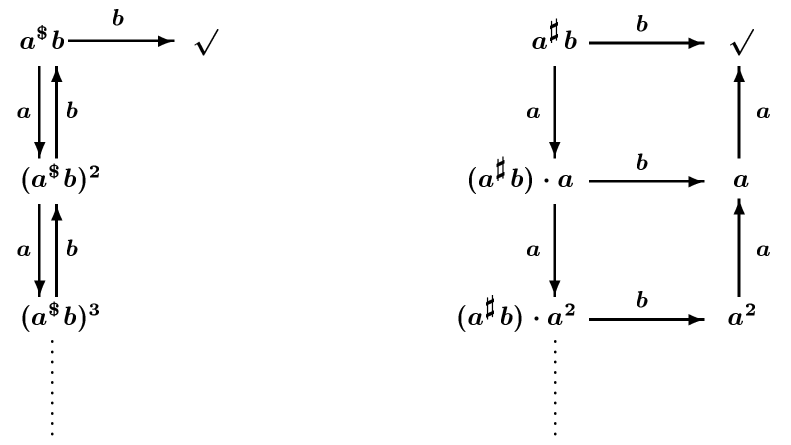
University of Amsterdam
Programming Research Group



Two Recursive Generalizations of Iteration in Process Algebra

Jan A. Bergstra
Alban Ponse

Report P9808 june 1998

University of Amsterdam

Department of Computer Science

Programming Research Group

Two Recursive Generalizations of Iteration in Process Algebra

Jan A. Bergstra
Alban Ponse

Report P9808 june 1998

J.A. Bergstra

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7591
e-mail: janb@wins.uva.nl

Department of Philosophy
Utrecht University

P.O.Box 80126
NL-3508 TC Utrecht
The Netherlands

tel. +31 30 253 5579
e-mail: Jan.Bergstra@phil.uu.nl

A. Ponse

Programming Research Group
Department of Computer Science
University of Amsterdam

Kruislaan 403
NL-1098 SJ Amsterdam
The Netherlands

tel. +31 20 525 7592
e-mail: alban@wins.uva.nl

Universiteit van Amsterdam, 1998

Two Recursive Generalizations of Iteration in Process Algebra

Jan A. Bergstra[1,2] Alban Ponse[1,3]

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

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

[3] *Corresponding author:* `alban@wins.uva.nl`

Abstract. The process algebraic framework ACP with abstraction is extended with two recursive generalizations of iteration: the *push-down* operation \$, defined by $x^{\$}y = x(x^{\$}y)(x^{\$}y) + y$, and the *nesting* operation $\sharp$, defined by $x^{\sharp}y = x(x^{\sharp}y)x + y$. With help of auxiliary actions, handshaking communication on these, and one of \$ or $\sharp$ we provide simple definitions of the following standard processes: *stack, context-free process, bag,* and *queue.* This supports the equational founding of process algebra: standard processes can be represented as terms.

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

1 Introduction

A 'classic' quotation of MILNER ([Mil83]) states that for a proper understanding of the basic issues concerning the behaviour 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."*

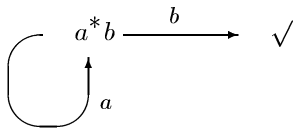
The addition of a single recursive operation to the process algebraic framework ACP (Algebra of Communicating Processes) with abstraction ([BK85, BW90]) provides general expressive power: each computable process can be expressed. We consider two candidates for such a recursive operation. The 'distinct and intuitive idea' embodied by each of these is a simple way of counting. Rather than focusing on a particular machine model for computability[1] we prove that standard processes as one finds in a text book on process theory ([Hoa85, Mil89, BW90]) can be specified in a straightforward way.

In [BBP94] we introduced *Binary Kleene star*, also called *iteration*, in process algebra. Binary Kleene star stems from KLEENE [Kle56] and is—in our notation—defined by

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

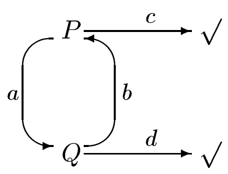
[1] In our setting, register machine programming would be a natural candidate. In Section 7 (Conclusions) we return to this issue.

In ACP, product models sequential composition, and + models choice. As in regular algebra, the symbol $\cdot$ is often omitted, and + binds weakest. So, x^*y expresses "either do x and repeat x^*y, or terminate with 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*: given a set A of actions, a process is *regular over* A if it is equivalent with 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 left-linear, regular grammar). Some examples with actions a, b, c and d:



and

$$P = a \cdot Q + c \qquad Q = b \cdot P + d$$



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

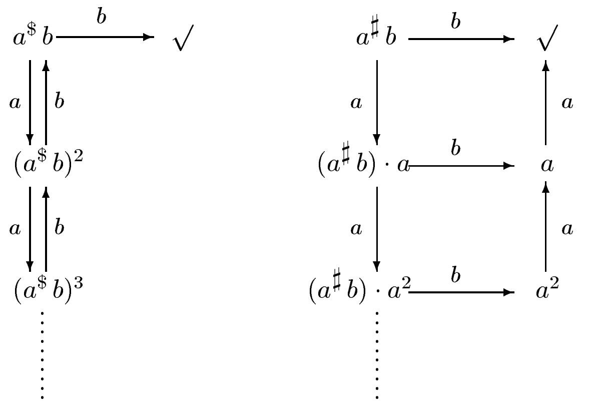
The focus on iteration in process algebra raised interest in variations of the Kleene star operation (cf. [BBP93, BBP94, Fok94, AF+96, AF97, Fok97]). Here we introduce the binary *push-down* operation $\_^{\$}\_$ as a generalization of iteration. This operation is defined by

$$x^{\$}y = x \cdot (x^{\$}y) \cdot (x^{\$}y) + y. \quad \text{(Either do } x \text{ and repeat } (x^{\$}y)(x^{\$}y)\text{, or terminate with } y.)$$

A second recursive operation that generalizes iteration, introduced in [BBP94], is the binary *nesting* operation $\_^{\sharp}\_$, defined by

$$x^{\sharp}y = x \cdot (x^{\sharp}y) \cdot x + y. \qquad \text{(Either do } x \text{ and repeat } (x^{\sharp}y)x\text{, or terminate with } y.)$$

Both \$ and $\sharp$ are 'non-regular': with each of these a non-regular process can be defined. As an example, consider processes $a^{\$}b$ and $a^{\sharp}b$ with actions a, b as illustrated below:



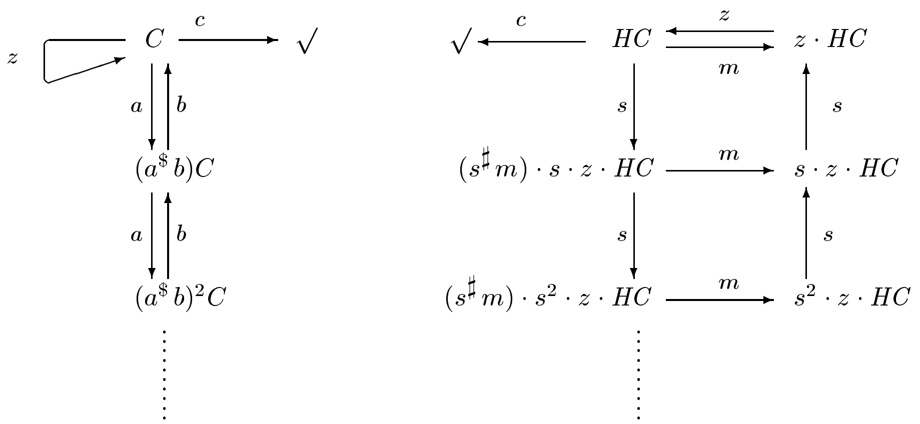
where the states that are process terms determine possible further behaviour, i.e., outgoing transitions. None of $a^{\$}b$, $a^{\sharp}b$ is regular in any common semantics.

In this paper we show by direct, algebraic proofs that the following standard processes can be defined with either \$ or $\sharp$, the operations from ACP with abstraction, auxiliary actions, and handshaking communication between these actions:

- A *stack* over a finite data type,
- A *context-free* process (roughly: a process specifiable by a context-free grammar),
- 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, comprising branching bisimulation and observation equivalence (for an overview and references see [BW90]).

Two basic, auxiliary processes used in our proofs are the *counter* C and the *half-counter* HC, both displayed below:



Using Kleene star and push-down, the counter C can be defined by $C = (a(a^{\$}\,b) + z)^*c$ with actions a (add 'one'), b (subtract 'one'), z (test 'zero'), and c ('clear', remove the counter). 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 $HC = ((s^{\sharp}\,m)z)^*c$. Initially HC is in 'add-mode', from which it can terminate by executing c, or in which it can stay by performing s-actions ('steps'). Furthermore, HC can evolve from add-mode into 'subtract-mode' by performing m. In subtract-mode, the half-counter can only "count back" to zero by performing s-actions after which it can re-enter its initial state by performing a zero-test action z. We provide a definition of HC using $\sharp$ as only recursive operation.

Our proof strategy is as follows: first we show (in Section 3) 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 4 a stack, which is a particular context-free process that generalizes a counter. Then, in Section 5, we show how an arbitrary context-free process can be defined with help of a regular control process and stacks. In Section 6 we provide simple specifications of a bag (multiset) and a queue, processes that both are not context-free.

Acknowledgement. We thank Bas Luttik (CWI) for comments.

2 Process Algebra, Axioms and Notation

We shortly recall some process algebra. The process algebraic framework $\mathsf{ACP}(A,\gamma)$ (Algebra of Communicating Processes, see, e.g, [BK84b, BW90]) has two parameters: a set A of constants modeling atomic actions, and a (partial) binary, commutative and associative *communication function* γ on A, defining which actions communicate. Furthermore there is a constant δ (deadlock or inaction). Process operations of $\mathsf{ACP}(A,\gamma)$ are alternative composition or choice $(+)$, sequential composition $(\cdot)$, parallel composition or merge $(\|)$, left and communication merge ($\mathbin{\|\!\_}$ and $|$, used for the axiomatization of $\|$), and encapsulation (∂_H, renaming actions in $H \subseteq A$ into δ). We mostly suppress the $\cdot$ in process expressions, and brackets according to the following precedences: $\cdot > \{\|, \mathbin{\|\!\_}, |\} > +$.

$\mathsf{ACP}(A,\gamma)$ is further extended to $\mathsf{ACP}^\tau(A,\gamma)$ (ACP with branching bisimulation) by adding the constant τ (silent step) and hiding τ_I (renaming actions in $I \subseteq A$ into τ). Process expressions are subject to the axioms of $\mathsf{ACP}^\tau(A,\gamma)$, displayed in Table 1 ($x, y, z, \ldots$ ranging over processes). Note that $+$ and $\cdot$ are associative, and that $+$ also is commutative and idempotent. We further assume commutativity and associativity of $\|$ and $|$, also known as SC (Standard Concurrency [BT84]). In this paper we only consider two-party communication or *handshaking*, axiomatized by $x \mid y \mid z = \delta$ (see [BT84]). For a detailed introduction to $\mathsf{ACP}(A,\gamma)$, $\mathsf{ACP}^\tau(A,\gamma)$ and SC we refer to [BW90].

Axioms for the $*$-operation are included in Table 1. In [FZ94], Fokkink and Zantema prove that (A1)–(A5) and (BKS1)–(BKS3) axiomatize strong bisimilarity for processes defined with $+, \cdot$ and $*$. We write $\mathsf{ACP}^*(A,\gamma)$ and $\mathsf{ACP}^{\tau *}(A,\gamma)$ for the extension with $*$ and relevant axioms.

Let for $\diamond \in \{\$, \sharp\}$, $\mathsf{ACP}^{\tau\diamond}(A,\gamma)$ 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

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

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

$$r_i \mid s_i = t$$

for each pair of actions r_i, s_i. Furthermore, we will often use Σ-notation for finite sums: the expression

$$\big(\textstyle\sum_{j=1}^n p_j\big)$$

abbreviates $p_1 + p_2 + \ldots + p_n$. 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 prove that for each regular process P there is a *finite* extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\diamond}(A^{ext},\gamma)$ with $\diamond \in \{\$, \sharp\}$. Also, we provide a definition of a counter in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ and of a half-counter in $\mathsf{ACP}^{\tau\sharp}(A^{ext},\gamma)$ for suitable, finite extension A^{ext}. We first give a more precise definition of regularity and introduce the rule RSP, which we use to prove our expressivity results.

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

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

Let $i=1,...,n$ and $n\geq 1$. Process P_i is *regular over* $A_\delta = A\cup\{\delta\}$ if P_i is a solution for X_i from a finite, linear specification of the form

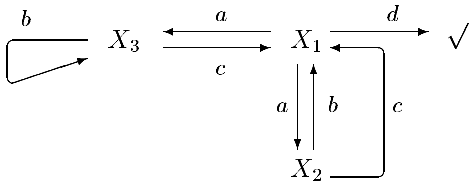
$$X_i = \left(\textstyle\sum_{j=1}^n \alpha_{i,j}\cdot X_j\right) + \beta_i, \tag{1}$$

where $\alpha_{i,j}$ and β_i are finite sums of actions or δ. E.g., a^*b is regular because it satisfies $X_1 = aX_1 + b$, and the same holds for $a(a^*b)+b$. Sometimes, for ease of specification, we shall allow $\alpha_{i,j}$ and β_i in (1) to be expressions built from A with $+$, and also $\cdot$ and $*$. This still defines a regular process: such a system can always be unfolded into a linear one. For example

$$X = (a(b+c)+a(b^*c))X + d,$$

with actions a,b,c,d can be unfolded into

$$\begin{aligned} X_1 &= aX_2 + aX_3 + d \\ X_2 &= (b+c)X_1 \\ X_3 &= cX_1 + bX_3. \end{aligned}$$



Now, process P is a solution for X if and only if it is a solution for X_1. With axioms (A6) and (A7), the required form for the X_i equations—involving δ-summands—can easily be obtained:

$$\begin{array}{lcl} X_1 & = & \delta X_1 + aX_2 + aX_3 + d \\ X_2 & = & (b+c)X_1 + \delta X_2 + \delta X_3 + \delta \\ X_3 & = & cX_1 + \delta X_2 + bX_3 + \delta. \end{array}$$

Each solution for the X_i in (1) above determines a *unique* process: the 'next steps' and resulting states are uniquely determined by $\alpha_{i,j}$ and β_i. This also holds in case a linear specification is *infinite*, i.e., parameterized over (some encoding of) $\mathbb{N}$. For instance (for $n \in \mathbb{N}$),

$$\begin{array}{lcl} X_1(0) & = & aX_1(1) + b \\ X_1(n+1) & = & aX_1(n+2) + bX_2(n) \\ X_2(0) & = & a \\ X_2(n+1) & = & aX_2(n). \end{array}$$

Then $a^\sharp b$ is a solution for $X_1(0)$, $(a^\sharp b)a^{n+1}$ for $X_1(n+1)$, and a^{n+1} for $X_2(n)$. Because a (finite or infinite) linear specification specifies a unique process, it is called *guarded*. The conditional rule RSP (*Recursive Specification Principle*, see, e.g. [BW90]) states that each guarded recursive specification has at most one solution per variable. For example, RSP implies that

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

as both are a solution for X_1 in $X_1 = aX_2,\ X_2 = aX_1$: take $a^*\delta$ and $(aa)^*\delta$ as solutions for X_2, and derive

$$\begin{array}{lcll} a^*\delta & = & \boldsymbol{a}\ (a^*\delta) & \text{(so } a^*\delta \text{ solves } X_1 \text{ and } X_2\text{),} \\ a((aa)^*\delta) & = & \boldsymbol{a}\ ((aa)^*\delta) & \text{(so } a((aa)^*\delta) \text{ solves } X_1 \\ (aa)^*\delta & = & \boldsymbol{a}\ (a((aa)^*\delta)) & \quad\text{and } (aa)^*\delta \text{ solves } X_2\text{).} \end{array}$$

Theorem 3.1. *For each regular process P over A_δ there is a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ with handshaking only, and the actions in A not subject to communication.*

Proof. Let the regular process P_1 be given by $P_i = \left(\sum_{j=1}^n \alpha_{i,j}\cdot P_j\right) + \beta_i$ $(i = 1,...,n)$ 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:

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

and consider the following processes:

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

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

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

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

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

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

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

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

Theorem 3.2. *For each regular process P over A_δ there is a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\sharp}(A^{ext}, \gamma)$ with handshaking only, and the actions in A not subject to communication.*

Proof. Let the regular process P_1 be given by $P_i = \left(\sum_{j=1}^{n} \alpha_{i,j} \cdot P_j\right) + \beta_i$ ($i = 1, ..., n$) 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:

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

where the s_{n+1}, r_{n+1} communications are used to remove remnants of $^\sharp$-recursion. Consider the following processes:

$\left(\sum_{j=1}^{n} \alpha_{i,j} \cdot s_j\right) + \beta_i \cdot s_0$	abbreviated by	G_i for $i = 1, ..., n$
$\left(\left(\sum_{j=1}^{n} r_j \cdot G_j\right) + s_{n+1} \cdot r_{n+1}\right)^\sharp r_0$	abbreviated by	M
$\left(\sum_{j=1}^{n+1} r_j \cdot s_j\right)^\sharp (r_0 \cdot s_0)$	abbreviated by	N.

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

Theorem 3.3. 1. *A counter $(a(a^\$ b) + z)^* c$ can be defined in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ if $|A^{ext} \setminus A| = 5$.*
2. *With seven additional actions, a half-counter $((s^\sharp m)z)^* c$ can be specified in $\mathsf{ACP}^{\tau\sharp}(A^{ext}, \gamma)$.*

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

$(a(a^\$ b) + z)s_1 + c$	abbreviated by	P
$(r_1 \cdot P)^\$ r_0$	abbreviated by	Q
$(r_1 \cdot s_1)^\$ s_0$	abbreviated by	R.

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

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

$$\begin{array}{rll} (s^\sharp m)z \cdot s_1 + c \cdot s_0 & \text{abbreviated by} & S \\ (r_1 \cdot S + s_2 \cdot r_2)^\sharp r_0 & \text{abbreviated by} & T \\ (r_1 \cdot s_1 + r_2 \cdot s_2)^\sharp (r_0 \cdot s_0) & \text{abbreviated by} & U. \end{array}$$

With an application of RSP it follows that $((s^\sharp m)z)^* c = \tau_{\{t\}} \circ \partial_{\{r_i, s_i \mid i=0,1,2\}}(S \cdot T \parallel U)$. □

4 Expressing a Stack

We provide specifications of a *stack* over a finite data type in $\mathsf{ACP}^{\tau\diamond}(A^{ext}, \gamma)$, for $\diamond \in \{\$, \sharp\}$, with help of a regular control process and two (half-)counters. Let $D = \{d_1, ..., d_N\}$ for some $N \in \mathbb{N} \setminus \{0\}$ be a finite set of data elements, ranged over by d. Let furthermore D^* be the set of finite strings over D, ranged over by w, and let ϵ denote the empty string. The stack $S(\epsilon)$ over D with empty-testing and termination option is defined by the infinite linear specification

$$\begin{array}{rcl} S(\epsilon) & = & \left(\sum_{j=1}^{N} r(d_j) \cdot S(d_j)\right) + s(\mathit{empty}) \cdot S(\epsilon) + r(\mathit{stop}) \\ S(dw) & = & \left(\sum_{j=1}^{N} r(d_j) \cdot S(d_j dw)\right) + s(d) \cdot S(w). \end{array}$$

Here the contents of the stack is represented by the argument of S: $S(dw)$ is the stack that contains dw with d on top. Action $r(d_i)$ (receive d_i) models the push of d_i onto the stack, and action $s(d_i)$ (send d_i) represents deletion of d_i 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 $N = 1$ ($D = \{d_1\}$), the equations above specify a counter: the stack contents then models the counter value.

Theorem 4.1. *A stack over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ with handshaking only, A^{ext} a finite extension of A and the actions in A not subject to communication.*

Proof. Let a stack $S(\epsilon)$ be given as described above. Without loss of generality, assume $D = \{d_1, ..., d_N\}$ for some $N > 1$ (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 $\ulcorner . \urcorner : D^* \to \mathbb{N}$:

$$\begin{array}{rcl} \ulcorner \epsilon \urcorner & = & 0, \\ \ulcorner d_j w \urcorner & = & j + N \ulcorner w \urcorner. \end{array}$$

This encoding is a bijection with inverse $\mathit{decode} : \mathbb{N} \to D^*$ (let $\star$ denote concatenation of strings):

$$\mathit{decode}(n) = \begin{cases} \epsilon & \text{if } n = 0, \\ d_N \star \mathit{decode}(\frac{n-N}{N}) & \text{if } n \neq 0 \text{ and } n \; mod \; N = 0, \\ d_{(n \; mod \; N)} \star \mathit{decode}(\frac{n-(n \; mod \; N)}{N}) & \text{otherwise.} \end{cases}$$

So in case $N = 3$, e.g., $\ulcorner d_3 d_1 d_2 \urcorner = 24$, and $\mathit{decode}(32) = d_2 d_1 d_3 \in \{d_1, d_2, d_3\}^*$.

Next we define two counters that we use to specify $S(\epsilon)$ in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$:

$$C_j = (\overline{a}_j(\overline{a}_j{}^{\$}\overline{b}_j) + \overline{z}_j)^*\ \overline{c}_j \qquad (j=1,2),$$

with add-action $\overline{a}_j$, subtract-action $\overline{b}_j$, zero-testing $\overline{z}_j$, and stop-action $\overline{c}_j$ all in $A^{ext}\setminus A$. We shall use the following abbreviations ($n\in\mathbb{N}$):

$$\begin{array}{rcl} C_j(0) & = & C_j, \\ C_j(n+1) & = & (\overline{a}_j{}^{\$}\overline{b}_j)\cdot C_j(n). \end{array}$$

We further define a regular control process X_ϵ with actions $a_j, b_j, z_j, c_j \in A^{ext}\setminus A$ and those of the stack. In combination with the C_j, the process X_ϵ is used to define $S(\epsilon)$. Note that the coding discussed above does not occur explicitly in this specification.

$$X_\epsilon \quad = \quad \left(\textstyle\sum_{j=1}^N r(d_j)\cdot a_1^j\cdot X_j\right) + s(empty)\cdot X_\epsilon + r(stop)\cdot c_1\cdot c_2,$$

and for $k=1,...,N$:

$$\begin{array}{rcll} X_k & = & \left(\sum_{j=1}^N r(d_j)\cdot \mathrm{Push}_j\right) + s(d_k)\cdot \mathrm{Pop}_k & \\ \mathrm{Push}_k & = & \mathrm{Shift\,1\,to\,2}\cdot a_1^k\cdot N\,\mathrm{Shift\,2\,to\,1}\cdot X_k & \\ \mathrm{Pop}_k & = & b_1^k\cdot \frac{1}{N}\,\mathrm{Shift\,1\,to\,2}\cdot \mathrm{Test}_\epsilon & \\ \mathrm{Shift\,1\,to\,2} & = & (b_1\cdot a_2)^* z_1 & \text{(shift the contents of } C_1 \text{ to } C_2\text{)} \\ N\,\mathrm{Shift\,2\,to\,1} & = & (b_2\cdot a_1^N)^* z_2 & \text{(shift the } N\text{-fold of } C_2 \text{ to } C_1\text{)} \\ \frac{1}{N}\,\mathrm{Shift\,1\,to\,2} & = & (b_1^N\cdot a_2)^* z_1 & \text{(shift the number of } N\text{-folds of } C_1 \text{ to } C_2\text{)} \\ \mathrm{Test}_\epsilon & = & b_2\cdot a_1\cdot \mathrm{Test}_1 + z_2\cdot X_\epsilon & \text{(determine whether the stack is empty,} \\ \mathrm{Test}_1 & = & b_2\cdot a_1\cdot \mathrm{Test}_2 + z_2\cdot X_1 & \text{ or which } D\text{-element is on top)} \\ \mathrm{Test}_2 & = & b_2\cdot a_1\cdot \mathrm{Test}_3 + z_2\cdot X_2 & \\ \vdots & & & \\ \mathrm{Test}_N & = & b_2\cdot a_1\cdot \mathrm{Test}_1 + z_2\cdot X_N. & \end{array}$$

Let γ for $j=1,2$ be defined on $(A^{ext}\setminus A)^2$ by $\gamma(a_j,\overline{a}_j)=\gamma(b_j,\overline{b}_j)=\gamma(z_j,\overline{z}_j)=\gamma(c_j,\overline{c}_j)=t\in A^{ext}\setminus A$, and let $H=\{a_j,\overline{a}_j,b_j,\overline{b}_j,z_j,\overline{z}_j,c_j,\overline{c}_j\mid j=1,2\}$. We show that

$$\tau_{\{t\}}\circ\partial_H(X_\epsilon\parallel C_1(0)\parallel C_2(0))$$

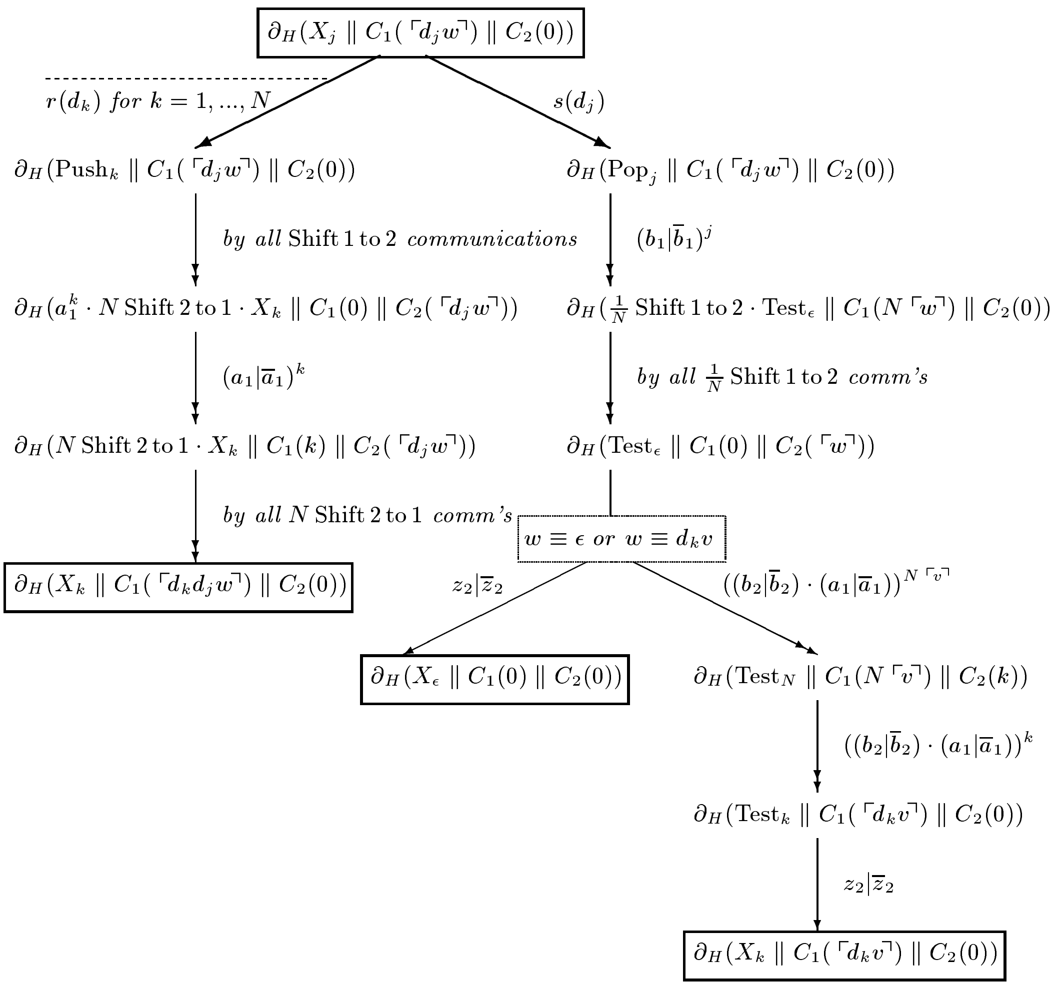
behaves as $S(\epsilon)$, the empty stack:

$$\begin{array}{rcl} \tau_{\{t\}}\circ\partial_H(X_\epsilon\parallel C_1(0)\parallel C_2(0)) & = & \left(\sum_{j=1}^N r(d_j)\cdot\tau_{\{t\}}\circ\partial_H(a_1^j\cdot X_j\parallel C_1(0)\parallel C_2(0))\right) \\ & & +\ s(empty)\cdot\tau_{\{t\}}\circ\partial_H(X_\epsilon\parallel C_1(0)\parallel C_2(0)) \\ & & +\ r(stop)\cdot\tau_{\{t\}}\circ\partial_H(c_1\cdot c_2\parallel C_1(0)\parallel C_2(0)) \\ & = & \left(\sum_{j=1}^N r(d_j)\cdot\tau^{j+1}\cdot\tau_{\{t\}}\circ\partial_H(X_j\parallel C_1(j)\parallel C_2(0))\right) \\ & & +\ s(empty)\cdot\tau_{\{t\}}\circ\partial_H(X_\epsilon\parallel C_1(0)\parallel C_2(0)) \\ & & +\ r(stop)\cdot\tau\cdot\tau \\ & = & \left(\sum_{j=1}^N r(d_j)\cdot\tau_{\{t\}}\circ\partial_H(X_j\parallel C_1(\ulcorner d_j\urcorner)\parallel C_2(0))\right) \qquad (2) \\ & & +\ s(empty)\cdot\tau_{\{t\}}\circ\partial_H(X_\epsilon\parallel C_1(0)\parallel C_2(0)) + r(stop). \end{array}$$

We are done if $\tau_{\{t\}} \circ \partial_H(X_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0))$ behaves as $S(d_j w)$, the stack with contents $d_j w$ for some $w \in D^*$. We prove this by first omitting the $\tau_{\{t\}}$-operation, and analyzing the behaviour of $\partial_H(X_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0))$. This analysis is arranged in a graphical style in Fig. 1, where $P \xrightarrow{a} Q$ represents the statement $P = a \cdot Q$ for some $a \in A$, $P \overset{w}{\twoheadrightarrow} Q$ represents $P = w \cdot Q$, and branching from expressions represents application of $+$. So the uppermost expression in Fig. 1 with its arrows and resulting expressions represents the obviously derivable equation

$$\begin{aligned}\partial_H(X_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0)) \quad = \quad & \big(\textstyle\sum_{k=1}^{N} r(d_k) \cdot \partial_H(\mathrm{Push}_k \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0))\big) \\ & + s(d_j) \cdot \partial_H(\mathrm{Pop}_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0)).\end{aligned}$$

Figure 1: Calculations with $\partial_H(X_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0))$.



By the axiom $x = x \cdot \tau$ (B1), identity (2) above, and the derivation displayed in Fig. 1 it follows that

$$\tau_{\{t\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)), \text{ and } \tau_{\{t\}} \circ \partial_H(X_j \parallel C_1(\ulcorner d_j w \urcorner) \parallel C_2(0))$$

satisfy the equations for $S(\epsilon)$ and $S(d_j w)$, respectively ($j = 1, ..., N$ and $w \in D^*$). By RSP it follows that

$$S(\epsilon) = \tau_{\{t\}} \circ \partial_H(X_\epsilon \parallel C_1 \parallel C_2).$$

By Theorems 3.1 and 3.3.1 it follows that once D is fixed, X_ϵ and hence the empty stack $S(\epsilon)$ can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ with handshaking for some $A^{ext} \supseteq A$. □

Theorem 4.2. *A stack over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\sharp}(A^{ext}, \gamma)$ with handshaking only, A^{ext} a finite extension of A and the actions in A not subject to communication.*

Proof. In [BBP94] it is proved that a stack over a finite data type can be defined in $\mathsf{ACP}^{\tau*\sharp}(A^{ext}, \gamma)$ (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 concerning regular control process) as above. So, it remains to be argued that all $*$-occurrences in the proof in [BBP94] can be avoided. For the regular control process this follows immediately from Theorem 3.2, and for a half-counter from Theorem 3.3.2. □

5 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

$$X_i = \left(\sum_{j=1}^{n} \left(\sum_{k=1}^{n} \alpha_{i,j,k} \cdot X_j \cdot X_k\right) + \beta_{i,j} \cdot X_j\right) + \gamma_i, \qquad i = 1, ..., n \text{ and } n \geq 1,$$

where $\alpha_{i,j,k}$, $\beta_{i,j}$ and γ_i are finite sums of actions or δ. (See [BBK93] for more information on rGNF.) In particular, rGNF specifications are guarded and RSP can be applied. As an example, $a^\sharp b$ is context-free as it is a solution for X_1 in $X_1 = aX_1X_2 + b$, $X_2 = a$. Also, a stack $S(\epsilon)$ over $\{d_1, ..., d_N\}$ is context-free: $S(\epsilon)$ is a solution of X_0 from the rGNF specification

$$\begin{array}{lcll} X_0 & = & \left(\sum_{j=1}^{N} r(d_j) X_j X_0\right) + s(\mathit{empty}) \cdot X_0 + r(\mathit{stop}) & \\ X_k & = & \left(\sum_{j=1}^{N} r(d_j) X_j X_k\right) + s(d_k) & (k = 1, ..., N). \end{array}$$

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

Theorem 5.1. *For each context-free process P over A_δ there is a finite extension A^{ext} of A such that P can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ ($\mathsf{ACP}^{\tau\sharp}(A^{ext}, \gamma)$) with handshaking only, and the actions in A not subject to communication.*

Proof. Let the context-free process P_1 be given by $P_i = \left(\sum_{j=1}^n \left(\sum_{k=1}^n \alpha_{i,j,k} \cdot P_j \cdot P_k\right) + \beta_{i,j} \cdot P_j\right) + \gamma_i$ $(i = 1, ..., n)$ where $\alpha_{i,j,k}$, $\beta_{i,j}$ and γ_i are finite sums of actions or δ. Consider the stack $S(\epsilon)$ over data-type $D = \{1, ..., n\}$ defined by

$$\begin{array}{rcl} S(\epsilon) & = & \left(\sum_{j=1}^n r_j \cdot S(j)\right) + r_0 \\ S(k\sigma) & = & \left(\sum_{j=1}^n r_j \cdot S(jk\sigma)\right) + s_k \cdot S(\sigma) \quad (k \in D,\ \sigma \in D^*) \end{array}$$

(thus empty-testing is absent), and let for $i = 1, ...n$,

$$\begin{array}{rcl} H_i & = & \left(\sum_{j=1}^n \left(\sum_{k=1}^n \alpha_{i,j,k} \cdot s_k \cdot s_j\right) + \beta_{i,j} \cdot s_j\right) + \gamma_i \\ P & = & \left(\sum_{j=1}^n r_j \cdot H_j\right)^* s_0. \end{array}$$

By Theorems 3.1 and 4.1 (3.2 and 4.2) there exists a finite extension A^{ext} of A such that P and $S(\epsilon)$ are specifiable in $\mathsf{ACP}^{\tau\$}(A^{ext}, \gamma)$ $(\mathsf{ACP}^{\tau\sharp}(A^{ext}, \gamma)$, respectively). With RSP it follows that $P_i = \tau_{\{t\}} \circ \partial_{\{r_i, s_i | i=0,...,n\}}(H_i \cdot P \parallel S(\epsilon))$. $\Box$

6 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 in $\mathsf{ACP}^{\tau\diamond}(A^{ext}, \gamma)$ for $\diamond \in \{\$, \sharp\}$. First we consider a bag or *multiset*.

Let again $D = \{d_1, ..., d_N\}$ for some $N \in \mathbb{N} \setminus \{0\}$ be a finite set of data elements, ranged over by d, and let ρ range over $\mathbb{N}^N$, where ρ_k denotes the k-th component of ρ for $k = 1, ..., N$. In this way, ρ encodes the bag with ρ_k occurrences of d_k. In order to give a straightforward specification of the bag over D, we define the function $\Uparrow: \mathbb{N}^N \times N \to \mathbb{N}^N$ in a component-wise fashion:

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

Likewise, the function $\Downarrow: \mathbb{N}^N \times N \to \mathbb{N}^N$ is defined as

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

Let furthermore $\lambda \in \mathbb{N}^N$ denote the sequence of N zero's. Then for $\rho \in \mathbb{N}^N \setminus \{\lambda\}$, the bag $B(\lambda)$ over D with empty-testing and termination option is defined by:

$$\begin{array}{rcl} B(\lambda) & = & \left(\sum_{j=1}^N r(d_j) \cdot B(\lambda \Uparrow j)\right) + s(empty) \cdot B(\lambda) + r(stop) \\ B(\rho) & = & \left(\sum_{j=1}^N r(d_j) \cdot B(\rho \Uparrow j)\right) + \left(\sum_{\{i | \rho_i > 0\}} s(d_i) \cdot B(\rho \Downarrow i)\right). \end{array}$$

Here the contents of the bag is represented by the B-argument: $B(\rho)$ is the bag that contains ρ_i occurrences of d_i. Action $r(d_i)$ (receive d_i) models insertion of d_i in the bag, and action

[2] A bag (queue) over more than one data element is not context-free, see [BK84a] ([BB88], respectively).

$s(d_i)$ (send d_i) models deletion of d_i 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 6.1. *A bag or multiset over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ or $\mathsf{ACP}^{\tau\sharp}(A^{ext},\gamma)$ with handshaking only, A^{ext} a finite extension of A and the actions in A not subject to communication.*

Proof. Let the bag $B(\lambda)$ and D 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:

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

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

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 $C_1, ..., C_N$, where C_j has add-action $\overline{a}_j$, subtract-action $\overline{b}_j$, zero-testing $\overline{z}_j$, and stop-action $\overline{c}_j$, and the regular control process X_λ defined by the following 2^N equations ($\sigma \in \{0,1\}^N \setminus \{\lambda\}$):

$$\begin{array}{rcl} X_\lambda & = & \left(\sum_{j=1}^N r(d_j) \cdot a_j \cdot X_{\lambda\uparrow j}\right) + s(\mathit{empty}) \cdot X_\lambda + r(\mathit{stop}) \cdot c_1 \cdot ... \cdot c_N \\ X_\sigma & = & \left(\sum_{j=1}^N r(d_j) \cdot a_j \cdot X_{\sigma\uparrow j}\right) + \left(\sum_{\{i \mid \sigma_i = 1\}} s(d_i) \cdot b_i \cdot (z_i \cdot X_{\sigma\downarrow i} + b_i \cdot a_i \cdot X_\sigma)\right). \end{array}$$

By Theorems 3.1 and 4.1 (3.2 and 4.2), X_λ and the C_j can be specified in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ ($\mathsf{ACP}^{\tau\sharp}(A^{ext},\gamma)$, respectively) for a suitable extension A^{ext}. With RSP it follows that

$$B(\lambda) \quad = \quad \tau_{\{t\}} \circ \partial_H(X_\lambda \parallel C_1 \parallel ... \parallel C_N),$$

where $H = \{x_j, \overline{x}_j \mid x \in \{a,b,z,c\},\ j = 1,...,N\}$ and $x_j \mid \overline{x}_j = t$ for $x \in \{a,b,z,c\},\ j = 1,...,N$. □

We finish this section with the specification of a queue over a finite data type $D = \{d_1, ..., d_N\}$ ($N > 0$) using only either \$ or $\sharp$. The queue $Q(\epsilon)$ over D with empty-testing and termination option is defined by the infinite linear specification

$$\begin{array}{rcl} Q(\epsilon) & = & \left(\sum_{j=1}^N r(d_j) \cdot Q(d_j)\right) + s(\mathit{empty}) \cdot Q(\epsilon) + r(\mathit{stop}) \\ Q(wd) & = & \left(\sum_{j=1}^N r(d_j) \cdot Q(d_j wd)\right) + s(d) \cdot Q(w). \end{array}$$

Here the contents of the queue is represented by the argument of Q: $Q(wd)$ is the queue that contains wd with d on top. Action $r(d)$ (receive d) models insertion of d in the queue, and action $s(d)$ (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 6.2. *A queue over a finite data type D with actions from A can be expressed in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ or $\mathsf{ACP}^{\tau\sharp}(A^{ext},\gamma)$ with handshaking only, A^{ext} a finite extension of A and the actions in A not subject to communication.*

Proof. Let the queue $Q(\epsilon)$ and D be defined as above. We use two stacks $S(\epsilon)$ and $T(\epsilon)$ with empty-testing and stop-facilities. A push-action of d_j onto S is modeled by the action $r_S(d_j)$, a pop-action by $s_S(d)$, an empty-test by $\overline{z}_S$, and a termination action by $\overline{c}_S$. In a similar fashion, using subscripts T instead of S, we define the actions of T.

The idea is to define regular processes X_ϵ and X_d such that $\tau_{\{t\}}\circ\partial_H(X_\epsilon \parallel S(\epsilon) \parallel T(\epsilon))$ models $Q(\epsilon)$, and $\tau_{\{t\}}\circ\partial_H(X_d \parallel S(wd) \parallel T(\epsilon))$ models $Q(wd)$, the queue with contents wd where d is on top. Deletion is modeled by shifting $S(wd)$ to $T(d\tilde{w})$ (where $\tilde{w}$ is the reverse of w), deleting d, and shifting back $T(\tilde{w})$ to $S(w)$. Consider the regular control process X_ϵ defined by the following $N+4$ equations ($d\in D$):

$$
\begin{aligned}
X_\epsilon &= \left(\textstyle\sum_{j=1}^N r(d_j)\cdot s_S(d_j)\cdot X_{d_j}\right) + s(empty)\cdot X_\epsilon + r(stop)\cdot c_S\cdot c_T\\
X_d &= \left(\textstyle\sum_{j=1}^N r(d_j)\cdot s_S(d_j)\cdot X_d\right) + s(d)\cdot \mathit{Shift}(S,T)\cdot r_T(d)\cdot \mathit{Next}\\
\mathit{Next} &= \left(\textstyle\sum_{j=1}^N r_T(d_j)\cdot s_S(d_j)\cdot \mathit{Shift}(T,S)\cdot X_{d_j}\right) + z_T\cdot X_\epsilon\\
\mathit{Shift}(S,T) &= \left(\textstyle\sum_{j=1}^N r_S(d_j)\cdot s_T(d_j)\right)^* z_S\\
\mathit{Shift}(T,S) &= \left(\textstyle\sum_{j=1}^N r_T(d_j)\cdot s_S(d_j)\right)^* z_T.
\end{aligned}
$$

By Theorems 3.1 and 4.1 (3.2 and 4.2), X_ϵ, $S(\epsilon)$ and $T(\epsilon)$ can be specified in $\mathsf{ACP}^{\tau\$}(A^{ext},\gamma)$ or in $\mathsf{ACP}^{\tau\sharp}(A^{ext},\gamma)$, respectively, for a suitable extension A^{ext}.

Now let $H=\{r_i(d), s_i(d), z_i, \overline{z}_i, c_i, \overline{c}_i \mid i\in\{S,T\},\ d\in D\}$, and $r_i(d)\mid s_i(d)=z_i \mid \overline{z}_i = c_i \mid \overline{c}_i = t$ for $i\in\{S,T\}$. Then it follows with RSP that $Q(\epsilon)=\tau_{\{t\}}\circ\partial_H(X_\epsilon \parallel S(\epsilon)\parallel T(\epsilon))$. □

7 Conclusions

In process algebra, a (potentially) infinite process is traditionally represented as a solution of a system of guarded recursive equations, and proof theory and verification often focus on reasoning about such recursive systems. Though specification (and verification) of concurrent processes defined in this way serves its purpose well, recursive *operations* give a more direct representation, and are easier to comprehend. We showed that an expressive format is obtained by adding one of $\$,\sharp$ as a primitive to ACP with abstraction. In fact, it is evident that each *computable process*—given by a total recursive function describing next steps and resulting states (cf. [BBK87, Pon96])—can be specified in $\mathsf{ACP}^{\tau\diamond}(A^{ext},\gamma)$ for $\diamond\in\{\$,\sharp\}$ and A^{ext} a finite extension of A: with regular processes and counters, register machine computation can be directly represented. Adding binary Kleene star as well yields a more flexible format for specification and verification of concurrent processes. These results support the equational founding of process algebra: any process of interest can simply be represented by a term. We note that our expressivity results do not rely upon *fairness rules*[3] : in none of the constructions the possibility of an infinite τ-trace occurs. Furthermore, the only abstraction law used is $x\tau = x$. Therefore, our results are preserved in observation equivalence, and even in a setting that distinguishes infinite τ-traces from finite ones, such as *divergence sensitive branching bisimulation*, defined in [GV95].

[3] Fairness models the assumption that in case of a a τ-loop eventually some other alternative—if available—is executed. An example is captured by the law $\tau(\tau^* x)=\tau x$ (see [BBP94]), valid in branching bisimilarity.

Finally, we give brief consideration to some proof theoretical issues. We notice that there exists a finite set A of actions with communication function γ such that

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

with $\diamond \in \{\$, \sharp\}$ is *undecidable* for closed 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 closed process terms t_n, u_n (parameterized by n) such that

$$\begin{array}{lcl} n \in V_1 & \Rightarrow & \mathsf{ACP}^{\tau\diamond}(A,\gamma) \vdash t_n = u_n, \\ n \in V_2 & \Rightarrow & t_n \not\cong u_n, \end{array}$$

where $\cong$ expresses branching bisimulation. 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 $\mathsf{ACP}^{\tau\sharp}(A,\gamma)$ and $\mathsf{ACP}^{\tau\$}(A,\gamma)$. Let $\mathsf{RSP}^{\$}$ be the conditional rule

$$x = yxx + z \quad\Longrightarrow\quad x = y^{\$}z.$$

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 $x^{\$}y = x(x^{\$}y)(x^{\$}y) + y$ and $\mathsf{RSP}^{\$}$ is complete with respect to strong bisimilarity over that signature. Transition rules for \$ are

$$\frac{x \xrightarrow{a} \surd}{\begin{array}{l} x^{\$}y \xrightarrow{a} (x^{\$}y)(x^{\$}y) \\ y^{\$}x \xrightarrow{a} \surd \end{array}} \qquad\qquad \frac{x \xrightarrow{a} x'}{\begin{array}{l} x^{\$}y \xrightarrow{a} x'(x^{\$}y)(x^{\$}y) \\ y^{\$}x \xrightarrow{a} x' \end{array}}$$

and the remaining transition rules and strong bisimulation equivalence are defined as usual (cf. [BW90]). Transition rules for $\sharp$ are defined in [BBP94], and a similar open question can be raised. Note that $\mathsf{RSP}^{\$}$ as such is not even sound if τ is involved. For example, $\tau = \tau\tau\tau + \delta$, though $\tau = \tau^{\$}\delta$ is not acceptable (τa can eventually perform a, whereas $(\tau^{\$}\delta)a$ cannot). We notice that the completeness result of Fokkink and Zantema [FZ94] implies that $\mathsf{BPA}(A)$ with Kleene's axiom $x^*y = x(x^*y) + y$ 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) (see Table 1):

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

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

References

[AF97] L. Aceto and W.J. Fokkink. An equational axiomatization for multi-exit iteration, *Information and Computation*, 137:121–158, 1997.

[AF+96] L. Aceto, W.J. Fokkink, R.J. van Glabbeek, and A. Ingolfsdottir. Axiomatizing prefix iteration with silent steps. *Information and Computation*, 127:26–40, 1996.

[BB88] J.C.M. Baeten and J.A. Bergstra. Global renaming operators in concrete process algebra. *Information and Computation*, 78(3):205–245, 1988.

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

[BBK93] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653 – 682, 1993.

[BBP93] J.A. Bergstra I. Bethke, and A. Ponse. Process algebra with iteration. Technical Report P9314, Programming Research Group, University of Amsterdam, 1993.
(See also `http://www.wins.uva.nl/research/prog/reports/reports.html`.)

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

[BK84a] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proc. 11^{th} ICALP*, pages 82–95. Volume 172 of *Lecture Notes in Computer Science*, Springer-Verlag, 1984. An extended version appeared in A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes, Utrecht 1994*, pages 1–25. Workshops in Computing, Springer-Verlag, 1995.

[BK84b] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109-137, 1984.

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

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

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

[Fok94] W.J. Fokkink. A complete equational axiomatization for prefix iteration. *Information Processing Letters*, 52:333–337, 1994.

[Fok97] W.J. Fokkink. Axiomatizations for the perpetual loop in process algebra. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24^{th} ICALP*, pages 571–581. Volume 1256 of *Lecture Notes in Computer Science*, Springer-Verlag, 1997.

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

[GV95] J.F. Groote and S.F.M. van Vlijmen. A modal logic for μCRL. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra*, pages 131–150. Volume 53 of *CSLI Lecture Notes*, Stanford, 1995.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

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

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

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

[Pon96] A. Ponse. Computable processes and bisimulation equivalence. *Formal Aspects of Computing*, 8(6):648–678, 1996.

Technical Reports of the Programming Research Group

Note: These reports can be obtained using the technical reports overview on our WWW site (`http://www.wins.uva.nl/research/prog/reports/`) or by anonymous ftp to `ftp.wins.uva.nl`, directory `pub/programming-research/reports/`.

[P9808] J.A. Bergstra and A. Ponse. *Two Recursive Generalizations of Iteration in Process Algebra.*

[P9807] M.P.A. Sellink and C. Verhoef. *An Architecture for Software Maintenance.*

[P9806] M. B. van der Zwaag. *Some Verifications in Process Algebra with Iota.*

[P9805] M.P.A. Sellink and C. Verhoef. *Development, Assessment, and Reengineering of Language Descriptions.*

[P9804] M.P.A. Sellink and C. Verhoef. *Native Patterns.*

[P9803] J.J. Brunekreef. *Annotated Algebraic Specification of the Syntax and Semantics of the Programming Language* Alma-0.

[P9802] W. Fokkink and C. Verhoef. *Conservative Extension in Positive/Negative Conditional Term Rewriting with Applications to Software Renovation Factories.*

[P9801] P. Klint and C. Verhoef. *Evolutionary Software Engineering: A Component-based Approach.*

[P9726] J.A. Bergstra and A. Ponse. *Grid Protocol Specifications.*

[P9725] J.A. Bergstra and A. Ponse. *Process Algebra Primitives for File Transfer.*

[P9724] J.A. Bergstra and A. Ponse. *Process Algebra with Four-Valued Logic.*

[P9723] J.A. Bergstra and A. Ponse. *Kleene's Three-Valued Logic and Process Algebra.*

[P9722] J.A. Bergstra and A. Ponse. *Bochvar-McCarthy Logic and Process Algebra.*

[P9721] M.P.A. Sellink and C. Verhoef. *Reflections on the Evolution of COBOL.*

[P9720] M. van der Graaf. *A Specification of Box to HTML in ASF+SDF.*

[P9719] M. van den Brand, M.P.A. Sellink, and C. Verhoef. *Current Parsing Techniques in Software Renovation Considered Harmful.*

[P9718] M. de Jonge. *Reuse of ASF+SDF Specifications by means of Renaming.*

[P9717] A. Sellink (Ed.). *Participants Proceedings of the 2nd International Workshop on the Theory and Practice of Algebraic Specifications.*

[P9717] E. Visser. *From Context-free Grammars with Priorities to Character Class Grammars.*

[P9716] H.M. Sneed. *Dealing with the Dual Crisis — Year 2000 and Euro — What Reverse Engineering can do to Help.*

[P9715] W. Fokkink and C. Verhoef. *An SOS Message: Conservative Extension in Higher Order Positive/Negative Conditional Term Rewriting.*

[P9714] M. van den Brand, M.P.A. Sellink, and C. Verhoef. *Control Flow Normalization for COBOL/CICS Legacy Systems.*

[P9713] B. Diertens. *Simulation and Animation of Process Algebra Specifications.*

[P9712] V. Partington. *Implementation of an Imperative Programming Language with Backtracking.*

[P9711] L. Moonen. *A Generic Architecture for Data Flow Analysis to Support Reverse Engineering.*

[P9710] B. Luttik and E. Visser. *Specification ofRewriting Strategies.*

[P9709] J.A. Bergstra and M.P.A. Sellink. *An Arithmetical Module for Rationals and Reals.*

[P9708] E. Visser. *Character Classes.*

[P9707] E. Visser. *Scannerless Generalized-LR Parsing.*

[P9706] E. Visser. *A Family of Syntax Definition Formalisms.*

[P9705] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Generation of Components for Software Renovation Factories from Context-free Grammars.*

[P9704] P.A. Olivier. *Debugging Distributed Applications Using a Coordination Architecture.*

[P9703] H.P. Korver and M.P.A. Sellink. *A Formal Axiomatization for Alphabet Reasoning with Parametrized Processes.*

[P9702] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef. *Reengineering COBOL Software Implies Specification of the Underlying Dialects.*

[P9701] E. Visser. *Polymorphic Syntax Definition.*

[P9618] M.G.J. van den Brand, P. Klint, and C. verhoef. *Re-engineering needs Generic Programming Language Technology.*

[P9617] P.I. Manuel. *ANSI Cobol III in SDF + an ASF Definition of a Y2K Tool.*

[P9616] P.H. Rodenburg. *A Complete System of Four-valued Logic.*

[P9615] S.P. Luttik and P.H. Rodenburg. *Transformations of Reduction Systems.*

[P9614] M.G.J. van den Brand, P. Klint, and C. Verhoef. *Core Technologies for System Renovation.*

[P9613] L. Moonen. *Data Flow Analysis for Reverse Engineering.*

[P9612] J.A. Hillebrand. *Transforming an ASF+SDF Specification into a ToolBus Application.*

[P9611] M.P.A. Sellink. *On the conservativity of Leibniz Equality.*

[P9610] T.B. Dinesh and S.M. Üsküdarlı. *Specifying input and output of visual languages.*