\mathsf{d})$ defines the queue that contains $w\mathsf{d}$ with d on top. Action $r(\mathsf{d})$ (receive d) models insertion of d in the queue, and action $s(\mathsf{d})$ (send d) represents deletion of the top d from the queue. Action $s(\mathit{empty})$ models empty-testing, and action $r(\mathit{stop})$ models termination of the

(empty) queue. A non-terminating or non-empty-testing queue over D can be obtained by leaving out the concerning summands. If $N=1$, the equations above specify a counter: the queue contents then models the counter value.

Theorem 10. *A queue over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}},\gamma)$ and in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication.*

Proof. Let a queue over $D=\{\mathsf{d}_1,\ldots,\mathsf{d}_N\}$ be defined as above. We use two stacks $S_1(\varepsilon)$ and $S_2(\varepsilon)$ over D with empty-testing and stop-facilities. For $i=1,2$, a push-action of d onto $S_i(w)$ is modeled by the action $r_i(\mathsf{d})$, a pop-action by $s_i(\mathsf{d})$, an empty-test by $\bar{c}_i$, and a termination action by $\bar{d}_i$.

The idea is to define regular processes R_ε and R_{d} ($\mathsf{d}\in D$) such that

$$\tau_{\{t\}}\circ\partial_H(R_\varepsilon\|S_1(\varepsilon)\|S_2(\varepsilon)),$$

solves $X(\varepsilon)$, and

$$\tau_{\{t\}}\circ\partial_H(R_{\mathsf{d}}\|S_1(w\mathsf{d})\|S_2(\varepsilon)),$$

solves $X(w\mathsf{d})$, thus represents the queue with contents $w\mathsf{d}$ and d on top. Deletion is modeled by shifting $S_1(w\mathsf{d})$ to $S_2(\mathsf{d}\tilde{w})$ where $\tilde{w}$ is the reverse of w, deleting d, and shifting back $S_2(\tilde{w})$ to $S_1(w)$. Consider the regular control process R_ε defined by the following $N+4$ equations ($\mathsf{d}\in D$):

$$R_\varepsilon=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot s_1(\mathsf{d}_j)\cdot R_{\mathsf{d}_j}+s(\mathit{empty})\cdot R_\varepsilon+r(\mathit{stop})\cdot d_1\cdot d_2,$$

$$R_{\mathsf{d}}=\sum_{j=1}^{N} r(\mathsf{d}_j)\cdot s_1(\mathsf{d}_j)\cdot R_{\mathsf{d}}+s(\mathsf{d})\cdot \mathit{Shift}_2^1\cdot r_2(\mathsf{d})\cdot \mathit{Next},$$

$$\mathit{Next}=\sum_{j=1}^{N} r_2(\mathsf{d}_j)\cdot s_1(\mathsf{d}_j)\cdot \mathit{Shift}_1^2\cdot R_{\mathsf{d}_j}+c_2\cdot R_\varepsilon,$$

$$\mathit{Shift}_2^1=\left(\sum_{j=1}^{N} r_1(\mathsf{d}_j)\right)\cdot s_2(\mathsf{d}_j))^* c_1,$$

$$\mathit{Shift}_1^2=\left(\sum_{j=1}^{N} r_2(\mathsf{d}_j)\right)\cdot s_1(\mathsf{d}_j))^* c_2.$$

By Theorems 3 and 6 (4 and 7), R_ε, $S_1(\varepsilon)$ and $S_2(\varepsilon)$ can be specified in $\mathsf{ACP}^{\tau\$}(A^{\mathrm{ext}},\gamma)$ or in $\mathsf{ACP}^{\tau\sharp}(A^{\mathrm{ext}},\gamma)$, respectively, for a suitable extension A^{ext}.

Now let $H=\{r_i(\mathsf{d}),s_i(\mathsf{d}),c_i,\bar{c}_i,d_i,\bar{d}_i \mid i\in\{1,2\},\ \mathsf{d}\in D\}$, and $\gamma(r_i(\mathsf{d}),s_i(\mathsf{d}))=\gamma(c_i,\bar{c}_i)=\gamma(d_i,\bar{d}_i)=t$ for $i=1,2$. It is easily seen that

$$\tau_{\{t\}}\circ\partial_H(R_\varepsilon\|S_1(\varepsilon)\|S_2(\varepsilon)),$$

is a solution for $X(\varepsilon)$ in the specification of the queue. So by RSP this yields the definition of the queue over D. $\square$

8. Back and forth iteration

In this section we show that all standard processes discussed before can also be specified in $\mathsf{ACP}^{\tau\leftrightarrows}(A,\gamma)$. Recall that the back and forth operation $\leftrightarrows$ is defined by

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

We follow the same approach as before, and first show that each regular process over A_δ can be expressed in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}},\gamma)$ for a suitable set A^{ext} of actions.

Theorem 11. *For each regular process P over A_δ there exists a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}},\gamma)$ with handshaking only, and the actions in A not subject to communication.*

Proof. Let P_1 be a regular process over A_δ given by $P_i = \sum_{j=1}^{n} \alpha_{i,j}P_j + \beta_i$ $(i = 1,\ldots,n)$ where $\alpha_{i,j}$ and β_i are finite sums of actions or δ. Define A^{ext} as the extension of A with the following $2n+3$ actions:

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

and consider the following processes:

$$\sum_{j=1}^{n} \alpha_{i,j}s_j + \beta_i \text{ abbreviated by } F_i \text{ for } i = 1,\ldots,n,$$

$$\left(\sum_{j=1}^{n} r_jF_j\right)^{\leftrightarrows} r_0 \text{ abbreviated by } K,$$

$$\left(\sum_{j=1}^{n} r_js_j\right)^{\leftrightarrows} s_0 \text{ abbreviated by } L.$$

Then it follows with RSP that $P_1 = \tau_{\{t\}} \circ \partial_H(F_1K \| L)$. $\square$

Next, we introduce "bfi-counters".

Theorem 12. *Let $A \supseteq \{a,b,c,d\}$. A bfi-counter $((a^{\leftrightarrows}b)c)^*d$ can be defined in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}},\gamma)$ with handshaking only and the actions in A not subject to communication if $|A^{\mathrm{ext}}\backslash A| = 5$.*

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

$$(a^{\leftrightarrows}b)cs_1 + d \text{ abbreviated by } S$$

$(r_1S)^{\leftrightarrows}r_0$ abbreviated by T

$(r_1s_1)^{\leftrightarrows}s_0$ abbreviated by U.

Then $((a^{\leftrightarrows}b)c)^*d = \tau_{\{t\}} \circ \partial_H(ST \| U)$ follows with RSP. □

In a similar way as was done in [13], but using two bfi-counters instead of half-counters, one can model a stack over a finite data type with actions in A in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}}, \gamma)$ $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}}, \gamma)$, and handshaking communication only over the finite set $A^{\mathrm{ext}} \backslash A$. As a consequence, all previously mentioned standard processes over A_δ can be specified in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}}, \gamma)$ for a suitable choice of A^{ext}.

Theorem 13. *Let P be either a stack, a context-free process, a bag or a queue with actions in A_δ. Then there exists a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\leftrightarrows}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, and the actions in A not subject to communication.*

9. Expressiveness

In this section we discuss some general questions concerning the expressive power of $\mathsf{ACP}^{\tau}(A, \gamma)$ with recursive operations. First we recall some basic results on $\mathsf{ACP}^{\tau *}(A, \gamma)$. Then we argue that each *computable process* over A_δ—i.e., a process that can be characterized by a total recursive function describing for each state the next steps and resulting states in terms of an appropriate encoding (cf. [38]) — can be specified in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}}, \gamma)$ for $\diamond \in \{\$, \sharp, \leftrightarrows\}$. Finally we prove that abstraction is necessary for this result.

In [13] it is proved that each regular process over A_δ can be specified in $\mathsf{ACP}^{\tau *}(A^{\mathrm{ext}}, \gamma)$ with handshaking only, A^{ext} a finite extension of A, and the actions in A not subject to communication (for a short proof see [15, Theorem 2.1]). Moreover, abstraction and the law $x\tau = x$ are necessary for this result. Furthermore, it is shown in [13] that no non-regular processes over A_δ (such as, e.g., $a^{\$}b$) can be specified in $\mathsf{ACP}^{\tau *}(A, \gamma)$.

In [8], Baeten et al. show that each computable process can be specified in $\mathsf{ACP}(A, \gamma)$ with abstraction and guarded, finite recursive specifications. Their proof is based on a modeling of Turing machine computation with two stacks and a regular control process, and although it refers to rooted weak bisimulation equivalence, it holds as well for rooted branching bisimulation equivalence (as defined in [29]). It follows that each computable process over A_δ can be specified in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}}, \gamma)$ for $\diamond \in \{\$, \sharp, \leftrightarrows\}$ and A^{ext} a finite extension of A, using only handshaking over $A^{\mathrm{ext}} \backslash A$.

In this paper we do not explicitly introduce models – that is, process algebras – for $\mathsf{ACP}^{\tau\diamond}(A, \gamma)$ (for the sake of completeness, we do provide *transition rules* for the new operations in the next section). Nevertheless it is not hard to show that abstraction is indispensable for the above-mentioned type of expressiveness results. This can be argued as follows (cf. [8]). Let $A \supseteq \{a, b\}$ and let $\mathscr{P}$ be the set of process terms (closed terms)

over $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$, i.e., $\mathsf{ACP}(A,\gamma)$ extended with *all* recursive operations occurring in this paper. By a diagonalization argument we can define a computable process over A that cannot be expressed in $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$. This is even the case if a very liberal behavioural equivalence is adopted. In order to show this, let $\ulcorner\_\urcorner : \mathscr{P} \to \mathbb{N}$ encode all process terms over $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$, and let $proc : \mathbb{N} \to \mathscr{P}$ be a function such that $proc(\ulcorner P \urcorner) = P$ for all $P \in \mathscr{P}$. Furthermore, let the function $f : \mathbb{N} \times \mathscr{P} \to \mathscr{P}$ be such that $f(n,P)$ characterizes which actions can possibly be executed after n steps (for $a \in A_\delta$):

$$f(0,a) = a,$$

$$f(n+1,a) = \delta,$$

$$f(0,ax) = a,$$

$$f(n+1,ax) = f(n,x),$$

$$f(n,x+y) = f(n,x) + f(n,y).$$

So for each n and P, $f(n,P)$ equals $a \in A_\delta$ or a sum of atomic actions. The definition of this function is adequate because each process term in $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$ can be equated to one in *head normal form*, i.e., in the form

$$P = \sum_{i=1}^{k} a_i P_i + \sum_{j=1}^{l} b_j,$$

where the a_i, b_j are actions and empty sums equal δ (cf. [11]). Finally, let $\sim$ be a behavioural equivalence for $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$ in the range from strong bisimilarity up to trace equivalence (see [26, 28]), and consider the data-parametric linear specification (for k ranging over $\mathbb{N}$)

$$X(k) = \begin{cases} a \cdot X(k+1) & \text{if } f(k, proc(k)) \sim f(k, proc(k)) + b, \\ b \cdot X(k+1) & \text{otherwise.} \end{cases}$$

Clearly, each solution P for $X(0)$ is a computable process that repeatedly executes either a or b. It is easily seen that $P \not\sim proc(k)$ for all $k \in \mathbb{N}$: suppose the contrary and let n be such that $P \sim proc(n)$. If $f(n,proc(n)) \sim f(n,proc(n)) + b$, then $proc(n)$ has an execution trace in which after n steps the action b can be executed, whereas P can only perform a after n steps (characterized by the equation for $X(n)$). If $f(n,proc(n)) \not\sim f(n,proc(n)) + b$, then b is not possible after n execution steps of $proc(n)$, whereas P can do a b after n steps. So, P cannot be expressed in $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$ modulo $\sim$. Summing up, we state the following result.

Theorem 14. *For each computable process P over A_δ there exists a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\diamond}(A^{\mathrm{ext}},\gamma)$ for $\diamond \in \{\$, \sharp, \leftrightarrows\}$ with*

handshaking only, and the actions in A not subject to communication. Furthermore, abstraction and the law $x\tau = x$ are essential for this result.

Adopting the explicit standard semantics for $\mathsf{ACP}(A,\gamma)$ (SOS-semantics with strong bisimilarity, see [11, 24]) and the transition rules for all recursive operations, the necessity of abstraction can be shown in a more direct way: for any $A \supseteq \{a,b\}$, the regular process P specified by

$$P = aQ + b,$$
$$Q = aP + a$$

(see Section 1) cannot be expressed in $\mathsf{ACP}^{*,\$,\sharp,\leftrightarrows}(A,\gamma)$. This can be proved by an analysis of properties yielded by the particular transition rules involved (cf. [13, 14]).

10. Conclusions

In process algebra, a (potentially) infinite process is traditionally represented as a solution for a variable in a system of guarded recursive equations, and proof theory and verification tend to focus on reasoning about such recursive systems. Although specification and verification of concurrent processes defined in this way serve their purpose well, recursive *operations* constitute a more direct representation and are easier to comprehend. In 1984, Milner was the first to consider the unary Kleene star in process algebra [36]. An early axiomatic approach to a restricted form of (binary) iteration in the realm of process algebra is Hennessy's treatment of delay operators [31]. For an overview of process algebra with iterators we refer to [14].

In this paper we showed that adding one of $\$, \sharp, \leftrightarrows$ as a primitive to ACP with abstraction yields an expressive format, and that abstraction is a necessary feature. These results support the equational founding of process algebra: any computable process (over a finite set of actions) can simply be represented by a term. Adding binary Kleene star as well yields a more flexible format for the specification of concurrent processes, and a setting in which each operation embodies some distinct and intuitive idea. If one adopts the natural point of view that a *binary iterator* $F(x,y)$ in process algebra must satisfy the general format

$$F(x,y) = x \cdot C[x,y,F(x,y)] + y$$

with $C[x,y,F(x,y)]$ a context expressing some sequential term and containing $F(x,y)$, then it is not hard to see that $\$, \sharp$ and $\leftrightarrows$ are the simplest candidates for a non-regular, binary iterator. Of course, the binary Kleene star is the simplest candidate for a regular, binary iterator.

We notice that in the case of $\mathsf{ACP}^{\tau *\$}(A,\gamma)$ we could have used a more direct way to analyze the expressive power: with regular processes and counters, it is quite straightforward to model *register machine computation*. Being able to implement each (unary)

recursive function in this way, it is not difficult to show that each computable process over A_δ can be specified in $\mathsf{ACP}^{\tau *\$}(A^{\mathrm{ext}},\gamma)$. However, such an approach does not by itself reveal how to define standard processes such as a stack. That is why we preferred to focus on the particular standard processes discussed in this paper, and to further refer to the expressiveness result in [8]. Note that our definitions of these standard processes do not rely upon *fairness*[3] *rules*: in none of the constructions the possibility of an infinite τ-trace occurs, and the only axiom on the silent step τ that we used is $x\tau = x$. Therefore, these results are preserved under all behavioral equivalences that respect this axiom, such as the rooted versions of branching bisimulation [29], delay bisimulation [34], η-bisimulation [10], weak bisimulation [37], and even in a setting that distinguishes infinite τ-traces from finite ones, such as *divergence sensitive branching bisimulation* [30].

In the remainder of this section we give brief consideration to some proof theoretical issues. First, we notice that there exist a finite set A of actions and communication function γ such that

$$\mathsf{ACP}^{\tau\diamond}(A,\gamma) \vdash t = u$$

with $\diamond \in \{\$, \sharp, \leftrightarrows\}$ is *undecidable* for process terms t, u. We sketch the proof: let V_1, V_2 be r.e. sets of natural numbers that are recursively inseparable. Using a register machine encoding one may provide families of process terms t_n, u_n (parameterized by n) such that

$$n \in V_1 \Rightarrow \mathsf{ACP}^{\tau\diamond}(A,\gamma) \vdash t_n = u_n,$$

$$n \in V_2 \Rightarrow t_n \ncong u_n,$$

where $\cong$ expresses branching bisimulation equivalence, the standard semantics for $\mathsf{ACP}^{\tau}(A,\gamma)$ [29]. With decidability of $\mathsf{ACP}^{\tau\diamond}(A,\gamma)$ one could obtain a recursive separation of V_1 and V_2, which is contradictory. If binary Kleene star is available, one can show (in a setting without τ and using strong bisimulation instead) that $\mathsf{ACP}^{*\diamond}(A,\gamma)$ is undecidable as well.

We come to an end with a short comment on RSP-variants for binary iterators. Let $\mathsf{RSP}^{\$}$ be the conditional rule

$$x = y(xx) + z \Rightarrow x = y^{\$}z.$$

It is an open question whether $\mathsf{BPA}(A)$, i.e., *basic process algebra* defined by axioms (A1)–(A5) of $\mathsf{ACP}(A,\gamma)$ (see Table 1) together with the axiom $x^{\$}y = x((x^{\$}y)(x^{\$}y)) + y$ and $\mathsf{RSP}^{\$}$ is complete with respect to strong bisimilarity over that signature. The four transition rules for $\$$ are

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

[3] Fairness models the assumption that in case of a τ-loop eventually some other alternative – if available – is executed. An example is captured by the law $\tau(\tau^* x) = \tau x$ (see [13]), valid in branching bisimilarity.

while the remaining transition rules and strong bisimulation equivalence are defined as usual (cf. [11]). For example, one easily derives the transition $(a\$b)^2 \xrightarrow{a}(a\$b)^2 \cdot (a\$b)$. (Note that $(a\$b)^2 \cdot (a\$b)$ and $(a\$b)^3 = (a\$b) \cdot (a\$b)^2$ are syntactically different, so some transitions in our illustrations in the introduction do not precisely reflect these rules.) The transition rules for $*$ and $\sharp$ are as expected (see [13]), and those for $\leftrightarrows$ are the following:

$$\frac{x \xrightarrow{a} \surd}{x \leftrightarrows y \xrightarrow{a}(x \leftrightarrows y)y}, \quad \frac{x \xrightarrow{a} \surd}{y \leftrightarrows x \xrightarrow{a} \surd}, \quad \frac{x \xrightarrow{a} x'}{x \leftrightarrows y \xrightarrow{a} x'((x \leftrightarrows y)y)}, \quad \frac{x \xrightarrow{a} x'}{y \leftrightarrows x \xrightarrow{a} x'}.$$

For the settings with $\sharp$ and $\leftrightarrows$ similar RSP-variants can be defined, and similar open questions can be raised. We notice that the equational axiomatization of $\mathsf{BPA}^*(A)$ [25], i.e., axioms (A1)–(A5) and (BKS1)–(BKS3) (see Table 1), implies that $\mathsf{BPA}(A)$ with (BKS1) and RSP^*, i.e.,

$$x = yx + z \Rightarrow x = y^*z,$$

is complete with respect to (strong) bisimilarity. It is not hard to prove (BKS2) and (BKS3):

$$\frac{\begin{aligned}(\boldsymbol{x^*y})\boldsymbol{z} &= (x(x^*y) + y)z \\ &= (x(x^*y))z + yz \\ &= x((\boldsymbol{x^*y})\boldsymbol{z}) + yz\end{aligned}}{(x^*y)z = x^*(yz)}\ \mathsf{RSP}^*$$

$$\frac{\begin{aligned}(\boldsymbol{x+y})^*\boldsymbol{z} &= (x+y)((x+y)^*z) + z \\ &= x((\boldsymbol{x+y})^*\boldsymbol{z}) + y((x+y)^*z) + z\end{aligned}}{(x+y)^*z = x^*(y((x+y)^*z) + z)}\ \mathsf{RSP}^*.$$

Extending $\mathsf{BPA}^*(A)$ with δ rules out a finite equational axiomatization of strong bisimulation equivalence (see [39]). Equational axiomatizations of bisimilarity for other $\mathsf{BPA}(A)$-oriented systems with some form of iteration can be found in [21, 2, 6, 22, 1, 27, 4, 5]. As for $\mathsf{ACP}^*(A,\gamma)$, note that also the axiom (BKS4) (i.e., $\partial_H(x^*y) = \partial_H(x)^*\partial_H(y)$) easily follows from RSP^*. Finally, if τ is involved, all RSP-variants mentioned above need a guardedness restriction. For example, $\tau = \tau(\tau\tau) + \delta$, though $\tau = \tau^{\$}\delta$ is not acceptable (τa can eventually perform a, whereas $(\tau^{\$}\delta)a$ cannot). The merits and demerits of RSP-based proof systems and the quest for complete proof systems are topics for future research (cf. [3]).

Acknowledgements

We thank Bas Luttik and the referees for useful comments.

References

[1] L. Aceto, W.J. Fokkink, An equational axiomatization for multi-exit iteration, Inform. and Comput. 137 (1997) 121–158.

[2] L. Aceto, W.J. Fokkink, R.J. van Glabbeek, A. Ingólfsdóttir, Axiomatizing prefix iteration with silent steps, Inform. and Comput. 127 (1) (1996) 26–40.

[3] L. Aceto, W.J. Fokkink, A. Ingólfsdóttir, A menagerie of non-finitely based process semantics over BPA^*: from ready simulation to completed traces, Mathe. Structures Comput. Sci. 8 (3) (1998) 193–230.

[4] L. Aceto, W.J. Fokkink, A. Ingólfsdóttir, A Cook's tour of equational axiomatizations for prefix iteration, in: M. Nivat (Ed.), Proc. 1st Conf. on Foundations of Software Science and Computation Structures (FoSSaCS '98), Lisbon, LNCS 1378, Springer, Berlin, 1998, pp. 20–34.

[5] L. Aceto, J.F. Groote, A complete equational axiomatization for MPA with string iteration, Theoret. Comput. Sci. 211 (1/2) (1999) 339–374.

[6] L. Aceto, A. Ingólfsdóttir, An equational axiomatization of observation congruence for prefix iteration, in: M. Wirsing, M. Nivat (Eds.), Proc. 5th Conf. on Algebraic Methodology and Software Technology (AMAST '96), Munich, LNCS 1101, Springer, Berlin, 1996, pp. 195–209.

[7] J.C.M. Baeten, J.A. Bergstra, Global renaming operators in concrete process algebra, Inform. Comput. 78 (3) (1988) 205–245.

[8] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, On the consistency of Koomen's fair abstraction rule, Theoret. Comput. Sci. 51 (1/2) (1987) 129–176.

[9] J.C.M. Baeten, J.A. Bergstra, J.W. Klop, Decidability of bisimulation equivalence for processes generating context-free languages, J. ACM 40 (3) (1993) 653–682.

[10] J.C.M. Baeten, R.J. van Glabbeek, Another look at abstraction in process algebra, in: T. Ottmann (Ed.), Proc. 14th Colloquium on Automata, Languages and Programming (ICALP'87), Karlsruhe, LNCS 267, Springer, Berlin, 1987, pp. 84–94.

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

[12] J.A. Bergstra, I. Bethke, A. Ponse, Process algebra with iteration. Technical Report P9314, Programming Research Group, University of Amsterdam, 1993. (At `http://www.science.uva.nl/research/prog/reports/reports.html.`)

[13] J.A. Bergstra, I. Bethke, A. Ponse, Process algebra with iteration and nesting, Comput. J. 37 (4) (1994) 243–258.

[14] J.A. Bergstra, W.J. Fokkink, A. Ponse, Process algebra with recursive operations, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, North-Holland, Amsterdam, to appear.

[15] J.A. Bergstra, J.A. Hillebrand, A. Ponse, Grid protocols based on synchronous communication, Sci. Comput. Programming 29 (1/2) (1997) 199–233.

[16] J.A. Bergstra, J.W. Klop, The algebra of recursively defined processes and the algebra of regular processes, in: J. Paredaens (Ed.), Proc. 11th ICALP, Vol. 172, Lecture Notes in Computer Science, Springer, 1984, pp. 82–95. An extended version appeared in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (Eds.), Algebra of Communicating Processes, Utrecht 1994, Workshops in Computing, Springer, Berlin, 1995, pp. 1–25.

[17] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inform. Control 60 (1/3) (1984) 109–137.

[18] J.A. Bergstra, J.W. Klop, Algebra of communicating processes with abstraction, Theoret. Comput. Sci. 37 (1) (1985) 77–121.

[19] J.A. Bergstra, J.V. Tucker, Top down design and the algebra of communicating processes, Sci. Comput. Programming 5 (2) (1984) 171–199.

[20] M.A. Bezem, A. Ponse, Two finite specifications of a queue, Theoret. Comput. Sci. 177 (2) (1997) 487–507.

[21] W.J. Fokkink, A complete equational axiomatization for prefix iteration, Inform. Process. Lett. 52 (1994) 333–337.

[22] W.J. Fokkink, A complete axiomatization for prefix iteration in branching bisimulation, Fundamenta Informaticae 26 (2) (1996) 103–113.

[23] W.J. Fokkink, Axiomatizations for the perpetual loop in process algebra, in: P. Degano, R. Gorrieri, A. Marchetti-Spaccamela (Eds.), Proc. 24th ICALP, Lecture Notes in Computer Science, Vol. 1256, Springer, Berlin, 1997, pp. 571–581.
[24] W.J. Fokkink, Introduction to Process Algebra, Springer, Berlin, 2000.
[25] W.J. Fokkink, H. Zantema, Basic process algebra with iteration: completeness of its equational axioms, Comput. J. 37 (4) (1994) 259–267.
[26] R.J. van Glabbeek, The linear time – branching time spectrum, in: J.C.M. Baeten, J.W. Klop (Eds.), Proc. 1st Conf. on Concurrency Theory (CONCUR '90), Amsterdam, LNCS 458, Springer, Berlin, 1990, pp. 278–297.
[27] R.J. van Glabbeek, Axiomatizing flat iteration, in: A. Mazurkiewicz, J. Winkowski (Eds.), Proc. 8th Conf. on Concurrency Theory (CONCUR '98), Warsaw, Lecture Notes in Computer Science, Vol. 1243, Springer, Berlin, 1997, pp. 228–242.
[28] R.J. van Glabbeek, The linear time – branching time spectrum I, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, North-Holland, Amsterdam, to appear.
[29] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.
[30] J.F. Groote, S.F.M. van Vlijmen, A modal logic for μCRL, in: A. Ponse, M. de Rijke, Y. Venema (Eds.), Modal Logic and Process Algebra, Vol. 53, CSLI Lecture Notes, Stanford, 1995, pp. 131–150.
[31] M. Hennessy, Axiomatising finite delay operators, Acta Inform. 21 (1984) 61–88.
[32] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall International, Engelwood Cliff, NJ, 1985.
[33] S.C. Kleene, Representation of events in nerve nets and finite automata, Automata Studies, Princeton University Press, Princeton NJ, 1956, pp. 3–41.
[34] R. Milner, A modal characterisation of observable machine-behaviour, in: E. Astesiano, C. Böhm (Eds.), Proc. 6th Colloquium on Trees in Algebra and Programming (CAAP '81), Genoa, LNCS 112, Springer, Berlin, 1981, pp. 25–34.
[35] R. Milner, Calculi for synchrony and asynchrony, Theoret. Comput. Sci. 25 (1983) 267–310.
[36] R. Milner, A complete inference system for a class of regular behaviours, J. Comput. System Sci. 28 (3) (1984) 439–466.
[37] R. Milner, Communication and Concurrency, Prentice-Hall International, Englewood Cliff, NJ, 1989.
[38] A. Ponse, Computable processes and bisimulation equivalence, Formal Aspects Comput. 8 (6) (1996) 648–678.
[39] P.M. Sewell, Nonaxiomatisability of equivalences over finite state processes, Ann. Pure Appl. Logic 90 (1/3) (1997) 163–191.