Process Algebra with Recursive Operations

Jan Bergstra[2,3] Wan Fokkink[1] Alban Ponse[1,3]

[1] *CWI*
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
url: http://www.cwi.nl/

[2] *Utrecht University, Department of Philosophy*
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
url: http://www.phil.uu.nl/eng/home.html

[3] *University of Amsterdam, Programming Research Group*
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
url: http://www.wins.uva.nl/research/prog/

Contents

1 Introduction **2**

2 Preliminaries: Axioms and Operational Semantics **6**

2.1 ACP-Based Systems 6

2.2 Transition Rules and Operational Semantics 8

2.3 Recursive Specifications 13

3 Axiomatisation of the Binary Kleene Star **15**

3.1 Preliminaries 15

3.2 Completeness 16

3.3 Irredundancy of the Axioms 24

3.4 Negative Results 25

3.5 Extensions of $\mathrm{BPA}^*(A)$ 27

4 Axiomatisations of Other Iterative Operations **28**

4.1 Axiomatisation of No-Exit Iteration 28

4.2 Axiomatisation of Multi-Exit Iteration 30

4.3 Axiomatisation of Prefix Iteration 32

4.4 Axiomatisation of String Iteration 34

4.5 Axiomatisation of Flat Iteration 35

5 Expressivity Results **37**

5.1 Expressivity of the Binary Kleene Star 37

5.2 Expressivity of Multi-Exit Iteration 41

5.3 Expressivity of String Iteration 43

5.4 Expressivity of Flat Iteration 44

6 Non-Regular Recursive Operations **46**

6.1 Process Algebra with a Push-Down Operation 48

6.2 Expressing a Stack 50

6.3 Undecidability Results 54

7 Special Constants **55**

7.1 Silent Step and Fairness 55

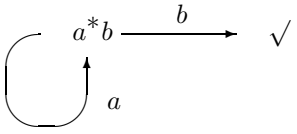
7.2 Empty Process 56

1 Introduction

In process algebra, a (potentially) infinite process is usually represented as the solution of 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 purposes well, recursive *operations* give a more direct representation and are easier to comprehend. The *Kleene star* $*$ can be considered as the most fundamental recursive operation. In process algebra, the defining equation for the binary Kleene star reads

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

where $\cdot$ models sequential composition and $+$ models non-deterministic choice ($\cdot$ binds stronger than $+$). In terms of operational semantics, the process x^*y chooses between x and y, and upon termination of x has this choice again. For example, the expression a^*b for atomic actions a and b can be depicted by



where $\surd$ expresses successful termination. This chapter discusses the Kleene star in the setting of process algebra, and considers some derived recursive operations, with a focus on axiomatisations and expressiveness hierarchies.

In the summer of 1951, S.C. Kleene was supported by the RAND Corporation, leading to Research Memorandum "Representation of Events in Nerve Nets and

Finite Automata" [53]. The material in that paper, based on the fundamental paper [65][1], was republished under the same title five years later [54]. In this seminal work, KLEENE introduced the binary operation $*$ for describing 'regular events'. He defined *regular expressions*, which correspond to finite automata, and gave algebraic transformation rules for these, notably

$$E^*F = E(E^*F) \vee F$$

(E^*F being the *iterate* of E on F). KLEENE noted the correspondence with conventions of algebra, treating $E \vee F$ as the analogue of $E + F$, and EF as the product of E and F.

In 1958, COPI, ELGOT, and WRIGHT [34] showed interest in the results from [54]. However, they judged KLEENE's theorems on analysis[2] and synthesis[3] obscured both by the complexity of his basic concepts and by the nature of the elements used in his nets. They introduced simpler and stronger nets (in a sense weakening KLEENE's synthesis result, but stating that this "brings the essential nature of the result into sharper focus"), and simpler operations. In particular they introduced a unary $*$ operation

> "[...] because the operation Kleene uses seems "essentially" singulary and because the singulary operation simplifies the algebra of regular events. It should be noted that the singulary and binary star operations are interdefinable." [34, page 195]

This contradicts KLEENE's original argument in [53, page 50] that the length of an event is at least one, and that for this reason he did not define E^* as a unary operation.

Four years later, REDKO [69] proved that there does not exist a sound and complete finite *equational* axiomatisation for regular expressions. (This proof was simplified and corrected by PILLING; see [35, Chapter 11].) In 1966, SALOMAA [70] presented a sound and complete finite axiomatisation for regular expressions, with as basic ingredient an *implicational* axiom dating back to ARDEN [9], namely (in process algebra notation):

$$x = (y \cdot x) + z \quad \Longrightarrow \quad x = y^*z$$

if y does not have the so-called *empty word property*. According to KOZEN [56], this last property is not algebraic, in the sense that it is not preserved under action refinement; he proposed two alternative implicational axioms that do not have this

[1] Kleene judged the theory for nerve nets with circles in [65] (McCulloch and Pitts, 1943) to be obscure, and proceeded independently. He came up with "regular events" and the major correspondence results in automata theory.

[2] Theorem 5, stating that finite automata model regular events.

[3] Theorem 3, stating that each regular event can be described by a finite automaton ("nerve net").

drawback. KROB [57] settled two conjectures by CONWAY [35], to obtain an *infinite* sound and complete equational axiomatisation for regular expressions. BLOOM and ÉSIK [14] developed an alternative infinite equational axiomatisation for regular expressions, within the framework of iteration theories.

In 1984, MILNER [61] was the first to consider the unary Kleene star in process algebra, modulo *strong bisimulation equivalence* [66]. In contrast with regular expressions, this setting is not sufficiently expressive to describe all *regular* (i.e., finite-state) processes. Moreover, the *merge* operator $x \parallel y$ [59] that executes its two arguments in parallel, and which is fundamental to process algebra, cannot always be eliminated from process terms in the presence of the Kleene star. MILNER presented an axiomatisation for the unary Kleene star in strong bisimulation semantics, being a subset of SALOMAA's axiomatisation, and asked whether his axiomatisation is complete. FOKKINK [40] solved this question for *no-exit iteration* $x^*\delta$, where the special constant δ, called *deadlock*, does not exhibit any behaviour; since δ blocks the exit, $x^*\delta$ executes x infinitely often. BLOOM, ÉSIK, and TAUBNER [15] presented a complete axiomatisation for regular synchronisation trees modulo strong bisimulation equivalence, within the framework of iteration theories. MILNER also asked for a characterisation of those recursive specifications that can be described modulo strong bisimulation semantics in process algebra with the unary Kleene star. BOSSCHER [24] solved this question in the absence of the deadlock. The two questions by MILNER in their full generality remain unsolved.

The unary Kleene star naturally gives rise to the *empty process* [74], which does not combine well with the merge operator. Therefore, BERGSTRA, BETHKE, and PONSE [12, 13] returned in 1993 to the binary version x^*y of the Kleene star in process algebra, naming the operator after its discoverer. TROEGER [73] introduced a process specification language with iteration, in which he introduced a striking axiom for the binary Kleene star, here presented in process algebra notation:

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

FOKKINK and ZANTEMA [39, 44] gave an affirmative answer to an open question from [13], namely, that the defining axiom and TROEGER's axiom for the binary Kleene star, together with

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

and the five standard axioms on $+$ and $\cdot$ for *basic process algebra* [18], provide an equational characterisation of strong bisimulation equivalence.

CORRADINI, DE NICOLA, and LABELLA studied the unary Kleene star in the presence of deadlock modulo *resource bisimulation equivalence*. In [31, 32] they presented a complete axiomatisation based on KOZEN's conditional axioms. In [33] they came up with a complete equational axiomatisation including TROEGER's axiom.

ACETO, FOKKINK, and INGÓLFSDÓTTIR [4] proved that a whole range of process semantics coarser than strong bisimulation do not allow a finite equational characterisation of the binary Kleene star. Furthermore, SEWELL [72] showed that there

does not exist a finite equational characterisation of the binary Kleene star modulo strong bisimulation equivalence in the presence of the deadlock δ, due to the fact that $(a^k)^*\delta$ is strongly bisimilar to $a^*\delta$ for positive integers k.

Several variations of the binary Kleene star were introduced, to obtain particular desirable properties.

- In order to increase the expressive power of the binary Kleene star in strong bisimulation semantics, Bergstra, Bethke, and Ponse [12] proposed *multi-exit iteration* $(x_1, \ldots, x_k)^*(y_1, \ldots, y_k)$ for positive integers k, with as defining equation

$$(x_1, \ldots, x_k)^*(y_1, \ldots, y_k) = x_1 \cdot ((x_2, \ldots, x_k, x_1)^*(y_2, \ldots, y_k, y_1)) + y_1.$$

  Aceto and Fokkink [1] presented an axiomatic characterisation of multi-exit iteration in basic process algebra, modulo strong bisimulation equivalence.

- *Prefix iteration* (similar to the *delay* operation from Hennessy [51]) is obtained by restricting the left-hand side of the binary Kleene star to atomic actions. Aceto, Fokkink, and Ingólfsdóttir [5, 37] presented finite equational characterisations of prefix iteration in *basic CCS* [59], modulo a whole range of process semantics.

- *String iteration* [7] is obtained by restricting the left-hand side of the binary Kleene star to non-empty finite strings of atomic actions. Aceto and Groote [7] presented an equational characterisation of string iteration in basic CCS, modulo strong bisimulation equivalence.

- Bergstra, Bethke, and Ponse [12] introduced *flat iteration*, which is obtained by restricting the left-hand side of the binary Kleene star to sums of atomic actions. Unlike the binary Kleene star, the merge operator can be eliminated from process terms in the presence of flat iteration. In [48], van Glabbeek presented a complete finite equational characterisation of flat iteration in basic CCS extended with the *silent step* τ [59], modulo four rooted weak bisimulation semantics that take into account the silent nature of the special constant τ (identifying $x\tau$ and x).

This chapter also concerns expressivity of (subsystems of) the *algebra of communicating processes* (ACP) [18] extended with the constant τ and *abstraction* operators [19], and enriched with the binary Kleene star or variants thereof. The τ, which represents a silent step, in combination with abstraction operators, which rename actions into τ, enables one to abstract from internal behaviour. In weaker semantics that identify $x\tau$ and x, each *regular* process can be specified in ACP_τ (i.e., ACP with abstraction) and the binary Kleene star, using only *handshaking* (i.e., two-party communication) [21] and some auxiliary actions. Furthermore, in such a semantics, each *computable* process can be specified in ACP_τ with abstraction and a

single recursive, non-regular operation, using only handshaking and some auxiliary actions.

This chapter is set up as follows. Section 2 introduces the preliminaries. Section 3 contains an exposition on axiomatisations for the binary Kleene star, while Section 4 discusses axiomatisations for related iterative operations. Section 5 compares the expressivity of these iterative operations, and Section 6 studies the expressivity of some non-regular recursive operations. Finally, Section 7 touches upon topics such as fairness and the empty process.

Acknowledgements. We thank Faron Moller for providing useful comments.

2 Preliminaries: Axioms and Operational Semantics

In this section we recall various process algebra axiom systems, structural operational semantics, behavioural equivalences, and recursive specifications. For a detailed introduction to these matters see, e.g., Baeten and Weijland [28].

2.1 ACP-Based Systems

Let A be a finite set of *atomic actions* $a, b, c, \ldots$, and let δ be a constant not in A. We write A_δ for $A \cup \{\delta\}$. Let furthermore $\_ \mid \_ : A_\delta \times A_\delta \to A_\delta$ be a *communication function* that is commutative,

$$a \mid b = b \mid a \text{ for all } a, b \in A_\delta,$$

associative,

$$a \mid (b \mid c) = (a \mid b) \mid c \text{ for all } a, b, c \in A_\delta,$$

and that satisfies $\delta \mid a = \delta$ for all $a \in A_\delta$. The communication function $\mid$ will be used to define *communication actions*: in the case that $a \mid b = c \in A$, the simultaneous execution of actions a and b results in communication action c. The actions in A and the communication function $\mid$ can be regarded as parameters of the process algebra axiom systems that are presented below.

The process algebraic framework $\mathrm{ACP}(A, |, \tau)$ stands for a particular signature over fixed A and communication function $|$, and a set of axioms over this signature. Let

$\mathcal{P}$ denote the set of *process terms* over this signature:

sorts:	A	(the given, finite set of atomic actions),
	$\mathcal{P}$	(the set of process terms; $A \subseteq \mathcal{P}$),
operations:	$+ : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(non-deterministic choice or sum),
	$\cdot : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(sequential composition),
	$\parallel : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(merge),
	$\between : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(left merge),
	$\mid : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$	(communication merge, extending the given communication function),
	$\partial_H : \mathcal{P} \to \mathcal{P}$	(encapsulation, $H \subseteq A$),
	$\tau_I : \mathcal{P} \to \mathcal{P}$	(abstraction, $I \subseteq A$),
constants:	$\delta \in \mathcal{P}$	(deadlock),
	$\tau \in \mathcal{P}$	(silent step).

Intuitively, an action a represents indivisible behaviour, δ represents inaction, and τ represents invisible internal behaviour. Moreover, $P + Q$ executes either P or Q, $P \cdot Q$ first executes P and at its successful termination proceeds to execute Q, and $P \parallel Q$ executes P and Q in parallel allowing communication of actions from P and Q. The operators $P \between Q$ and $P \mid Q$ both capture part of the behaviour of $P \parallel Q$: $P \between Q$ takes its first transition from P, while the first transition of $P \mid Q$ is a communication of actions from P and Q. Finally, in $\partial_H(P)$ all actions from H in P are blocked, while in $\tau_I(P)$ all actions from I in P are renamed into τ.

We take $\cdot$ to be the operation that binds strongest, and $+$ the one that binds weakest. As usual in algebra, we tend to write xy instead of $x \cdot y$. For $\square \in \{+, \cdot, \parallel, \mid\}$ we will assume that expressions $P_0 \square \cdots \square P_n$ associate to the right. Furthermore, for $k \geq 1$ we define x^{k+1} as $x \cdot x^k$, and x^1 as x.

In Table 1 the axioms of the system $\mathrm{ACP}(A, \mid, \tau)$ are collected. Note that $+$ and $\cdot$ are associative, and that $+$ is moreover commutative and idempotent. In the case that $a \mid (b \mid c) = \delta$ for all $a, b, c \in A$, while $\mid$ itself defines a communication, we speak of *handshaking*. We will study the following subsystems of $\mathrm{ACP}(A, \mid, \tau)$:

- $\mathrm{BPA}(A)$. The signature of $\mathrm{BPA}(A)$ contains the elements of A, non-deterministic choice, and sequential composition. The axioms of $\mathrm{BPA}(A)$ are (A1)–(A5).
- $\mathrm{BPA}_\delta(A)$. The signature of $\mathrm{BPA}_\delta(A)$ is the signature of $\mathrm{BPA}(A)$ extended with the deadlock. The axioms of $\mathrm{BPA}_\delta(A)$ are (A1)–(A7).
- $\mathrm{PA}(A)$. The signature of $\mathrm{PA}(A)$ is the signature of $\mathrm{BPA}(A)$ extended with the merge and left merge. The axioms of $\mathrm{PA}(A)$ are (A1)–(A5), (CM2)–(CM4)

(with a ranging over A), and

(M1) $x \parallel y = x \between y + y \between x$.

- $\mathrm{PA}_\delta(A)$. The signature of $\mathrm{PA}_\delta(A)$ is the signature of $\mathrm{PA}(A)$ extended with the deadlock. The axioms of $\mathrm{PA}(A)$ are (A1)–(A7), (M1), and (CM2)–(CM4) (with a ranging over A_δ).
- $\mathrm{ACP}(A,|)$. The signature of $\mathrm{ACP}(A,|)$ is the signature of $\mathrm{ACP}(A,|,\tau)$ *without* the silent step and abstraction operators. The axioms of $\mathrm{ACP}(A,|)$ are (A1)–(A7), (CF1)–(CF2), (CM1)–(CM9), and (D1)–(D4) (with a, b ranging over A_δ).

We note that in $\mathrm{PA}(A)$ and $\mathrm{PA}_\delta(A)$, commutativity of the merge ($x \parallel y = y \parallel x$) can be derived from axioms (A1) and (M1).

The binary equality relation $=$ on process terms induced by an axiom system is obtained by taking all closed instantiations of axioms, and closing it under equivalence (i.e., under reflexivity, symmetry, and transitivity) and under contexts.

2.2 Transition Rules and Operational Semantics

We define a structural operational semantics in the style of Plotkin [68], to relate each process term to a labelled transition system. Then we define strong bisimulation as an equivalence between labelled transition systems, which carries over to process terms. The operational semantics and strong bisimulation are used in the proofs on axiomatisations in Sections 3 and 4, and on classification results in Section 5.

A *labelled transition system* (LTS) is a tuple $\langle S, \{\xrightarrow{a}, \xrightarrow{a}\surd \mid a \in A\}, s\rangle$, where

S is a set of *states*,
$\xrightarrow{a}$ for $a \in A$ is a binary relation between states,
$\xrightarrow{a}\surd$ for $a \in A$ is a unary predicate on states,
$s \in S$ is the *initial state*.

Expressions $s \xrightarrow{a} s'$ and $s \xrightarrow{a}\surd$ are called *transitions*. Intuitively, $s \xrightarrow{a} s'$ denotes that from state s one can evolve to state s' by the execution of action a, while $s \xrightarrow{a}\surd$ denotes that from state s one can terminate successfully by the execution of action a ($\surd$ is pronounced "tick").

Consider one of the process algebra axiom systems $\mathrm{BPA}(A) - \mathrm{ACP}(A,|,\tau)$, and let $\mathcal{P}$ represent all process terms given by its signature. We want to relate each process term in $\mathcal{P}$ to a labelled transition system. We take the process terms in $\mathcal{P}$ as the set of states, and the atomic actions in A as the set of labels. (Note that atomic actions can denote both states and labels.) Exploiting the syntactic structure of process terms, the transition relations $\xrightarrow{a}$ and $\xrightarrow{a}\surd$ for $a \in A$ are defined by means of inductive proof rules called *transition rules* $\frac{S}{c}$. Validity of the *premises* in

Table 1: The axioms for $\mathrm{ACP}(A, |, \tau)$, where $a, b \in A_{\delta\tau}$ and $H, I \subseteq A$.

(A1)	$x + y$	$=$	$y + x$
(A2)	$x + (y + z)$	$=$	$(x + y) + z$
(A3)	$x + x$	$=$	x
(A4)	$(x + y)z$	$=$	$xz + yz$
(A5)	$(xy)z$	$=$	$x(yz)$
(A6)	$x + \delta$	$=$	x
(A7)	δx	$=$	δ
(C1)	$a \mid b$	$=$	$b \mid a$
(C2)	$(a \mid b) \mid c$	$=$	$a \mid (b \mid c)$
(C3)	$\delta \mid a$	$=$	δ
(CM1)	$x \parallel y$	$=$	$(x \between y + y \between x) + x \mid y$
(CM2)	$a \between x$	$=$	ax
(CM3)	$ax \between y$	$=$	$a(x \parallel y)$
(CM4)	$(x + y) \between z$	$=$	$x \between z + y \between z$
(CM5)	$ax \mid b$	$=$	$(a \mid b)x$
(CM6)	$a \mid bx$	$=$	$(a \mid b)x$
(CM7)	$ax \mid by$	$=$	$(a \mid b)(x \parallel y)$
(CM8)	$(x + y) \mid z$	$=$	$x \mid z + y \mid z$
(CM9)	$x \mid (y + z)$	$=$	$x \mid y + x \mid z$
(D1)	$\partial_H(a)$	$=$	a if $a \notin H$
(D2)	$\partial_H(a)$	$=$	δ if $a \in H$
(D3)	$\partial_H(x + y)$	$=$	$\partial_H(x) + \partial_H(y)$
(D4)	$\partial_H(xy)$	$=$	$\partial_H(x)\partial_H(y)$
(B1)	$x\tau$	$=$	x
(TI1)	$\tau_I(a)$	$=$	a if $a \notin I$
(TI2)	$\tau_I(a)$	$=$	τ if $a \in I$
(TI3)	$\tau_I(x + y)$	$=$	$\tau_I(x) + \tau_I(y)$
(TI4)	$\tau_I(xy)$	$=$	$\tau_I(x)\tau_I(y)$

S, under a certain substitution, implies validity of the *conclusion* c under the same substitution.

The transition rules in Table 2 define the labelled transition system associated to a process term in $\mathrm{ACP}(A, |, \tau)$. The signature and parameters of $\mathcal{P}$ (possibly including a communication function $|$) determine which transition rules are appropriate. For example, the last two transition rules for $\|$ (i.e., with $x \| y$ in the left-hand sides of their conclusions) are not relevant for $\mathrm{PA}_\delta(A)$. Note that the deadlock δ has no outgoing transitions. The labelled transition system related to process term P has P itself as initial state. Often we will write simply P for the labelled transition system related to P.

Intuitively, *strong bisimulation* relates two states if the LTSs rooted at these states have the same branching structure. This semantics does not take into account the silent nature of τ.

Definition 2.2.1. [Strong bisimulation] A *strong bisimulation* is a binary, symmetric relation $\mathcal{R}$ over the set of states that satisfies

$$\begin{array}{lcl} P\,\mathcal{R}\,Q \;\wedge\; P \xrightarrow{a} P' & \Longrightarrow & \exists Q'\,(Q \xrightarrow{a} Q' \;\wedge\; P'\,\mathcal{R}\,Q') \\ P\,\mathcal{R}\,Q \;\wedge\; P \xrightarrow{a} \surd & \Longrightarrow & Q \xrightarrow{a} \surd \end{array}$$

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

Theorem 2.2.2.

1. (Equivalence) *It is not hard to see that strong bisimulation is an* equivalence *relation over* $\mathrm{ACP}(A, |)$.

2. (Congruence) *Strong bisimulation equivalence is a* congruence *relation up to* $\mathrm{ACP}(A, |)$, *meaning that* $P_0 \leftrightarroweq Q_0$ *and* $P_1 \leftrightarroweq Q_1$ *implies* $P_0 + P_1 \leftrightarroweq Q_0 + Q_1$, $P_0 P_1 \leftrightarroweq Q_0 Q_1$, $P_0 \| P_1 \leftrightarroweq Q_0 \| Q_1$, $P_0 \between P_1 \leftrightarroweq Q_0 \between Q_1$, $P_0 \mid P_1 \leftrightarroweq Q_0 \mid Q_1$, *and* $\partial_H(P_0) \leftrightarroweq \partial_H(Q_0)$. *This follows from the fact that the transition rules in Table 2 are in* path *format; see [27, 36, 49, 6].*

3. (Soundness) *Up to and including* $\mathrm{ACP}(A, |)$, *all axiom systems are* sound *with respect to strong bisimilation equivalence, meaning that* $P = Q$ *implies* $P \leftrightarroweq Q$. *Since strong bisimulation is a congruence, soundness follows from the fact that all closed instantiations of axioms in* $\mathrm{ACP}(A, |)$ *are valid in strong bisimulation semantics.*

4. (Completeness) *Up to and including* $\mathrm{ACP}(A, |)$, *all axiom systems are complete with respect to strong bisimilation equivalence, meaning that* $P \leftrightarroweq Q$ *implies* $P = Q$; *see, e.g., [28, 43].*

Table 2: Transition rules for $\mathrm{ACP}(A, |, \tau)$, where $a, b \in A_\tau,\ H, I \subseteq A$.

$$a \xrightarrow{a} \surd, \quad a \in A_\tau$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd} \qquad \frac{y \xrightarrow{a} \surd}{x + y \xrightarrow{a} \surd}$$

$$\frac{x \xrightarrow{a} x'}{xy \xrightarrow{a} x'y} \qquad \frac{x \xrightarrow{a} \surd}{xy \xrightarrow{a} y}$$

$$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} \qquad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} \qquad \frac{x \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} y} \qquad \frac{y \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} x}$$

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} \text{ if } a \mid b \in A \qquad \frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd}{x \parallel y \xrightarrow{a|b} \surd} \text{ if } a \mid b \in A$$

$$\frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} y'} \text{ if } a \mid b \in A \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd}{x \parallel y \xrightarrow{a|b} x'} \text{ if } a \mid b \in A$$

$$\frac{x \xrightarrow{a} x'}{x \between y \xrightarrow{a} x' \parallel y} \qquad \frac{x \xrightarrow{a} \surd}{x \between y \xrightarrow{a} y}$$

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x \mid y \xrightarrow{a|b} x' \parallel y'} \qquad \frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} y'}{x \mid y \xrightarrow{a|b} y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} \surd}{x \mid y \xrightarrow{a|b} x'} \qquad \frac{x \xrightarrow{a} \surd \quad y \xrightarrow{b} \surd}{x \mid y \xrightarrow{a|b} \surd}$$

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

$$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')} \text{ if } a \in I \qquad \frac{x \xrightarrow{a} \surd}{\tau_I(x) \xrightarrow{\tau} \surd} \text{ if } a \in I$$

$$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')} \text{ if } a \notin I \qquad \frac{x \xrightarrow{a} \surd}{\tau_I(x) \xrightarrow{a} \surd} \text{ if } a \notin I$$

We proceed to define some more semantic equivalence relations on states in a labelled transition system that do not take into account the silent nature of τ. These definitions use the following notions, assuming an underlying LTS.

Let $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q$ for $\sigma \in A^*$ denote that state P can evolve to state Q by the execution of the sequence of actions σ. This binary relation on states is defined as follows (with ϵ denoting the empty string):

- $P \stackrel{\epsilon}{\longrightarrow\!\!\!\!\!\rightarrow} P$;
- $\dfrac{P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q \quad Q \stackrel{a}{\longrightarrow} R}{P \stackrel{\sigma a}{\longrightarrow\!\!\!\!\!\rightarrow} R}$;
- $\dfrac{P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q \quad Q \stackrel{a}{\longrightarrow} \surd}{P \stackrel{\sigma a}{\longrightarrow\!\!\!\!\!\rightarrow} \surd}$.

A string $\sigma \in A^*$ is a *trace* of P if $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q$ for some state Q or $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} \surd$.

Definition 2.2.3. [Substate] Q is a *substate* of P if $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q$ for some $\sigma \in A^*$. Q is a *proper substate* of P if $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} Q$ for some $\sigma \in A^* \setminus \{\epsilon\}$. □

Definition 2.2.4. [Ready simulation] A *simulation* is a binary relation $\mathcal{R}$ over the set of states that satisfies:

$$\begin{array}{lcl} P\,\mathcal{R}\,Q \;\wedge\; P \stackrel{a}{\longrightarrow} P' & \Longrightarrow & \exists Q'\,(Q \stackrel{a}{\longrightarrow} Q' \;\wedge\; P'\,\mathcal{R}\,Q') \\ P\,\mathcal{R}\,Q \;\wedge\; P \stackrel{a}{\longrightarrow} \surd & \Longrightarrow & Q \stackrel{a}{\longrightarrow} \surd \end{array}$$

A simulation $\mathcal{R}$ is a *ready simulation* if whenever $P\,\mathcal{R}\,Q$ and a is a trace of Q, then a is a trace of P.

Two states P and Q are *simulation equivalent*, notation $P \simeq_S Q$, if $P\,\mathcal{R}_1\,Q$ and $Q\,\mathcal{R}_2\,P$ for simulations $\mathcal{R}_1$ and $\mathcal{R}_2$.

Two states P and Q are *ready simulation equivalent*, notation $P \simeq_{RS} Q$, if $P\,\mathcal{R}_1\,Q$ and $Q\,\mathcal{R}_2\,P$ for ready simulations $\mathcal{R}_1$ and $\mathcal{R}_2$. □

Definition 2.2.5. [Language equivalence] Two states P and Q are *language equivalent*, notation $P \simeq_L Q$, if for each trace $P \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} \surd$ there is a trace $Q \stackrel{\sigma}{\longrightarrow\!\!\!\!\!\rightarrow} \surd$, and vice versa. □

Definition 2.2.6. [Trace equivalence] Two states P and Q are *trace equivalent*, notation $P \simeq_T Q$, if they give rise to the same set of traces. □

In [45], VAN GLABBEEK gave a comparison of a wide range of behavioural equivalences and *preorders* (i.e., relations that are in general not symmetric) that do not take into account the silent nature of τ. Apart from the equivalence relation discussed above, he studied *completed trace* preorder and a variety of *decorated trace* preorders, which are based on decorated versions of traces. These preorders are coarser than strong bisimulation and ready simulation, but more refined than language and trace equivalence. In Section 4.3, prefix iteration is axiomatised with respect to some of the decorated trace preorders. The reader is referred to [45] for the definitions of these preorders.

Four standard semantic equivalences that take into account the silent nature of τ (and that constitute congruence relations over $\mathrm{ACP}(A,|,\tau)$) are rooted branching bisimulation [50], rooted delay bisimulation [60], rooted η-bisimulation [16], and rooted weak bisimulation [62]. Of these four semantics, rooted branching bisimulation constitutes the finest relation, while rooted weak bisimulation constitutes the coarsest relation. In all four semantics the axiom (B1) (i.e., $x\tau = x$) is valid, and that is all that is needed for the expressivity results concerning $\mathrm{ACP}(A,|,\tau)$ discussed in this chapter. In [47], VAN GLABBEEK gave a comparison of a wide range of process semantics that take into account the silent nature of τ.

2.3 Recursive Specifications

Although iterative operations are recursive by nature, we will use recursive specifications and associated notions to prove some of the results concerning these operations.

Definition 2.3.1. [Recursion] We assume a set of *recursion variables* $\{X_j \mid j \in J\}$ for some index set J. A *recursive specification* E is a set of *recursive equations* $X_j = T_j$ for $j \in J$, where T_j is an $\mathrm{ACP}(A,|,\tau)$-term in which the recursion variables X_i for $i \in J$ may occur.

Given some process semantics, processes P_j (for $j \in J$) form a *solution* of E if substitution of P_j for X_j in the recursive equations of E yields equations that hold in this semantics. □

If a recursive specification has solutions, then these solutions are often referred to by the names of the corresponding recursion variables in E. Table 3 presents the transition rules for recursive specifications.

Table 3: Transition rules for recursion.

$$\frac{T \xrightarrow{a} x'}{X \xrightarrow{a} x'} \text{ if } X = T \in E \qquad \frac{T \xrightarrow{a} \surd}{X \xrightarrow{a} \surd} \text{ if } X = T \in E$$

If for instance $E = \{X = aX + b\}$, then $X \xrightarrow{a} X$ by the first transition rule, and $X \xrightarrow{b} \surd$ by the second transition rule.

Recursive specifications need not have unique solutions in any reasonable process semantics, examples being $\{X = X\}$ and $\{X = \tau X\}$. The following two definitions are sufficient to remedy this imperfection.

Definition 2.3.2. [τ-Guardedness] Let P be an expression containing a recursion variable X.

An occurrence of X in P is *τ-guarded* if P has a subexpression aQ where $a \in A_\tau$ and Q contains this occurrence of X.

A recursive specification $E = \{X_j = T_j \mid j \in J\}$ is *τ-guarded* if by repeatedly substituting T_j expressions for occurrences of X_j, and by applying axioms of $\mathrm{ACP}(A, |, \tau)$, one can obtain the situation that all occurrences of recursion variables in right-hand sides of recursive equations are τ-guarded. □

Definition 2.3.3. [τ-Convergence] A recursive specification $E = \{X_j = T_j \mid j \in J\}$ is *τ-convergent* if $X_j \xrightarrow{\sigma} P$ (for $j \in J$) implies that there is no infinite τ-trace $P \xrightarrow{\tau} P' \xrightarrow{\tau} P'' \xrightarrow{\tau} \cdots$. □

Recursive specifications that are τ-guarded and τ-convergent have a unique solution modulo any reasonable process semantics. The existence of a solution underlies the soundness of the *recursive definition principle* [10]. We proceed to introduce the *recursive specification principle* (RSP), discussed in for instance [20, 10]. This principle states that the recursive specification $E = \{X_j = T_j \mid j \in J\}$ has at most one solution per recursion variable, modulo the process semantics under consideration.

$$\text{(RSP)} \quad \frac{y_j = T_j\{y_i/X_i \mid i \in J\} \text{ for } j \in J}{X_k = y_k} \quad (\text{for } k \in J)$$

(RSP) is sound with respect to the process equivalences mentioned thus far, provided E is both τ-convergent and τ-guarded. Note that (RSP) is not sound with respect to the recursive specifications $\{X = X\}$ and $\{X = \tau X\}$.

Definition 2.3.4. [Regular process] A process P is *regular* if P is bisimilar to a process $\tau_I(Q)$ where Q is a solution for a recursion variable in a finite recursive specification

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

where $\alpha_{i,j}$ and β_j are finite sums of actions in A (the empty sum representing δ). □

3 Axiomatisation of the Binary Kleene Star

In Section 1 we introduced the *binary Kleene star* (BKS), notation $*$. This operation is defined by the equation

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

This section considers its finite equational axiomatisation in strong bisimulation semantics, and presents a negative result on the finite equational axiomatisability of BKS in a variety of other process semantics.

3.1 Preliminaries

BPA$^*(A)$ is obtained by extending BPA(A) with BKS. Bergstra, Bethke, and Ponse [13] introduced an axiomatisation for BPA$^*(A)$ modulo strong bisimulation equivalence, which consists of axioms (A1)–(A5) for BPA(A), extended with the axioms (BKS1)–(BKS3) for BKS in Table 4. The axiom (BKS3) stems from Troeger [73].

Table 4: Axioms for BKS.

(BKS1)	$x(x^*y) + y$	$=$	x^*y
(BKS2)	$x^*(yz)$	$=$	$(x^*y)z$
(BKS3)	$x^*(y((x+y)^*z) + z)$	$=$	$(x+y)^*z$

In order to provide process terms over BPA$^*(A)$ with an operational semantics, we introduce transition rules for BKS. The transition rules for BKS in Table 5 express that x^*y repeatedly executes x until it executes y. Together with the transition rules for BPA(A) in Table 2 they provide labelled transition systems to process terms over BPA$^*(A)$. Note that by the first two transition rules in Table 5, a state P can have itself as a proper substate. For example, $a^*b \xrightarrow{a} a^*b$.

Table 5: Transition rules for BKS.

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

The transition rules for BKS are in *path* format [27, 49]. Hence, strong bisimulation equivalence is a congruence with respect to BPA$^*(A)$; see [36, 6]. Furthermore, its

axiomatisation is sound for BPA$^*(A)$ modulo strong bisimulation equivalence. Since strong bisimulation equivalence is a congruence, this can be verified by checking soundness for each axiom separately. It can be easily shown that the BKS axioms are valid in strong bisimulation.

Fokkink and Zantema [44] proved that the axiomatisation for BPA$^*(A)$ is complete modulo strong bisimulation equivalence. Their proof is based on a *term rewriting* analysis (see, e.g., [22]), in a quest to reduce bisimilar process terms to the same *ground normal form*, which does not reduce any further. Since this aim cannot be fulfilled for BKS, this operator is replaced by an operator representing $x(x^*y)$, and the BKS axioms are adopted to fit this new operator. Those axioms are turned into conditional rewrite rules, which are applied *modulo associativity and commutativity* of the $+$ (see, e.g., [67]). *Knuth-Bendix completion* [52] is applied to make the conditional rewrite system *weakly confluent*. *Termination* of the resulting conditional term rewriting system is obtained by means of the technique of *semantic labelling* from Zantema [75]. Hence, each process term is provably equal to a normal form. Finally, a careful case analysis learns that if two normal forms are strongly bisimilar, then they are syntactically equal modulo associativity and commutativity of the $+$. This observation yields the desired completeness result.

An alternative completeness proof was proposed in [39], based on induction on the structure of process terms. That proof method is more general, and was later on applied to obtain completeness results for axiomatisations of iteration operations in [1, 33, 40]. In the light of the generic applicability of this proof method and the significance of the completeness result in the realm of this chapter, we present the proof from [39] in some detail.

Following Milner [63] (see also [46]), the latter proof strategy can also be used to derive *ω-completeness* of the axiomatisation for BPA$^*(A)$. That is, if P and Q are *open* terms over BPA$^*(A)$, which may contain variables, and if $\sigma(P) = \sigma(Q)$ holds for all closed instantiations σ, then $P = Q$ can be derived from the axioms. Often, ω-completeness can be proved by providing variables with an operational semantics, such that $P \leftrightarrows Q$ holds with respect to this new operational semantics if and only if $\sigma(P) \leftrightarrows \sigma(Q)$ holds for all closed instantiations σ with respect to the original operational semantics. In [39], completeness of the axiomatisation for BPA$^*(A)$ modulo bisimulation equivalence is derived for open terms, which immediately implies ω-completeness of the axiomatisation. Here, we present the proof from [39] for closed (instead of open) terms. The reader is referred to [39] for a proof of ω-completeness. (The motivation to refrain from this generalisation here is clarity of presentation; we prefer to work in an unambiguous semantic framework throughout this chapter.)

3.2 Completeness

We note that each process term over BPA$^*(A)$ has only finitely many substates. In the sequel, process terms are considered modulo associativity and commutativity of the $+$, and we write $P =_{\mathrm{AC}} Q$ if P and Q can be equated by axioms (A1) and

(A2). As usual, $\sum_{i=1}^{n} P_i$ represents $P_1 + \cdots + P_n$. We take care to avoid empty sums (where $\sum_{i=1}^{0} P_i + Q$ is not considered empty and equals Q).

For each process term P, its collection of possible transitions is non-empty and finite, say $\{P \xrightarrow{a_i} P_i \mid i = 1, \ldots, m\} \cup \{P \xrightarrow{b_j} \surd \mid j = 1, \ldots, n\}$. We call

$$\sum_{i=1}^{m} a_i P_i + \sum_{j=1}^{n} b_j$$

the *HNF-expansion* of P (Head Normal Form expansion, cf. [28]). The process terms $a_i P_i$ and b_j are the *summands* of P.

Lemma 3.2.1. *Each process term is provably equal to its HNF-expansion.*

Proof. Straightforward, by structural induction, using (A4), (A5), and (BKS1). □

Process terms in BPA$^*(A)$ are *normed*, which means that they are able to terminate in finitely many transitions. The *norm* [11] of a process yields the length of the shortest termination trace of this process. Norm can be defined inductively by

$$\begin{aligned} |a| &= 1 \\ |P + Q| &= \min\{|P|, |Q|\} \\ |PQ| &= |P| + |Q| \\ |P^* Q| &= |Q|. \end{aligned}$$

We note that strongly bisimilar processes have the same norm. The following lemma, due to Caucal [29], is typical for normed processes.

Lemma 3.2.2. *Let $PQ \leftrightarrows RS$. By symmetry we may assume $|Q| \leq |S|$. We can distinguish two cases:*

- *either $P \leftrightarrows R$ and $Q \leftrightarrows S$;*
- *or there is a proper substate P' of P such that $P \leftrightarrows RP'$ and $P'Q \leftrightarrows S$.*

Proof. This lemma follows from the following facts A and B.

A. If $PQ \leftrightarrows RS$ and $|Q| \leq |S|$, then either $Q \leftrightarrows S$, or there is a proper substate P' of P such that $P'Q \leftrightarrows S$.

We prove fact A by induction on $|P|$. First, let $|P| = 1$. Then $P \xrightarrow{a} \surd$ for some a, so $PQ \xrightarrow{a} Q$. Since $PQ \leftrightarrows RS$, and $|R'S| > |S| \geq |Q|$ for all substates R' of R, it follows that $R \xrightarrow{a} \surd$ and $Q \leftrightarrows S$. Then we are done.

Next, suppose we have proved the case for $|P| \leq n$, and let $|P| = n + 1$. Then there is a P' with $|P'| = n$ and $P \xrightarrow{a} P'$, which implies $PQ \xrightarrow{a} P'Q$. Since $PQ \leftrightarrows RS$, we have two options:

1. $R \xrightarrow{a} \surd$ and $P'Q \leftrightarrows S$. Then we are done.
2. $R \xrightarrow{a} R'$ and $P'Q \leftrightarrows R'S$. Since $|P'| = n$, induction yields either $Q \leftrightarrows S$ or $P''Q \leftrightarrows S$ for a proper substate P'' of P'. Again, we are done.

This concludes the proof of fact A.

B. If $PQ \leftrightarrows RQ$, then $P \leftrightarrows R$.

Define a binary relation $\mathcal{B}$ on process terms by $T \mathcal{B} U$ iff $TQ \leftrightarrows UQ$. We show that $\mathcal{B}$ constitutes a strong bisimulation relation between P and R.

- Since $\leftrightarrows$ is symmetric, so is $\mathcal{B}$.
- $PQ \leftrightarrows RQ$, so $P \mathcal{B} R$.
- Suppose $T \mathcal{B} U$ and $T \xrightarrow{a} \surd$. Then $TQ \xrightarrow{a} Q$. Since $TQ \leftrightarrows UQ$, and $|U'Q| > Q$ for all substates U' of U, it follows that $UQ \xrightarrow{a} Q$. In other words, $U \xrightarrow{a} \surd$.
- Suppose $T \mathcal{B} U$ and $T \xrightarrow{a} T'$. Then $TQ \xrightarrow{a} T'Q$. Since $TQ \leftrightarrows UQ$, and $|Q| < |T'Q|$, it follows that there is a transition $U \xrightarrow{a} U'$ with $T'Q \leftrightarrows U'Q$. Hence, $T' \mathcal{B} U'$.

This concludes the proof of fact B.

Finally, we show that facts A and B together prove the lemma. Let $PQ \leftrightarrows RS$ with $|Q| \leq |S|$. According to fact A we can distinguish two cases.

- $Q \leftrightarrows S$. Then $PQ \leftrightarrows RS \leftrightarrows RQ$, so fact B yields $P \leftrightarrows R$.
- $P'Q \leftrightarrows S$ for some proper substate P' of P. Then $PQ \leftrightarrows RS \leftrightarrows RP'Q$, so fact B yields $P \leftrightarrows RP'$.

□

We construct a set $\mathbb{B}$ of *basic terms*, such that each process term is provably equal to a basic term. The completeness theorem is proved by showing that strongly bisimilar basic terms are provably equal.

$$\begin{array}{rcl} (x+y)z & \longrightarrow & xz+yz \\ (xy)z & \longrightarrow & x(yz) \\ (x^*y)z & \longrightarrow & x^*(yz) \end{array}$$

The term rewriting system above consists of directions of the axioms (A4), (A5), and (BKS2), pointing from left to right. Its rewrite rules are to be interpreted modulo

associativity and commutativity of the $+$. The term rewriting system is terminating, meaning that there are no infinite reductions. This follows from the following weight function w in the natural numbers:

$$\begin{aligned} w(a) &= 2 \\ w(P+Q) &= w(P)+w(Q) \\ w(PQ) &= w(P)^2 w(Q) \\ w(P^*Q) &= w(P)+w(Q). \end{aligned}$$

It is not hard to see that if P reduces to Q in one or more rewrite steps, then $w(P) > w(Q)$. Since the ordering on the natural numbers is well-founded, we conclude that the term rewriting system is terminating. Let $\mathbb{G}$ denote the collection of *ground normal forms*, i.e., the collection of process terms that cannot be reduced by any of the three rewrite rules. Since the term rewriting system is terminating, and since its rewrite rules are directions of axioms, it follows that each process term is provably equal to a process term in $\mathbb{G}$. The elements in $\mathbb{G}$ are defined by:

$$P ::= a \mid P+P \mid aP \mid P^*P.$$

$\mathbb{G}$ is not yet our desired set of basic terms, due to the fact that there exist process terms in $\mathbb{G}$ which have a substate outside $\mathbb{G}$. We give an example.

Example 3.2.3. *Let $A = \{a, b, c\}$. Clearly, $(a^*b)^*c \in \mathbb{G}$, and*

$$(a^*b)^*c \xrightarrow{a} (a^*b)((a^*b)^*c).$$

The substate $(a^*b)((a^*b)^*c)$ is not in $\mathbb{G}$, because the third rewrite rule in $\mathcal{R}$ reduces this process term to $a^*(b((a^*b)^*c))$.

In order to overcome this complication, we introduce the following collection of process terms:

$$\mathbb{H} = \{P^*Q,\ P'(P^*Q) \mid P^*Q \in \mathbb{G} \text{ and } P' \text{ is a proper substate of } P\}.$$

We define an equivalence relation $\cong$ on $\mathbb{H}$ by putting $P'(P^*Q) \cong P^*Q$ for proper substates P' of P, and taking the reflexive, symmetric, transitive closure of $\cong$.

The set $\mathbb{B}$ of *basic terms* is the union of $\mathbb{G}$ and $\mathbb{H}$.

Lemma 3.2.4. *If $P \in \mathbb{B}$ and $P \xrightarrow{a} P'$, then $P' \in \mathbb{B}$.*

Proof. By induction on the structure of P.

If $P \in \mathbb{H} \setminus \mathbb{G}$, then it is of the form $Q'(Q^*R)$ for some $Q^*R \in \mathbb{G}$. So P' is of the form either Q^*R or $Q''(Q^*R)$ for some proper substate Q'' of Q'. In both cases, $P' \in \mathbb{B}$.

If $P \in \mathbb{G}$, then it is of the form $\sum_i a_i Q_i + \sum_j {R_j}^* S_j + \sum_k b_k$, where the Q_i, R_j, and S_j are in $\mathbb{G}$. So P' is of the form Q_i, ${R_j}^* S_j$, $R_j'({R_j}^* S_j)$, or S_j', which are all basic terms (in the last case, this follows by structural induction). $\square$

The *L-value* [44] of a process term is defined by

$$L(P) = \max\{|P'| \mid P' \text{ proper substate of } P\}.$$

$L(P) < L(PQ)$ and $L(P) < L(P^*Q)$, because for each proper substate P' of P, $P'Q$ is a proper substate of PQ and $P'(P^*Q)$ is a proper substate of P^*Q. Since norm is preserved under strong bisimulation, it follows that the same holds for L-value; i.e., if $P \leftrightarrows Q$ then $L(P) = L(Q)$.

We define an ordering $\prec$ on $\mathbb{B}$ as follows:

- $P \prec Q$ if $L(P) < L(Q)$;
- $P \prec Q$ if P is a substate of Q but Q is not a substate of P;
- if $P \prec Q$ and $Q \prec R$, then $P \prec R$.

Note that if $P, Q \in \mathbb{H}$ with $P \cong Q$, then P and Q have the same proper substates, and so $L(P) = L(Q)$. These observations imply that the ordering $\prec$ on $\mathbb{B}$ respects the equivalence $\cong$ on $\mathbb{H}$, that is, if $P \cong Q \prec R \cong S$, then $P \prec S$.

Lemma 3.2.5. *$\prec$ is a well-founded ordering on $\mathbb{B}$.*

Proof. If P is a substate of Q, then all proper substates of P are proper substates of Q, so $L(P) \leq L(Q)$. Hence, if $P \prec Q$ then $L(P) \leq L(Q)$.

Assume, toward a contradiction, that there exists an infinite backward chain $\cdots \prec P_2 \prec P_1 \prec P_0$. Since $L(P_{n+1}) \leq L(P_n)$ for all $n \in \mathbb{N}$, there is an N such that $L(P_n) = L(P_N)$ for all $n > N$. Since $P_n \prec P_N$ for $n > N$, it follows that P_n is a substate of P_N for $n > N$. Each process term has only finitely many substates, so there are $m, n > N$ with $n > m$ and $P_n =_{\mathrm{AC}} P_m$. Then $P_n \not\prec P_m$, so we have found a contradiction. Hence, $\prec$ is well-founded. □

In the proofs of the next two lemmas, we need a weight function g in the natural numbers, which is defined inductively by

$$\begin{aligned} g(a) &= 0 \\ g(P + Q) &= \max\{g(P), g(Q)\} \\ g(PQ) &= \max\{g(P), g(Q)\} \\ g(P^*Q) &= \max\{g(P), g(Q) + 1\}. \end{aligned}$$

It is not hard to see, by structural induction, that if $P \xrightarrow{a} P'$, then $g(P) \geq g(P')$.

Lemma 3.2.6. *Let $P^*Q \in \mathbb{B}$. If Q' is a proper substate of Q, then $Q' \prec P^*Q$.*

Proof. Q' is a substate of Q, so $g(Q') \leq g(Q) < g(P^*Q)$, which implies that P^*Q cannot be a substate of Q'. On the other hand, Q' is a substate of P^*Q, so $Q' \prec P^*Q$. □

Lemma 3.2.7. *If $P \in \mathbb{B}$ and $P \xrightarrow{a} P'$, then either $P' \prec P$, or $P, P' \in \mathbb{H}$ and $P \cong P'$.*

Proof. This lemma follows from the following facts A and B.

A. If $P \in \mathbb{B}$ and $P \xrightarrow{a} P'$, then either $P' \in \mathbb{H}$ or P' has smaller size than P.

We prove fact A by induction on the structure of P. Let

$$P =_{\mathrm{AC}} \sum_i a_i Q_i + \sum_j {R_j}^* S_j + \sum_k b_k.$$

Since $P \xrightarrow{a} P'$, P' is of one of the following forms.

- $P' =_{\mathrm{AC}} Q_i$ for some i. Then P' has smaller size than P.
- $P' =_{\mathrm{AC}} R_j'({R_j}^* S_j)$ or $P' =_{\mathrm{AC}} {R_j}^* S_j$ for some j. Then $P' \in \mathbb{H}$.
- $S_j \xrightarrow{a} P'$ for some j. Then induction yields that either $P' \in \mathbb{H}$, or P' has smaller size than S_j, which in turn has smaller size than P.

This concludes the proof of fact A.

B. If $P \in \mathbb{H}$ and $P \xrightarrow{a} P'$, then either $g(P) > g(P')$, or $P' \in \mathbb{H}$ and $P \cong P'$.

Since $P \in \mathbb{H}$, either $P =_{\mathrm{AC}} Q'(Q^*R)$ or $P =_{\mathrm{AC}} Q^*R$ for some Q and R. Hence, $P' =_{\mathrm{AC}} Q''(Q^*R)$, $P' =_{\mathrm{AC}} Q^*R$, or $P' =_{\mathrm{AC}} R'$ for a proper substate R' of R. In the first two cases $P' \in \mathbb{H}$ and $P \cong P'$, and in the last case $g(P') = g(R') \leq g(R) < g(Q^*R) = g(P)$. This concludes the proof of fact B.

Finally, we show that facts A and B together prove the lemma. Let $P \xrightarrow{a} P'$ with $P' \not\prec P$; we prove that $P, P' \in \mathbb{H}$ and $P \cong P'$.

Since P' is a substate of P and $P' \not\prec P$, P is a substate of P'. So there exists a sequence of transitions

$$P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} P_n \qquad (n \geq 1)$$

where $P_0 =_{\mathrm{AC}} P$, $P_1 =_{\mathrm{AC}} P'$, and $P_n =_{\mathrm{AC}} P$.

Suppose $P_k \notin \mathbb{H}$ for all k. Then according to fact A, P_{k+1} has smaller size than P_k for $k = 0, \ldots, n-1$, so P_n has smaller size than P_0. This contradicts $P_0 =_{\mathrm{AC}} P =_{\mathrm{AC}} P_n$. Hence, $P_l \in \mathbb{H}$ for some l.

Since each P_k is a substate of each $P_{k'}$, $g(P_k)$ must be the same for all k. Then it follows from fact B, together with $P_l \in \mathbb{H}$, that $P_k \in \mathbb{H}$ for all k and $P_0 \cong P_1 \cong \cdots \cong P_n$. □

Now we are ready to prove the desired completeness result for $\mathrm{BPA}^*(A)$.

Theorem 3.2.8. (A1)–(A5), (BKS1)–(BKS3) *is complete for* $\mathrm{BPA}^*(A)$ *modulo strong bisimulation equivalence.*

Proof. Each process term is provably equal to a basic term, so it is sufficient to show that strongly bisimilar basic terms are provably equal. Assume $P, Q \in \mathbb{B}$ with $P \leftrightarrows Q$; we show that $P = Q$, by induction on the ordering $\prec$. To be precise, we assume that we have already dealt with strongly bisimilar pairs $R, S \in \mathbb{B}$ with $R \prec P$ and $S \prec Q$, or $R \prec P$ and $S \cong Q$, or $R \cong P$ and $S \prec Q$.

First, assume that P or Q is not in $\mathbb{H}$, say $P \notin \mathbb{H}$. By the induction hypothesis, together with Lemma 3.2.7, for all transitions $P \xrightarrow{a} P'$ and $Q \xrightarrow{a} Q'$ with $P' \leftrightarrows Q'$ we have $P = Q$. Since $P \leftrightarrows Q$, axiom (A3) can be used to adapt the HNF-expansions of P and Q to the form

$$P = \sum_{i=1}^{m} a_i P_i + \sum_{j=1}^{n} b_j \qquad Q = \sum_{i=1}^{m} a_i Q_i + \sum_{j=1}^{n} b_j,$$

where $P_i = Q_i$ for $i = 1, \ldots, m$. Hence, $P = Q$.

Next, assume $P, Q \in \mathbb{H}$. We distinguish three cases.

1. Let $P =_{\mathrm{AC}} R^*S$ and $Q =_{\mathrm{AC}} T^*U$. We prove $R^*S = T^*U$.

   We spell out the HNF-expansions of R and T:

   $$R = \sum_{i \in I} R_i, \qquad T = \sum_{j \in J} T_j,$$

   where the R_i and the T_j are of the form either aV or a.

   Since $R^*S \leftrightarrows T^*U$, each $R_i(R^*S)$ for $i \in I$ is strongly bisimilar either to $T_j(T^*U)$ for a $j \in J$ or to a summand of U. We distinguish these two cases.

   (a) $R_i(R^*S) \leftrightarrows T_j(T^*U)$ for a $j \in J$. Then $R_i(R^*S) \leftrightarrows T_j(R^*S)$, so by Lemma 3.2.2 $R_i \leftrightarrows T_j$.

   (b) $R_i(R^*S) \leftrightarrows aU'$ for a transition $U \xrightarrow{a} U'$.

   Thus, I can be divided into the following, not necessarily disjoint, subsets:

   $$\begin{aligned} I_0 &= \{i \in I \mid \exists j \in J\ (R_i \leftrightarrows T_j)\} \\ I_1 &= \{i \in I \mid \exists U \xrightarrow{a} U'\ (R_i(R^*S) \leftrightarrows aU')\}. \end{aligned}$$

Similarly, J can be divided:

$$\begin{array}{lcl} J_0 & = & \{j \in J \mid \exists i \in I\ (T_j \leftrightarrows R_i)\} \\ J_1 & = & \{j \in J \mid \exists S \xrightarrow{a} S'\ (T_j(T^*U) \leftrightarrows aS')\}. \end{array}$$

If both I_1 and J_1 are non-empty, then $U'' \leftrightarrows R^*S$ for a proper substate U'' of U, and $S'' \leftrightarrows T^*U$ for a proper substate S'' of S, and so $U'' \leftrightarrows S''$. By Lemma 3.2.6, $S'' \prec R^*S$ and $U'' \prec T^*U$, so induction yields $R^*S = U'' = S'' = T^*U$, and we are done.

Hence, we may assume that either I_1 or J_1 is empty, say $J_1 = \emptyset$. We proceed to derive

$$\sum_{i \in I_1} R_i(R^*S) + S = U. \tag{1}$$

We show that each summand at the left-hand side of the equality sign is provably equal to a summand of U, and vice versa.

- By definition of I_1, for each $R_i(R^*S)$ with $i \in I_1$ there is a summand aU' of U such that $R_i(R^*S) \leftrightarrows aU'$. By Lemma 3.2.6 $U' \prec T^*U$, so induction yields $R_i(R^*S) = aU'$.
- Consider a summand aS' of S. Since $R^*S \leftrightarrows T^*U$ and $J_1 = \emptyset$, it follows that aS' is strongly bisimilar to a summand aU' of U, so induction yields $aS' = aU'$.
- Finally, summands a of S correspond with summands a of U.
- By the converse arguments it follows that each summand of U is provably equal to a summand at the left-hand side of the equality sign.

This concludes the derivation of (1).

Since $J_1 = \emptyset$, it follows that $J_0 \neq \emptyset$, so $I_0 \neq \emptyset$. By the definitions of I_0 and $J_0 = J$, each R_i with $i \in I_0$ is strongly bisimilar to a T_j with $j \in J$, and vice versa. Since $L(R_i) \leq L(R) < L(R^*S)$, induction yields $R_i = T_j$. Hence,

$$\sum_{i \in I_0} R_i = T. \tag{2}$$

Finally, we derive

$$\begin{array}{lcl} R^*S & \overset{\text{(A3)}}{=} & (\sum_{i \in I_0} R_i + \sum_{i \in I_1} R_i)^*S \\ & \overset{\text{(BKS3),(A3)}}{=} & (\sum_{i \in I_0} R_i)^*(\sum_{i \in I_1} R_i(R^*S) + S) \\ & \overset{\text{(1),(2)}}{=} & T^*U. \end{array}$$

2. Let $P =_{\mathrm{AC}} R'(R^*S)$ and $Q =_{\mathrm{AC}} T^*U$. We prove $R'(R^*S) = T^*U$.

   $|U| = |T^*U| = |R'(R^*S)| \geq 2$ implies that U does not have atomic summands, so its HNF-expansion is of the form $\sum_i a_i U_i$. Since $R'(R^*S) \leftrightarrows T^*U$, each U_i is strongly bisimilar to R^*S or to $R''(R^*S)$ for a proper substate R'' of R'. According to Lemma 3.2.6 $U_i \prec T^*U$, so induction yields $U_i = R^*S$ or $U_i = R''(R^*S)$. This holds for all i, so $U = \sum_i a_i U_i = V(R^*S)$ for some process term V. Then $R'(R^*S) \leftrightarrows T^*U \leftrightarrows (T^*V)(R^*S)$, so Lemma 3.2.2 implies $R' \leftrightarrows T^*V$. Since $L(R') < L(R'(R^*S))$, induction yields $R' = T^*V$. Hence, $R'(R^*S) = (T^*V)(R^*S) \overset{(\mathrm{BKS2})}{=} T^*(V(R^*S)) = T^*U$.

3. Let $P =_{\mathrm{AC}} R'(R^*S)$ and $Q =_{\mathrm{AC}} T'(T^*U)$. We prove $R'(R^*S) = T'(T^*U)$.

   By symmetry we may assume $|R^*S| \leq |T^*U|$. Lemma 3.2.2 distinguishes two possible cases.

   - $R' \leftrightarrows T'$ and $R^*S \leftrightarrows T^*U$. Since $L(R') < L(R'(R^*S))$, induction yields $R' = T'$, and case 1 applied to $R^*S \leftrightarrows T^*U$ yields $R^*S = T^*U$. Hence, $R'(R^*S) = T'(T^*U)$.
   - $R' \leftrightarrows T'R''$ and $R''(R^*S) \leftrightarrows T^*U$ for a proper substate R'' of R'. Since $L(R') < L(R'(R^*S))$, induction yields $R' = T'R''$. Furthermore, case 1 applied to $R''(R^*S) \leftrightarrows T^*U$ yields $R''(R^*S) = T^*U$. Hence, $R'(R^*S) = (T'R'')(R^*S) \overset{(\mathrm{BKS2})}{=} T'(R''(R^*S)) = T'(T^*U)$.

This finishes the derivation of $P = Q$. □

3.3 Irredundancy of the Axioms

Fokkink [39] showed that each of the BKS axioms is essential for the obtained completeness result.

Theorem 3.3.1. *If one of the BKS axioms is skipped from* (A1)–(A5)*,* (BKS1)–(BKS3)*, then this axiomatisation is no longer complete for* $\mathrm{BPA}^*(A)$ *modulo strong bisimulation equivalence.*

Proof. We apply a standard technique for proving that an equation e cannot be derived from an equational theory $\mathcal{E}$, which prescribes to define a model for $\mathcal{E}$ in which e is not valid.

In order to show that (BKS1) cannot be derived from (A1)–(A5), (BKS2), (BKS3), we define an interpretation function ϕ of open terms in the natural numbers:

$$\begin{aligned}
\phi(a) &= 0\\
\phi(x) &= 0\\
\phi(P+Q) &= \max\{\phi(P), \phi(Q)\}\\
\phi(PQ) &= \phi(P)\\
\phi(P^*Q) &= \max\{\phi(P)+1, \phi(Q)+1\}.
\end{aligned}$$

It is easy to see that this interpretation is a model for (A1)–(A5), (BKS2), (BKS3). However, $\phi(a(a^*a)+a)=0$, while $\phi(a^*a)=1$. Hence, $a(a^*a)+a=a^*a$ cannot be derived from (A1)–(A5), (BKS2), (BKS3).

In order to show that (BKS2) cannot be derived from (A1)–(A5), (BKS1), (BKS3) we define an interpretation function ψ of open terms in the natural numbers:

$$\begin{aligned}
\psi(a) &= 0\\
\psi(x) &= 0\\
\psi(P+Q) &= \max\{\psi(P),\psi(Q)\}\\
\psi(PQ) &= \psi(Q)\\
\psi(P^*Q) &= \max\{\psi(P)+1,\psi(Q)\}.
\end{aligned}$$

It is easy to see that this interpretation is a model for (A1)–(A5), (BKS1), (BKS3). However, $\psi((a^*a)a)=\psi(a)=0$, while $\psi(a^*(aa))=\max\{\psi(a)+1,\psi(aa)\}=1$. Hence, $(a^*a)a=a^*(aa)$ cannot be derived from (A1)–(A5), (BKS1), (BKS3).

In order to show that (BKS3) cannot be derived from (A1)–(A5), (BKS1), (BKS2), we define an interpretation function η of open terms in sets of natural numbers:

$$\begin{aligned}
\eta(a) &= \emptyset\\
\eta(x) &= \emptyset\\
\eta(P+Q) &= \eta(P)\cup\eta(Q)\\
\eta(PQ) &= \eta(P)\cup\eta(Q)\\
\eta(P^*Q) &= \eta(P)\cup\eta(Q)\cup\{|P|\}.
\end{aligned}$$

It is easy to see that this interpretation is a model for (A1)–(A5), (BKS1), (BKS2). However, $\eta((aa)^*(a((aa+a)^*a)+a))=\{|aa|,|aa+a|\}=\{1,2\}$ while $\eta((aa+a)^*a)=\{|aa+a|\}=\{1\}$. Hence, $(aa)^*(a((aa+a)^*a)+a)=(aa+a)^*a$ cannot be derived from (A1)–(A5), (BKS1), (BKS2). □

3.4 Negative Results

In contrast with the positive result on the finite equational axiomatisability of BPA$^*(A)$ modulo strong bisimulation equivalence, Aceto, Fokkink, and Ingólfsdóttir [4] showed that BPA$^*(A)$ is not finitely based modulo any process semantics in between *ready simulation* (see Definition 2.2.4) and *language equivalence* (see Definition 2.2.5). In the case of a singleton alphabet, this answered a problem in regular languages raised by Salomaa in [71]; see [3]. Crvenković, Dolinka, and Ésik [30] provided a more elegant answer to the latter question.

Ready simulation and language equivalence constitute congruence relations over BPA$^*(A)$ (in the case of language equivalence this follows from the fact that the transition rules for BPA$^*(A)$ are in *L cool* format [41]). Process semantics in the linear/branching time spectrum [45] that are finer than language equivalence and coarser than ready simulation, and which constitute congruence relations over

BPA$^*(A)$, are failure semantics, ready semantics, failure trace semantics, and ready trace semantics.

The result above follows from the existence of an infinite set of equations that cannot all be proved by means of any finite set of equations that is sound modulo language equivalence. This family of equations consists of

$$\text{E}.n \quad a^*(a^n) + (a^n)^*(a + \cdots + a^n) \;=\; (a^n)^*(a + \cdots + a^n)$$

for $n \geq 1$, where a is some action. Ready simulation is the finest semantics in the linear/branching time spectrum in which the E.n are sound. Note that for $n > 1$, none of the equations E.n is sound in strong bisimulation equivalence.

Given a finite set of equations that is sound with respect to language equivalence, Aceto, Fokkink, and Ingólfsdóttir construct a model $\mathcal{A}_p$ for these equations in which equation E.p fails, for some prime number p. The model that is used for this purpose is based on an adaptation of a construction due to Conway [35], who used it to obtain a new proof of a theorem, originally due to Redko [69], saying that BPA$^*(A)$ is not finitely based modulo language equivalence.

Let a be an action. For p a prime number, the carrier A_p of the algebra $\mathcal{A}_p$ consists of non-empty formal sums of $a^0, a^1, \ldots, a^{p-1}$, together with the formal symbol a^*, that is,

$$\{\sum_{i \in I} a^i \mid \emptyset \subset I \subseteq \{0, \ldots, p-1\}\} \cup \{a^*\} \;.$$

The syntax of $\mathcal{A}_p$ contains three more operators, which are semantic counterparts of the binary function symbols in BPA$^*(A)$. In order to avoid confusion, circled symbols denote the operators in the algebra $\mathcal{A}_p$: $\oplus$, $\odot$, and $\circledast$ represent the semantic counterparts of $+$, $\cdot$, and $*$, respectively. Table 6 presents an axiomatisation for $\mathcal{A}_p$.

Table 6: Axiomatisation for the algebra $\mathcal{A}_p$

$$\begin{array}{rcl}
a^* \oplus x & = & a^* \\
x \oplus a^* & = & a^* \\
\sum_{i \in I} a^i \oplus \sum_{j \in J} a^j & = & \sum_{h \in I \cup J} a^h \\
 & & \\
a^* \odot x & = & a^* \\
x \odot a^* & = & a^* \\
\sum_{i \in I} a^i \odot \sum_{j \in J} a^j & = & \sum_{h \in \{(i+j) \bmod p \,\mid\, (i,j) \in I \times J\}} a^h \\
 & & \\
x \circledast y & = & \begin{cases} y & \text{if } x = a^0 \\ a^* & \text{otherwise} \end{cases}
\end{array}$$

Process terms over BPA$^*(A)$ are mapped to $\mathcal{A}_p$ as expected: every action in A is mapped to the symbol a^1, while $+$, $\cdot$, and $*$ are mapped to $\oplus$, $\odot$, and $\circledast$, respectively.

For a process term P, the denotation of P in the algebra $\mathcal{A}_p$ is represented by $\mathcal{A}_p[\![P]\!]$. We note that equation E.p fails in $\mathcal{A}_p$. Namely,

$$\mathcal{A}_p[\![a^*(a^p) + (a^p)^*(a + \cdots + a^p)]\!] = a^* \neq \sum_{i=0}^{p-1} a^i = \mathcal{A}_p[\![(a^p)^*(a + \cdots + a^p)]\!] \ .$$

The following theorem is the key to the nonaxiomatisability result from [4].

Theorem 3.4.1. *For every finite set $\mathcal{E}$ of equations that are sound with respect to $\simeq_L$, there exists a prime number p such that all equations in $\mathcal{E}$ are valid in $\mathcal{A}_p$.*

Corollary 3.4.2. *No congruence relation over* BPA$^*(A)$ *that is included in $\simeq_L$ and satisfies E.n for all $n \geq 1$ has a complete finite equational axiomatisation.*

A process semantics that is coarser than language equivalence is *trace equivalence* $\simeq_T$, where two process terms are considered equivalent if they give rise to the same (not necessarily terminating) traces (see Definition 2.2.6). If $|A| > 1$, then trace equivalence is not a congruence relation over BPA(A); e.g., $a + aa \simeq_T aa$, but $(a + aa)b \not\simeq_T aab$. However, if the set A of actions is a singleton $\{a\}$, then trace equivalence constitutes a congruence relation over BPA$^*(A)$. In contrast with their negative results on the finite axiomatisability of BPA$^*(A)$ modulo process semantics between ready simulation and language equivalence, Aceto, Fokkink, and Ingólfsdóttir showed that BPA$^*(\{a\})$ modulo trace equivalence is axiomatised completely by the five axioms for BPA$(\{a\})$ together with the three axioms in Table 7.

Table 7: Axioms for trace equivalence ($A = \{a\}$)

(T1)	$x + (y^*z)$	$=$	a^*a
(T2)	$x + xy$	$=$	xy
(T3)	xy	$=$	yx

Theorem 3.4.3. (A1)–(A5), (T1)–(T3) *is complete for* BPA$^*(\{a\})$ *modulo trace equivalence.*

3.5 Extensions of BPA$^*(A)$

The signature of BPA$^*_\delta(A)$ is obtained by extending BPA$_\delta(A)$ with BKS. Its axioms are those of BPA$^*(A)$ and of BPA$_\delta(A)$, i.e., (A1)–(A7) in Table 1 for BPA$_\delta(A)$,

and (BKS1)–(BKS3) for BKS. Sewell [72] showed that there does not exist a complete equational axiomatisation for $\mathrm{BPA}^*_\delta(A)$ modulo strong bisimulation equivalence. This motivates the introduction of the implicational axiom

$$(\mathrm{RSP}^*) \quad \frac{x = yx + z}{x = y^* z}$$

It remains an open question, dating back to Milner [61], whether (A1)–(A7), (BKS1), (RSP^*) is complete modulo strong bisimulation equivalence. We note that (BKS2) and (BKS3) can be derived from this axiomatisation.

The signature of $\mathrm{PA}^*(A)$ is obtained by extending $\mathrm{PA}(A)$ with BKS. The axioms of $\mathrm{PA}^*(A)$ are those of $\mathrm{PA}(A)$ and (BKS1)–(BKS3). The system $\mathrm{PA}_\delta(A)$ can be extended in a similar way to $\mathrm{PA}^*_\delta(A)$. The system $\mathrm{ACP}^*(A, |)$ is defined by inclusion of (BKS1)–(BKS4); see Table 8. Note that (BKS4) can be derived using (RSP^*). Finally, $\mathrm{ACP}^*(A, |, \tau)$ is obtained by inclusion into $\mathrm{ACP}(A, |, \tau)$ of (BKS1)–(BKS5); see Table 8. Note that (RSP^*) is not sound for $\mathrm{ACP}^*(A, |, \tau)$; e.g.,

$$\tau = \tau\tau + \delta,$$

but $\tau = \tau^*\delta$ is not a desirable identity in any process semantics.

Table 8: Axioms for BKS with encapsulation and abstraction.

(BKS4)	$\partial_H(x^*y)$	$=$	$\partial_H(x)^*\partial_H(y)$
(BKS5)	$\tau_I(x^*y)$	$=$	$\tau_I(x)^*\tau_I(y)$

4 Axiomatisations of Other Iterative Operations

This section considers four restricted versions and one generalised version of BKS. We discuss the different advantages of each of these operators, and formulate various axiomatisations and completeness results.

4.1 Axiomatisation of No-Exit Iteration

No-exit iteration (NEI) x^ω is bisimilar to $x^*\delta$. No-exit iteration can be used to formally describe programs that repeat a certain procedure without end. Many communication protocols can be expressed, and shown correct, using no-exit iteration.

An explanation is that (concurrent) components of such protocols often perform repetitive behaviour in the following style (receive/process/send-repetition):

$$(\sum_{d\in D} r_i(d)\, P\, s_j(d))^\omega,$$

or

$$(\sum_{d\in D} r_i(d)\, P\, s_j(d) + Q)^\omega,$$

where Q handles an exceptional situation. A standard example in process algebra is the *alternating bit protocol* (see, e.g., [20]), specified as the concurrent execution of four components, each of which can be specified in the style above. Further examples of this specification and verification style can be found in [17, 76].

Table 9 presents two axioms for NEI. (NEI1) is its defining axiom, while (RSP$^\omega$) is an adaptation of (RSP*). The axiomatisations for BPA$^\omega(A)$ and BPA$^\omega_\delta(A)$ are obtained by extending BPA(A) and BPA$_\delta(A)$ with (NEI1) and (RSP$^\omega$).

Table 9: Axioms for NEI.

(NEI1)	$x^\omega = x(x^\omega)$
(RSP$^\omega$)	$\dfrac{x = yx}{x = y^\omega}$

In order to provide process terms over BPA$^\omega_\delta(A)$ with an operational semantics, we introduce transition rules for NEI. Together with the transition rules for BPA(A) in Table 2 they provide labelled transition systems to process terms over BPA$^\omega_\delta(A)$.

Table 10: Transition rules for NEI.

$$\frac{x \xrightarrow{a} x'}{x^\omega \xrightarrow{a} x'(x^\omega)} \qquad \frac{x \xrightarrow{a} \surd}{x^\omega \xrightarrow{a} x^\omega}$$

The transition rules for NEI are in path format. Hence, strong bisimulation equivalence is a congruence with respect to BPA$^\omega_\delta(A)$. Furthermore, its axiomatisation is sound for BPA$^\omega_\delta(A)$ modulo strong bisimulation equivalence. Since strong bisimulation equivalence is a congruence, this can be verified by checking soundness for each axiom separately. It is easily verified that (NEI1) and (RSP$^\omega$) are indeed sound modulo strong bisimulation equivalence.

The following two completeness results for no-exit iteration originate from [40]. Their proofs, which are omitted here, are based on the proof strategy from [39].

Theorem 4.1.1. (A1)–(A5), (NEI1), (RSP$^\omega$) *is complete for* BPA$^\omega(A)$ *modulo strong bisimulation equivalence.*

Theorem 4.1.2. (A1)–(A7), (NEI1), (RSP$^\omega$) *is complete for* BPA$^\omega_\delta(A)$ *modulo strong bisimulation equivalence.*

The observation by Sewell [72] that there does not exist a complete finite equational axiomatisation for BPA$^*_\delta(A)$ modulo strong bisimulation equivalence, is based on the fact that a^ω is strongly bisimilar to $(a^k)^\omega$ for $k \geq 1$. This argument can be copied to conclude that there do not exist complete finite equational axiomatisations for BPA$^\omega(A)$ and BPA$^\omega_\delta(A)$. Hence, the implicational axiom (RSP$^\omega$) is irredundant. It is not difficult to see that axiom (NEI1) is irredundant as well.

4.2 Axiomatisation of Multi-Exit Iteration

Milner [61] noted that not every regular process can be described in BPA$^*(A)$, up to strong bisimulation equivalence. The limited expressive power of BKS was highlighted in [12], where it was shown that the process described by the recursive specification

$$\begin{aligned} X_1 &= aX_2 + a \\ X_2 &= aX_1 + b \end{aligned}$$

cannot be expressed in BPA$^*(A)$ modulo strong bisimulation equivalence. See Section 5 for more information on the expressive power of iterative operators.

Bergstra, Bethke, and Ponse [12] introduced *multi-exit iteration* (MEI) as a more expressive variant of iteration. For every $k \geq 1$, and process terms P_i and Q_i $(1 \leq i \leq k)$, the process term $(P_1, \ldots, P_k)^*(Q_1, \ldots, Q_k)$ denotes a solution to the recursion variable X_1 in the recursive specification

$$\begin{aligned} X_1 &= P_1X_2 + Q_1 \\ &\vdots \\ X_{k-1} &= P_{k-1}X_k + Q_{k-1} \\ X_k &= P_kX_1 + Q_k \end{aligned}$$

Aceto and Fokkink [1] introduced the axiom system BPA$^{me*}(A)$, which is obtained by adding the MEI axioms (MEI1)–(MEI5) in Table 11 to BPA(A). The first three MEI axioms are adaptations of the three BKS axioms. The last two MEI axioms relate process terms of distinct exit degrees. (MEI4) is the multiplicative counterpart of (MEI3), while (MEI5) enables to reduce repetitive patterns at the left- and right-hand side of MEI.

Table 11: Axioms for MEI.

(MEI1)	$x_1((x_2,\ldots,x_k,x_1)^*(y_2,\ldots,y_k,y_1))+y_1 = (x_1,\ldots,x_k)^*(y_1,\ldots,y_k)$
(MEI2)	$((x_1,\ldots,x_k)^*(y_1,\ldots,y_k))z = (x_1,\ldots,x_k)^*(y_1z,\ldots,y_kz)$
(MEI3)	$(z_0,x_2,\ldots,x_k)^*(y_1+z_1((x_2,\ldots,x_k,z_0+z_1)^*(y_2,\ldots,y_k,y_1)),y_2,\ldots,y_k)$
	$= (z_0+z_1,x_2,\ldots,x_k)^*(y_1,\ldots,y_k)$
(MEI4)	$(z_0,z_1,x_2,\ldots,x_k)^*(y_1,z_2((x_2,\ldots,x_k,z_0(z_1+z_2))^*(y_2,\ldots,y_k,y_1)),y_2,\ldots,y_k)$
	$= (z_0(z_1+z_2),x_2,\ldots,x_k)^*(y_1,\ldots,y_k)$
(MEI5)	$((x_1,\ldots,x_k)^\ell)^*((y_1,\ldots,y_k)^\ell) = (x_1,\ldots,x_k)^*(y_1,\ldots,y_k)$

In order to provide process terms over $\mathrm{BPA}^{me*}(A)$ with an operational semantics, we introduce transition rules for MEI. Together with the transition rules for $\mathrm{BPA}(A)$ in Table 2 they provide labelled transition systems to process terms over $\mathrm{BPA}^{me*}(A)$.

Table 12: Transition rules for MEI.

$$\frac{x_1 \xrightarrow{a} x_1'}{(x_1,\ldots,x_k)^*(y_1,\ldots,y_k) \xrightarrow{a} x_1'((x_2,\ldots,x_k,x_1)^*(y_2,\ldots,y_k,y_1))}$$

$$\frac{x_1 \xrightarrow{a} \surd}{(x_1,\ldots,x_k)^*(y_1,\ldots,y_k) \xrightarrow{a} (x_2,\ldots,x_k,x_1)^*(y_2,\ldots,y_k,y_1)}$$

$$\frac{y_1 \xrightarrow{a} y_1'}{(x_1,\ldots,x_k)^*(y_1,\ldots,y_k) \xrightarrow{a} y_1'} \qquad \frac{y_1 \xrightarrow{a} \surd}{(x_1,\ldots,x_k)^*(y_1,\ldots,y_k) \xrightarrow{a} \surd}$$

The transition rules for MEI are in path format. Hence, strong bisimulation equivalence is a congruence with respect to $\mathrm{BPA}^{me*}(A)$. Furthermore, its axiomatisation is sound for $\mathrm{BPA}^{me*}(A)$ modulo strong bisimulation equivalence. Since strong bisimulation equivalence is a congruence, this can be verified by checking soundness for each axiom separately. It is easily verified that (MEI1), (MEI2), and (MEI5) are sound modulo strong bisimulation equivalence. See [1] for a detailed proof that (MEI3) and (MEI4) are sound modulo strong bisimulation equivalence.

In [1] it is proved that the axiomatisation $\mathrm{BPA}^{me*}(A)$ is complete for $\mathrm{BPA}^{me*}(A)$ modulo strong bisimulation equivalence. The completeness proof, which is omitted here, is based on the proof strategy from [39].

Theorem 4.2.1. (A1)–(A5), (MEI1)–(MEI5) *is complete for* $\mathrm{BPA}^{me*}(A)$ *modulo strong bisimulation equivalence.*

4.3 Axiomatisation of Prefix Iteration

Prefix iteration (PI) [37] is a variation of BKS, obtained by restricting its first argument to single atomic actions. The advantage of PI over BKS is twofold:

1. PI can be axiomatised in a setting with *prefix multiplication* of CCS [62], which is obtained from sequential composition by restricting its first argument to single atomic actions;
2. PI allows a complete equational axiomatisation modulo strong bisimulation equivalence in the presence of the deadlock δ.

We note that, in general, sequential composition can be restricted to prefix multiplication without loss of expressivity.

$\mathrm{BPA}^{p*}_{\delta}(A)$ consists of $\mathrm{BPA}(A)$, with sequential composition xy restricted to prefix multiplication ax from CCS, extended with PI. Table 13 presents a collection of axioms for PI. First of all, (PI1)–(PI2) from [37] axiomatise PI with respect to strong bisimulation.

Theorem 4.3.1. (A1)–(A3), (A6), (PI1)–(PI2) *is complete for* $\mathrm{BPA}^{p*}_{\delta}(A)$ *modulo strong bisimulation equivalence.*

The remaining equations and inequalities in Table 13 originate from Aceto, Fokkink, and Ingólfsdóttir [5], who proved completeness results for PI in a variety of behavioural equivalences and preorders in the linear/branching time spectrum [45]. These axiomatisations for $\mathrm{BPA}^{p*}_{\delta}(A)$ all incorporate axioms (A1)–(A3), (A6) for $\mathrm{BPA}_{\delta}(A)$ with prefix multiplication, standard axioms from the literature for $\mathrm{BPA}_{\delta}(A)$ modulo the behavioural equivalence in question, and (PI1)–(PI2) for PI.

(PCT1)–(PCT2) axiomatise PI with respect to completed trace equivalence.

Theorem 4.3.2. (A1)–(A3), (A6), (PI1)–(PI2), (CT1), (PCT1)–(PCT2) *is complete for* $\mathrm{BPA}^{p*}_{\delta}(A)$ *modulo completed trace equivalence.*

(PL1), in cooperation with (PCT1)–(PCT2), axiomatise PI with respect to language equivalence.

Theorem 4.3.3. (A1)–(A3), (A6), (PI1)–(PI2), (CT1), (PCT1)–(PCT2), (L1), (PL1), *is complete for* $\mathrm{BPA}^{p*}_{\delta}(A)$ *modulo language equivalence.*

No extra axioms are needed for PI modulo simulation preorder.

Table 13: Axioms for PI.

(PI1)	$a(a^*x)+x$	$=$	a^*x
(PI2)	$a^*(a^*x)$	$=$	a^*x
(CT1)	$a(x+y)$	$=$	$ax+ay$
(PCT1)	$a^*(x+y)$	$=$	a^*x+a^*y
(PCT2)	$a^*(ax)$	$=$	$a(a^*x)$
(L1)	$a\delta$	$=$	δ
(PL1)	$a^*\delta$	$=$	δ
(S1)	x	$\leq$	$x+y$
(RS1)	ax	$\leq$	$ax+ay$
(PRS1)	a^*x	$\leq$	$a^*(x+ay)$
(R1)	$a(bx+by+v)$	$\leq$	$a(bx+v)+a(by+w)$
(PR1)	$a(a^*(bx+by+v))$	$\leq$	$a(a^*(bx+v))+a(a^*(by+w))$
(PR2)	$a^*(bx+by+v+a(by+w))$	$\leq$	$a^*(bx+v+a(by+w))+by$
(F1)	$a(x+y)$	$\leq$	$ax+a(y+z)$
(PF1)	$a(a^*(x+y))$	$\leq$	$a(a^*x)+a(a^*(y+z))$
(PF2)	$a(a^*x)$	$\leq$	$a^*(a(x+y))$
(PF3)	$a^*(x+y+a(y+z))$	$\leq$	$a^*(x+a(y+z))+y$

Theorem 4.3.4. (A1)–(A3), (A6), (PI1)–(PI2), (S1) *is complete for* $\mathrm{BPA}_\delta^{p*}(A)$ *modulo simulation preorder.*

(PRS1) axiomatises PI with respect to ready simulation preorder.

Theorem 4.3.5. (A1)–(A3), (A6), (PI1)–(PI2), (RS1), (PRS1) *is complete for* $\mathrm{BPA}_\delta^{p*}(A)$ *modulo ready simulation preorder.*

(PR1)–(PR2), in cooperation with (PRS1), axiomatise PI with respect to readies preorder.

Theorem 4.3.6. (A1)–(A3), (A6), (PI1)–(PI2), (RS1), (R1), (PRS1), (PR1)–(PR2) *is complete for* $\mathrm{BPA}_\delta^{p*}(A)$ *modulo readies preorder.*

It is still an open question whether there exists a complete axiomatisation for $\mathrm{BPA}_\delta^{p*}(A)$ modulo failures preorder. Aceto, Fokkink, and Ingólfsdóttir conjectured that (PF1)–(PF3), in cooperation with (PRS1), are sufficient to axiomatise PI with respect to failures preorder.

Conjecture 4.3.7. (A1)–(A3), (A6), (PI1)–(PI2), (F1), (PRS1), (PF1)–(PF3) *is complete for* $\mathrm{BPA}_\delta^{p*}(A)$ *modulo failures preorder.*

To the best of our knowledge, axioms for PI with respect to ready trace preorder and failure trace preorder have not yet been formulated.

4.4 Axiomatisation of String Iteration

String iteration (SI) [7] is a variation of PI, in which the first argument of iteration is allowed to contain non-empty finite strings of atomic actions. Aceto and Groote [7] extended the axiomatisation for PI in strong bisimulation to SI. Like PI, and unlike BKS, SI allows an equational axiomatisation modulo strong bisimulation equivalence in the presence of the deadlock δ.

$\mathrm{BPA}_\delta^{s*}(A)$ consists of $\mathrm{BPA}(A)$, with sequential composition xy restricted to *string multiplication* wy, extended with SI. Table 13 presents two axioms for SI in strong bisimulation. In Table 13, w ranges over the collection A^+ of non-empty strings of atomic actions.

Theorem 4.4.1. (A1)–(A3), (SA5), (A6), (SI1)–(SI4) *is complete for* $\mathrm{BPA}_\delta^{s*}(A)$ *modulo strong bisimulation equivalence.*

Table 14: Axioms for SI.

(SA5)	$w_1(w_2x)$	$=$	$(w_1w_2)x$
(SI1)	$w(w^*x)+x$	$=$	w^*x
(SI2)	$w^*(w^*x)$	$=$	w^*x
(SI3)	$(w^n)^*\delta$	$=$	$w^*\delta$
(SI4)	$a((wa)^*\delta)$	$=$	$(aw)^*\delta$

4.5 Axiomatisation of Flat Iteration

In general, the merge operator cannot be eliminated from process terms with BKS. Therefore Bergstra, Bethke, and Ponse [12] introduced *flat iteration* (FI), which is obtained by restricting the left-hand side of BKS to sums of atomic actions. Similarly, *flat multiplication* is obtained by restricting the left-hand side of sequential composition to sums of atomic actions. The transition rules for these operators are simply the transition rules for BKS and sequential composition with the left-hand sides restricted to sums of atomic actions. $\mathrm{BPA}_\delta^{f*}(A)$ is obtained by adding FI to $\mathrm{BPA}_\delta(A)$ and restricting sequential composition to flat multiplication. The merge can be eliminated from process terms that contain FI. For example, typically, $(a^*b) \parallel (c^*d)$ is strongly bisimilar to

$$(a+c+a \mid c)^*((d+a \mid d)(a^*b)+(b+b \mid c)(c^*d)+b \mid d).$$

For a detailed discussion on this expressivity claim the reader is referred to Section 5.2.

In [48], van Glabbeek presented complete axiomatisations for $\mathrm{BPA}_\delta^{f*}(A)$ extended with the silent step τ, modulo four rooted bisimulation semantics that take into account the silent nature of τ: rooted *branching* bisimulation, rooted *delay* bisimulation, rooted η-bisimulation and rooted *weak* bisimulation. Of these four equivalences, rooted branching bisimulation constitutes the finest relation, while rooted weak bisimulation constitutes the coarsest relation; rooted delay bisimulation and rooted η-bisimulation are incomparable. All four equivalences constitute congruence relations over $\mathrm{BPA}_{\delta\tau}^{f*}(A)$ (in the case of rooted branching bisimulation equivalence this follows from the fact that the transition rules for $\mathrm{BPA}_{\delta\tau}^{f*}(A)$ are in *RBB safe* format [42]). The axiomatisations for FI are adaptations of axiomatisations introduced by Aceto, Fokkink, van Glabbeek, and Ingólfsdóttir [8, 38, 2] for PI.

Table 15 presents adaptations to prefix multiplication of axiom (A4) and of two standard axioms (B1)–(B2) for τ. Furthermore, (FI1) is an adaptation of (BKS3)

to FI, while (FI2) expresses the interplay of FI with the deadlock δ. In the axioms, α and β range over sums of atomic actions (the empty sum representing δ). Note that the defining equation of FI,

$$\beta^* x \;=\; \beta(\beta^* x) + x$$

can be derived from (FI1) by taking α to be δ.

Table 15: Axioms for FI.

(FA4)	$(\alpha+\beta)x$	$=$	$\alpha x+\beta x$
(FB1)	$\alpha\tau$	$=$	α
(FB2)	$\alpha(x+\tau y)$	$=$	$\alpha(x+\tau y)+\alpha y$
(FI1)	$\alpha^*(\beta((\alpha+\beta)^* x)+x)$	$=$	$(\alpha+\beta)^* x$
(FI2)	$\delta^* x$	$=$	x

Table 16 presents axioms for the interplay of FI with the silent step τ, modulo the four aforementioned equivalence relations. (FT1) is an instantiation of *Koomen's fair abstraction rule* (KFAR) [55, 10] for rooted branching and η-bisimulation, while (FT4) serves this same purpose for rooted delay and weak bisimulation. In [48], VAN GLABBEEK proved completeness for $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ with respect to rooted branching bisimulation; see Theorem 4.5.1. The complete axiomatisations for $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ modulo rooted delay, η-, and weak bisimulation equivalence can then be obtained from the complete axiomatisation modulo rooted branching bisimulation equivalence, using a reduction technique from VAN GLABBEEK and WEIJLAND [50]; see Theorems 4.5.2, 4.5.3, and 4.5.4.

Table 16: Axioms for FI with the silent step.

(FT1)	$(\alpha+\tau)^* x$	$=$	$\alpha^* x+\tau(\alpha^* x)$
(FT2)	$\alpha(\beta^*(\tau(\beta^*(x+y))+x))$	$=$	$\alpha(\beta^*(x+y))$
(FT3)	$\alpha^*(x+\tau y)$	$=$	$\alpha^*(x+\tau y+\alpha y)$
(FT4)	$(\alpha+\tau)^* x$	$=$	$\tau(\alpha^* x)$

Theorem 4.5.1. (A1)–(A3), (FA4), (A6)–(A7), (FB1), (FI1)–(FI2), (FT1)–(FT2) *is complete for* $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ *modulo rooted branching bisimulation equivalence.*

Theorem 4.5.2. (A1)–(A3), (FA4), (A6)–(A7), (FB1), (FI1)–(FI2), (FT4) *is complete for* $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ *modulo rooted delay bisimulation equivalence.*

Theorem 4.5.3. (A1)–(A3), (FA4), (A6)–(A7), (FB1)–(FB2), (FI1)–(FI2), (FT1)–(FT3) *is complete for* $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ *modulo rooted η-bisimulation equivalence.*

Theorem 4.5.4. (A1)–(A3), (FA4), (A6)–(A7), (FB1)–(FB2), (FI1)–(FI2), (FT3)–(FT4) *is complete for* $\mathrm{BPA}^{f*}_{\delta\tau}(A)$ *modulo rooted weak bisimulation equivalence.*

5 Expressivity Results

This section concerns expressivity of process algebra with recursive operations, to categorise what can be specified with the various recursive operations. Of course, answers to these questions depend on the particular process semantics one adopts.

5.1 Expressivity of the Binary Kleene Star

In [13], Bergstra, Bethke, and Ponse showed that the expressivity of systems with BKS can be analysed by establishing properties of cycles in labelled transition systems. These results were strengthened by Boselie [23]. We recall these results, and first introduce some further terminology. A state Q is a *successor* of state P if $P \xrightarrow{a} Q$. A *cycle* is a sequence of distinct states $(P_0, \ldots, P_n)$ such that P_{i+1} is a successor of P_i for $i = 0, \ldots, n-1$ and P_0 is a successor of P_n. An action a is an *exit action* of state P if $P \xrightarrow{a} \surd$. We use $\equiv$ to denote that two terms are syntactically the same.

Lemma 5.1.1. *Let C be a cycle in a labelled transition system associated to a process term over* $\mathrm{ACP}^*(A, |, \tau)$. *Then C has one of the following forms, for $n \in \mathbb{N}$:*

(i) $C = (P_0Q, P_1Q, \ldots, P_nQ)$;

(ii) $C = (P^*Q, P_1(P^*Q), \ldots, P_n(P^*Q))$, *or any cyclic permutation thereof;*

(iii) $C = (P_0 \parallel Q_0, P_1 \parallel Q_1, \ldots, P_n \parallel Q_n)$;

(iv) $C = (\partial_H(P_0), \partial_H(P_1), \ldots, \partial_H(P_n))$.

Proof. Let $C = (C_0, \ldots, C_n)$. We apply case distinction on C_0. Clearly C_0 is not a single atomic action, and as $+$, $\between$, $|$ do not occur as the first operation in right-hand sides of conclusions of transition rules, it follows that $C_0 \not\equiv P \diamond Q$ for $\diamond \in \{+, \between, |\}$.

Suppose $C_0 \equiv RS$. If S is not a state in C, then $C = (RS, R_1S, \ldots, R_nS)$, which corresponds to case (i). If S is a state in C, then $S \xrightarrow{\sigma} RS$ for some $\sigma \in A^*$. It is not hard to see that only the first transition rule for BKS can give rise to a transition $T \xrightarrow{a} T'$ where T is a proper subterm of T'. Hence, S is of the form P^*Q, and the first transition in the sequence $S \xrightarrow{\sigma} RS$ is invoked by the first transition rule for BKS. This yields form (ii).

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

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

Suppose $C_0 \equiv \partial_H(R)$. Since only the first transition rule for ∂_H can have been used, it follows that C is of form (iv). □

Lemma 5.1.1 can be used to derive further properties of cycles.

Lemma 5.1.2. *Let C be a cycle in a labelled transition system associated to a process term over $\mathrm{BPA}^*_\delta(A)$. Then there is at most one state P in C that has a successor Q such that P is not a proper substate of Q.*

Proof. As C belongs to a process term over $\mathrm{BPA}^*_\delta(A)$, it must be of the form (i) or (ii) in Lemma 5.1.1. We apply induction with respect to the size of C.

Suppose $C = (P_0Q, \ldots, P_nQ)$. By induction, the cycle $(P_0, \ldots, P_n)$ contains at most one state P_i that has a successor R such that P_i is not a proper substate of R. This implies that P_iQ is the only state in C that may have a successor S such that P_iQ is not a proper substate of S.

Suppose $C = (P^*Q, P_1(P^*Q), \ldots, P_n(P^*Q))$, or any cyclic permutation thereof. Then P^*Q is the only state in C that may have a successor R such that P^*Q is not a proper substate of R. □

Lemma 5.1.3. *Let C be a cycle in a labelled transition system associated to a process term over $\mathrm{PA}^*_\delta(A)$. If there is a state in C with an exit action, then every other state in C has only successors in C.*

Proof. As C belongs to a process term over $\mathrm{PA}^*_\delta(A)$, this cycle must be of the form (i), (ii), or (iii) in Lemma 5.1.1.

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

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

Suppose $C = (P_0 \parallel Q_0, \ldots, P_n \parallel Q_n)$. Since the communication merge is excluded from $\mathrm{PA}^*_\delta(A)$, none of the states in C has an exit action. □

Lemma 5.1.4. *Let C be a cycle in a labelled transition system associated to a process term over* $\mathrm{ACP}^*(A,|)$*. Then there is at most one state in C with an exit action.*

Proof. As C belongs to a process term over $\mathrm{ACP}^*(A,|)$, this cycle must be of the form (i), (ii), (iii), or (iv) in Lemma 5.1.1. We apply induction with respect to the size of C.

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

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

Suppose $C = (P_0 \parallel Q_0, \ldots, P_n \parallel Q_n)$. Assume $P_i \parallel Q_i$ and $P_j \parallel Q_j$ both have an exit action. Then by induction $P_i \parallel Q_i$ represent the same state, so $i = j$.

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

We have the following expressivity hierarchy for process algebra with BKS.

Theorem 5.1.5.

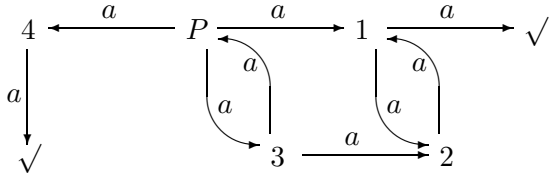
$$\mathrm{BPA}^*_\delta(A) \overset{1}{\prec} \mathrm{PA}^*_\delta(A) \overset{2}{\prec} \mathrm{ACP}^*(A,|) \overset{4}{\prec} \mathrm{ACP}^*(A,|,\tau)$$

where $\overset{k}{\prec}$ means "less expressive than, provided A contains at least k actions" modulo strong bisimulation equivalence, except for the last inequality, which requires the presence of τ and soundness of (B1) *(i.e., $x\tau = x$). If one does not restrict to handshaking,*

$$\mathrm{PA}^*_\delta(A) \overset{1}{\prec} \mathrm{ACP}^*(A,|).$$

The same inclusions hold in the absence of δ.

Proof. $\mathrm{BPA}^*(A) \overset{1}{\prec} \mathrm{PA}^*(A)$ and $\mathrm{BPA}^*_\delta(A) \overset{1}{\prec} \mathrm{PA}^*_\delta(A)$. Consider the $\mathrm{PA}^*(A)$ process term $P = (aa)^*a \parallel a$, which can be depicted as follows:



where

1 abbreviates $(aa)^*a$,
2 abbreviates $a((aa)^*a)$,
3 abbreviates $a((aa)^*a) \parallel a$, and
4 abbreviates a.

According to Lemma 5.1.2, P cannot be specified in $\mathrm{BPA}^*_\delta(A)$. Namely, the states P and 3 are not strongly bisimilar and form a cycle, while both states have a successor from which one cannot return to this (nor to any strongly bisimilar) cycle.

$\mathrm{PA}^*_\delta(A) \overset{i}{\prec} \mathrm{ACP}^*(A,|)$, where $i = 1$ in a setting without handshaking and $i = 2$ otherwise. Take P as in the previous case, and let $a \mid a$ be defined (either as a, thus no handshaking, or as $b \neq a$). Then on top of the picture above, the labelled transition system associated to P contains the following transitions: $P \xrightarrow{a|a} \surd$, $P \xrightarrow{a|a} 2$, and $3 \xrightarrow{a|a} 1$. According to Lemma 5.1.3, P cannot be specified in $\mathrm{PA}^*_\delta(A)$. Namely, the states P and 3 are not strongly bisimilar and form a cycle, while P has an exit action and 3 has a successor from which one cannot return to this (nor to any strongly bisimilar) cycle.

$\mathrm{ACP}^*(A,|) \overset{4}{\prec} \mathrm{ACP}^*(A,|,\tau)$. Take the recursive specification

$$\begin{aligned} X_1 &= aX_2 + a \\ X_2 &= aaX_1 + a. \end{aligned}$$

Assume auxiliary actions b, c, and d, with $c \mid c \overset{\mathrm{def}}{=} b$ and $d \mid d \overset{\mathrm{def}}{=} b$ the only communications defined. Let the process term P be defined by:

$$\begin{aligned} P &= \tau_{\{b\}} \circ \partial_{\{c,d\}}((a(ad+ac)+ac)Q \parallel R) \\ Q &= (c(a(a(ad+ac)+ac)))^*d \\ R &= (dc)^*cd. \end{aligned}$$

It can be derived from the axioms of $\mathrm{ACP}^*(A,|)$ together with (RSP) and (B1) that $P = X_1$. Hence, X_1 is expressible in $\mathrm{ACP}^*(A,|,\tau)$ modulo process semantics that respect (B1). According to Lemma 5.1.4, X_1 cannot be specified in $\mathrm{ACP}^*(A,|,\tau)$ modulo strong bisimulation equivalence, even if τ is allowed to occur in A as a non-silent action. Namely, X_1 and X_2 are not strongly bisimilar and form a cycle, while both X_1 and X_2 have an exit action. □

It is an open question whether $\mathrm{ACP}^*(A,|) \overset{3}{\prec} \mathrm{ACP}^*(A,|,\tau)$ holds.

The following theorem emphasises the expressive power of $\mathrm{ACP}^*(A,|,\tau)$.

Theorem 5.1.6. *For each regular process P there is a finite extension A^{ext} of A such that P can be expressed in $\mathrm{ACP}^*(A^{ext}, |, \tau)$, even if one restricts to handshaking and the actions in A are not subject to communication.*

Proof. P is a solution for the recursion variable X_1 in a recursive specification $X_i = \sum_{j=1}^{n}(\alpha_{i,j}X_j) + \beta_i$ for $i = 1, \ldots, n$, 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 $2n + 3$ fresh atomic actions:

$$in, \quad r_j, s_j \ (j = 0, \ldots, n).$$

Let $r_j \mid s_j \stackrel{\mathrm{def}}{=} in$ (for $j = 0, \ldots, n$) be the only communications defined (so we have handshaking, and the actions in A are not subject to communication). Furthermore, let $H \stackrel{\mathrm{def}}{=} \{r_j, s_j \mid i = 0, \ldots, n\}$, and

$$\begin{array}{lll} G_i & \text{abbreviates} & \sum_{j=1}^{n}(\alpha_{i,j}s_j) + \beta_i s_0 \text{ for } i = 1, \ldots, n, \\ Q & \text{abbreviates} & \sum_{j=1}^{n}(r_j G_j)^* r_0, \\ M & \text{abbreviates} & \sum_{j=1}^{n}(r_j s_j)^*(r_0 s_0). \end{array}$$

We derive

$$\begin{array}{rcl} \partial_H(G_iQ \parallel M) & = & \partial_H(\ (\sum_{j=1}^{n}(\alpha_{i,j}s_jQ) + \beta_i s_0 Q) \parallel M) \\ & = & \sum_{j=1}^{n}(\alpha_{i,j}\partial_H(s_jQ \parallel M)) + \beta_i \partial_H(s_0Q \parallel M) \\ & = & \sum_{j=1}^{n}(\alpha_{i,j}\, in\, \partial_H(Q \parallel s_jM)) + \beta_i\, in\, \partial_H(Q \parallel s_0) \\ & = & \sum_{j=1}^{n}(\alpha_{i,j}\, in\, in\, \partial_H(G_jQ \parallel M)) + \beta_i\, in\, in. \end{array}$$

Hence,

$$\tau_{\{in\}} \circ \partial_H(G_iQ \parallel M) \quad = \quad \sum_{j=1}^{n}(\alpha_{i,j}\tau_{\{in\}} \circ \partial_H(G_jQ \parallel M)) + \beta_i.$$

Consequently, $\tau_{\{in\}} \circ \partial_H(G_iQ \parallel M)$ satisfies the recursive equation for X_i (for $i = 1, \ldots, n$). By (RSP) it follows that $X_1 = \tau_{\{in\}} \circ \partial_H(G_1Q \parallel M)$. □

5.2 Expressivity of Multi-Exit Iteration

We first note that in the extension of $\mathrm{BPA}_\delta(A)$ with multi-exit iteration one cannot describe all regular processes modulo strong bisimulation equivalence. For example, the process described by

$$\begin{array}{rcl} X & = & aY + aZ \\ Y & = & aZ + a \\ Z & = & aX + aa \end{array}$$

cannot be expressed, as from the state X the two non-bisimilar exits a and aa can be reached in a single step.

In [1] it was shown that for every $k \geq 1$ there is a process over a single action that can be specified using $(k+1)$-exit iteration, but not using h-exit iteration with $h \leq k$. We proceed to sketch their argumentation. For $k \geq 1$, $\mathrm{BPA}^{me*(\leq k)}(A)$ denotes the set of process terms over $\mathrm{BPA}^{me*}(A)$ that only use h-exit iteration with $h \leq k$.

The set of *termination options* of a process term P over $\mathrm{BPA}^{me*}(A)$, is the smallest collection of process terms satisfying:

- if $P \xrightarrow{a} \surd$, then a is a termination option of P;
- if $P \xrightarrow{a} Q$ and Q does not contain occurrences of MEI, then aQ is a termination option of P.

Lemma 5.2.1. *Let C be a cycle in a labelled transition system associated to a process term over $\mathrm{BPA}^{me*(\leq k)}(A)$. Then C contains at most k states with distinct, non-empty sets of termination options.*

Proof. Let $C = (C_0, \ldots, C_n)$. We apply structural induction on C_0. Clearly C_0 is not a single atomic action, and as $+$ does not occur as the first operation in right-hand sides of conclusions of transition rules, it follows that C_0 is not of the form $P + Q$.

1. $C_0 \equiv P_0 Q_0$. There are two possibilities.
   (a) Q_0 is not a state in C. Then there is a cycle $(P_0, \ldots, P_n)$ such that $C_i \equiv P_i Q_0$ for $i = 0, \ldots, n$.
   If Q_0 contains occurrences of MEI, then all states in C have an empty set of termination options.
   If Q_0 does not contain occurrences of MEI, then the set of termination options of C_i (for $i = 0, \ldots, n$) is
   $$\{RQ_0 \mid R \text{ is a termination option of } P_i\}.$$
   The inductive hypothesis yields that there are at most k process terms P_i with distinct, non-empty sets of termination options. Hence, there are at most k states in C with distinct, non-empty sets of termination options.
   (b) Q_0 is a state C_l in C. By induction there are at most k states in the cycle $(C_l, \ldots, C_n, C_0, \ldots, C_{l-1})$ with distinct, non-empty sets of termination options. So the same holds for C.
2. $C_0 \equiv (P_1, \ldots, P_h)^*(Q_1, \ldots, Q_h)$, where $h \leq k$.
   Clearly, substates of the Q_i cannot be in C. Thus the only states in C with possibly non-empty sets of termination options are
   $$(P_i, \ldots, P_h, P_1, \ldots, P_{i-1})^*(Q_i, \ldots, Q_h, Q_1, \ldots, Q_{i-1})$$
   for $i = 1, \ldots, h$.

□

Theorem 5.2.2. $\mathrm{BPA}^{me*(\leq k)}(A) \overset{1}{\prec} \mathrm{BPA}^{me*(\leq k+1)}(A)$ *for* $k \geq 1$.

Proof. Let $a \in A$. It suffices to show that the $(k+1)$-exit iteration term

$$(a, a, \ldots, a)^*(a, a^2, \ldots, a^{k+1})$$

cannot be specified in $\mathrm{BPA}^{me*(\leq k)}(A)$ modulo strong bisimulation equivalence. This follows from Lemma 5.2.1, because this process term induces a cycle that traverses the process term

$$(a, \ldots, a)^*(a^i, \ldots, a^{k+1}, a, \ldots, a^{i-1}),$$

which has $\{a^i\}$ as set of termination options, for $i = 1, \ldots, k+1$. Clearly, $a^i \not\leftrightarrows a^j$ if $i \neq j$. □

5.3 Expressivity of String Iteration

In this section it is shown that for every $k \geq 1$ there is a process over a single action that can be specified by SI using a string of length $k+1$, but not by SI using strings of length at most k. For $k \geq 1$, $\mathrm{BPA}^{s*(\leq k)}(A)$ denotes the set of process terms over $\mathrm{BPA}^{s*}(A)$ that only use strings of length at most k.

Lemma 5.3.1. *Let C be a cycle in a labelled transition system associated to a process term over* $\mathrm{BPA}^{s*(\leq k)}(A)$. *Then C contains at most k distinct states.*

Proof. Let $C = (C_0, \ldots, C_n)$. We apply structural induction on C_0. Clearly C_0 is not a single atomic action, and as $+$ does not occur as the first operation in right-hand sides of conclusions of transition rules, it follows that C_0 is not of the form $P + Q$.

1. $C_0 \equiv wP$.

   Clearly, P is a state C_l in C. By induction there are at most k distinct states in the cycle $(C_l, \ldots, C_n, C_0, \ldots, C_{l-1})$. So the same holds for C.

2. $C_0 \equiv (a_1 \cdots a_h)^* P$, where $h \leq k$.

   Clearly, substates of P cannot be in C. Thus the only distinct states in C are

   $$(a_{i+1} \cdots a_h)((a_1 \cdots a_h)^* P) \qquad (\text{for } i = 1, \ldots, h).$$

□

Theorem 5.3.2. $\mathrm{BPA}^{s*(\leq k)}(A) \overset{1}{\prec} \mathrm{BPA}^{s*(\leq k+1)}(A)$ *for* $k \geq 1$.

Proof. Let $a \in A$. It suffices to show that the $(k+1)$-string iteration term

$$(a^{k+1})^* a$$

cannot be specified by a process term over $\mathrm{BPA}^{s*(\leq k)}(A)$ modulo strong bisimulation equivalence. This follows from Lemma 5.3.1, because the process term above induces a cycle that traverses the $k+1$ non-bisimilar process terms $a^i((a^{k+1})^* a)$ for $i = 0, \ldots, k$. □

5.4 Expressivity of Flat Iteration

This section presents some expressivity results on FI from [12]. $\mathrm{BPA}^{f*}(A)$, $\mathrm{BPA}_\delta^{f*}(A)$, $\mathrm{PA}^{f*}(A)$, and $\mathrm{ACP}^{f*}(A, |)$ are obtained by adding FI to $\mathrm{BPA}(A)$, $\mathrm{BPA}_\delta(A)$, $\mathrm{PA}(A)$, and $\mathrm{ACP}(A, |)$, respectively, and restricting sequential composition to flat multiplication.

As stated below, restricting sequential composition to prefix multiplication gives no loss of expressivity. *Flat iterative basic terms* over $\mathrm{BPA}^{f*}(A)$ are defined by the BNF grammar

$$P \quad ::= \quad a \mid P + P \mid aP \mid \alpha^* P$$

where $a \in A$ and α is an atomic sum. Flat iterative basic terms over $\mathrm{BPA}_\delta^{f*}(A)$ are defined by adding δ to the BNF grammar.

Lemma 5.4.1. *Each process term over* $\mathrm{BPA}^*(A)$ $[\mathrm{BPA}_\delta^*(A)]$ *with BKS restricted to FI is bisimilar to a flat iterative basic term over* $\mathrm{BPA}^{f*}(A)$ $[\mathrm{BPA}_\delta^{f*}(A)]$.

Proof. By structural induction, using the axioms of $\mathrm{BPA}_\delta(A)$ and those in Table 15. □

With respect to expressivity of systems with FI in strong bisimulation semantics we have the following results.

Theorem 5.4.2.

1. $\mathrm{BPA}^{f*}(A) \overset{1}{\prec} \mathrm{BPA}^*(A)$ *and* $\mathrm{BPA}_\delta^{f*}(A) \overset{1}{\prec} \mathrm{BPA}_\delta^*(A)$,
2. $\mathrm{BPA}^{f*}(A)$ *is as expressive as* $\mathrm{PA}^{f*}(A)$, *and*
3. $\mathrm{BPA}_\delta^{f*}(A)$ *is as expressive as* $\mathrm{ACP}^{f*}(A, |)$.

Proof. Fact 1 is trivially true, as FI does not give rise to cycles of length greater than one. For example, the process term $(aa)^*a$ over $\mathrm{BPA}^*(A)$, which has a cycle of length two, cannot be expressed in $\mathrm{BPA}_\delta^{f*}(A)$ modulo strong bisimulation equivalence.

We proceed to present the proof of fact 2. Fact 3 can be proved in a similar fashion.

From Lemma 5.4.1 it follows that $\mathrm{BPA}^{f*}(A)$ is as expressive as $\mathrm{PA}^{f*}(A)$ if all process terms $P \parallel Q$ and $P \between Q$ with P and Q flat iterative basic terms are expressible in $\mathrm{BPA}^{f*}(A)$. Expressibility of $P \parallel Q$ and $P \between Q$ in $\mathrm{BPA}^{f*}(A)$ can be proved in parallel, using induction on the size of such terms. We focus on the case $P \parallel Q$; the case $P \between Q$ can be dealt with in a similar fashion. We consider three cases, depending on whether P and Q are of the form α^*R.

1. Let $P \equiv \alpha^*R$ and $Q \equiv \beta^*S$. Then we derive (using commutativity of $\parallel$ in $\mathrm{PA}^*(A)$)

   $$P \parallel Q \;=\; (\alpha+\beta)(P \parallel Q) + R \between Q + S \between P$$

   The process terms $R \between Q$ and $S \between P$ have sizes smaller than $P \parallel Q$, so by induction they can be expressed in $\mathrm{BPA}^{f*}(A)$, say by U and V, respectively. By (RSP^*),

   $$P \parallel Q \;=\; (\alpha+\beta)^*(U+V)$$

   so $P \parallel Q$ is expressible in $\mathrm{BPA}^{f*}(A)$.

2. Let $P =_{\mathrm{AC}} \sum_i \alpha_i^* R_i + \sum_j a_j S_j + \sum_k b_k$ (with P not of the form α^*R) and $Q \equiv \beta^*T$. Then we derive

   $$\begin{aligned} P \parallel Q \;&=\; \beta(P \parallel Q) + T \between P + \textstyle\sum_i \alpha_i((\alpha_i^* R_i) \parallel Q) + \sum_j a_j(S_j \parallel Q) \\ &\quad + \textstyle\sum_k b_k Q. \end{aligned}$$

   The process terms $T \between P$, $(\alpha_i^* R_i) \parallel Q$, and $S_j \parallel Q$ have sizes smaller than $P \parallel Q$, so by induction they can be expressed in $\mathrm{BPA}^{f*}(A)$, say by U, V_i, and W_j, respectively. By (RSP^*),

   $$P \parallel Q \;=\; \beta^*(U + \textstyle\sum_i \alpha_i V_i + \sum_j a_j W_j + \sum_k b_k Q)$$

   so $P \parallel Q$ is expressible in $\mathrm{BPA}^{f*}(A)$.

3. Let $P =_{\mathrm{AC}} \sum_i \alpha_i^* R_i + \sum_j a_j S_j + \sum_k b_k$ (with P not of the form α^*R) and $Q =_{\mathrm{AC}} \sum_\ell \beta_\ell^* T_\ell + \sum_m c_m U_m + \sum_n d_n$ (with Q not of the form β^*T). Then we derive

   $$\begin{aligned} P \parallel Q \;&=\; \textstyle\sum_i \alpha_i((\alpha_i^* R_i) \parallel Q) + \sum_j a_j(S_j \parallel Q) + \sum_k b_k Q \\ &\quad + \textstyle\sum_\ell \beta_\ell((\beta_\ell^* T_\ell) \parallel P) + \sum_m c_m(U_m \parallel P) + \sum_n d_n P. \end{aligned}$$

The process terms $(\alpha_i^* R_i) \parallel Q$, $S_j \parallel Q$, $(\beta_\ell^* T_\ell) \parallel P$, and $U_m \parallel P$ have sizes smaller than $P \parallel Q$, so by induction they can be expressed in $\mathrm{BPA}^{f*}(A)$. Hence, $P \parallel Q$ is expressible in $\mathrm{BPA}^{f*}(A)$.

Owing to commutativity of the merge, the three cases above cover all possible forms of $P \parallel Q$. So we conclude that $P \parallel Q$ is expressible in $\mathrm{BPA}^{f*}(A)$. □

Note that parts in general process terms over $\mathrm{BPA}^{f*}(A)$ [$\mathrm{BPA}_\delta^{f*}(A)$] cannot be *equated* to process terms over $\mathrm{PA}^{f*}(A)$ [$\mathrm{ACP}^{f*}(A,|)$]. If one of the arguments of $\parallel$ specifies a cycle, this occurrence of $\parallel$ cannot be eliminated with the axioms provided.

6 Non-Regular Recursive Operations

With each of the recursive operations discussed before, one can define at most a regular process. In this section we consider some operations with which non-regular processes can be described. A typical example of a non-regular process is a stack over a finite data type. In [13], Bergstra, Bethke, and Ponse introduced the recursive, non-regular *nesting* operation $\sharp$, defined by

(NE) $\quad x^\sharp y = x((x^\sharp y)x) + y.$

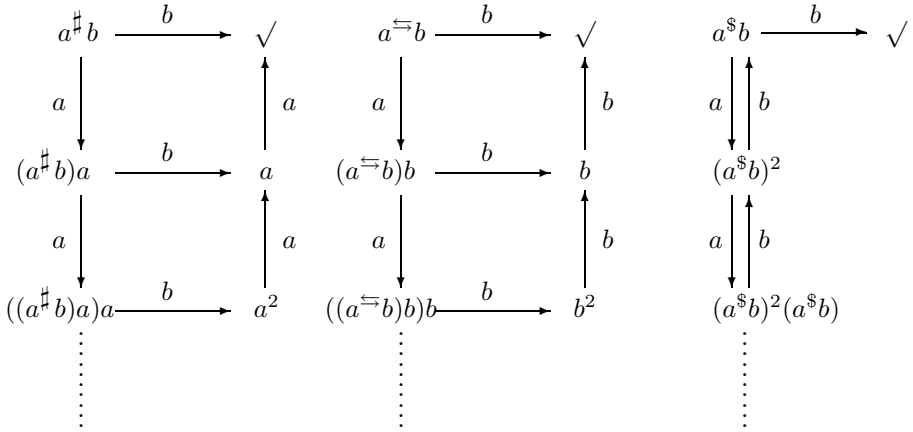
More recently, in [25, 26], Bergstra and Ponse introduced two other non-regular, recursive operations, the *back and forth* operation, notation $\leftrightarrows$, defined by

(BF) $\quad x^\leftrightarrows y = x((x^\leftrightarrows y)y) + y,$

and the *push-down* operation, notation \$, defined by

(PD) $\quad x^\$ y = x((x^\$ y)(x^\$ y)) + y.$

The transition rules for these operations are as expected. As an example, consider the process terms $a^\sharp b$, $a^\leftrightarrows b$, and $a^\$ b$, of which the labelled transition systems are illustrated below:



The three operations give rise to variants of RSP*:

$$(\mathrm{RSP}^{\sharp}) \quad \frac{x = y(xy) + z}{x = y^{\sharp} z}$$

$$(\mathrm{RSP}^{\leftrightarrows}) \quad \frac{x = y(xz) + z}{x = y^{\leftrightarrows} z}$$

$$(\mathrm{RSP}^{\$}) \quad \frac{x = y(xx) + z}{x = y^{\$} z}$$

It is easily seen that these three operations are non-regular, and it can be argued that they are the most simple candidates for obtaining a binary, non-regular recursive operation. Let $\diamond \in \{\sharp, \leftrightarrows, \$\}$. Adding $\diamond$ to the signature of $\mathrm{ACP}(A, |)$, and its defining axiom to those of $\mathrm{ACP}(A, |)$, yields the system $\mathrm{ACP}^{\diamond}(A, |)$. In the same way, we define $\mathrm{ACP}^{*\diamond}(A, |)$ as the extension of $\mathrm{ACP}^*(A, |)$ with $\diamond$. It is an open question whether the resulting axiomatisations with the corresponding RSP variant are complete modulo strong bisimulation equivalence.

In [25, 26], the following results were recorded.

- Adding abstraction to $\mathrm{ACP}^{\diamond}(A, |)$ with $\diamond \in \{\sharp, \leftrightarrows, \$\}$ and A sufficiently large yields universal expressivity modulo process semantics that respect (B1).[4] (Note that BKS need not be available.)
- For $\diamond \in \{\sharp, \leftrightarrows, \$\}$ and A sufficiently large, $\mathrm{ACP}^{*\diamond}(A, |)$ has an undecidable theory. The point is that one can encode register machine computability in a systematic way, and reduce recursive inseparability to provable equality in the initial algebra of $\mathrm{ACP}^{*\diamond}(A, |)$.

In [13], it was proved that a stack over a finite data type can be defined with the operations of ACP with abstraction and handshaking communication, with the help of a finite number of auxiliary actions and of the operations $*$ and $\sharp$. With two stacks and a regular control process, a *Turing machine* can be specified in process algebra; see [10]. As a consequence, each computable process can be specified in this setting. Bergstra and Ponse [25, 26] proved that adding only one of $\sharp$, \$, or $\leftrightarrows$ to $\mathrm{ACP}(A, |, \tau)$ yields a setting in which regular processes and stacks can be defined, and therefore each computable process. In this section we sketch the argumentation for the \$-case of these results. This case is more simple and direct than the other two cases.

In the forthcoming expressiveness proofs, strong bisimilarity of process terms is derived from the axioms. For clarity of presentation, in these derivations we assume

[4] In the case of rooted weak bisimulation semantics, the resulting theory can be judged expressively complete, as all semi-computable processes that initially are finitely branching can be expressed; see [26].

the presence of axioms for commutativity and associativity of the merge. However, the axiomatic derivability of the expressiveness results can also be obtained without these axioms.

6.1 Process Algebra with a Push-Down Operation

We first show that each regular process can be specified in $\mathrm{ACP}^{\$}(A, |, \tau)$ modulo process semantics that respect (B1), provided A is sufficiently large. This is the first cornerstone of the universal expressivity result for $\mathrm{ACP}^{\$}(A, |, \tau)$.

Theorem 6.1.1. *For each regular process P there is a finite extension A^{ext} of A such that P can be expressed in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$, even if one restricts to handshaking and the actions in A are not subject to communication.*

Proof. P is a solution for the recursion variable X_1 in a recursive specification $X_i = \sum_{j=1}^{n}(\alpha_{i,j}X_j) + \beta_i$ for $i = 1, \ldots, n$, 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 $2n + 3$ fresh atomic actions:

$$in, \quad r_j, s_j \ (j = 0, \ldots, n).$$

Let $r_j \mid s_j \stackrel{\mathrm{def}}{=} in$ (for $j = 0, \ldots, n$) be the only communications defined (so we have handshaking, and the actions in A are not subject to communication). Furthermore, let $H \stackrel{\mathrm{def}}{=} \{r_j, s_j \mid i = 0, \ldots, n\}$, and

$$\begin{array}{lll} F_i & \text{abbreviates} & (\sum_{j=1}^{n} \alpha_{i,j}s_j) + \beta_i \text{ for } i = 1, \ldots, n, \\ K & \text{abbreviates} & (\sum_{j=1}^{n} r_j F_j)^{\$} r_0, \\ M & \text{abbreviates} & (\sum_{j=1}^{n} r_j s_j)^{\$} s_0. \end{array}$$

Then $X_1 = \tau_{\{in\}} \circ \partial_H(F_1 K \parallel M)$. This can be shown with the help of the infinite recursive specification

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

Obviously, X_i is a solution for each $Y_i(k)$ $(i = 1, \ldots, n,\ k \in \mathbb{N})$. So by (RSP) it suffices to show that $\tau_{\{in\}} \circ \partial_H(F_i K \parallel M)$ is a solution for $Y_i(0)$. We show this by first omitting the $\tau_{\{in\}}$-application. For $k \in \mathbb{N}$ we derive

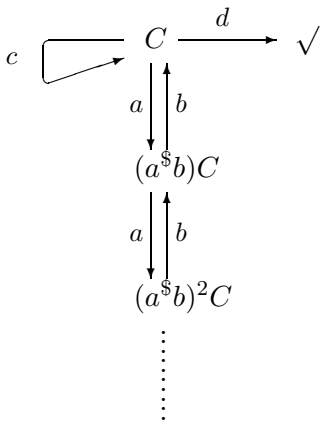
$$\begin{array}{rcl} \partial_H(F_i K^{k+1} \parallel M^{k+1}) & = & (\sum_{j=1}^{n} \alpha_{i,j} \partial_H(s_j K^{k+1} \parallel M^{k+1})) \\ & & + \beta_i \partial_H(K^{k+1} \parallel M^{k+1}) \\ & = & (\sum_{j=1}^{n} \alpha_{i,j}\, in\, \partial_H(K^{k+1} \parallel s_j M^{k+2})) + \beta_i\, in^{k+1} \\ & = & (\sum_{j=1}^{n} \alpha_{i,j}\, in\, in\, \partial_H(F_j K^{k+2} \parallel M^{k+2})) + \beta_i\, in^{k+1}. \end{array}$$

Hence, applying axioms (B1) and (TI1)–(TI4), we find for each $k \in \mathbb{N}$

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

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

A basic, auxiliary process used in the following proofs is the counter C displayed below, with actions a (add one), b (subtract one), c (test zero), and d (remove the counter):



This process can be recognised as a register, i.e., a memory location for a natural number with unbounded capacity and restricted access as modelled by the specific actions. Using BKS and push-down, the counter C can be defined by

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

The following result states that C can be defined without BKS, at cost of five auxiliary actions. In the next section we shall define a stack using two counters and a regular control process.

Lemma 6.1.2. *Let $A \stackrel{\text{def}}{=} \{a, b, c, d\}$. The counter $(a(a^{\$}b) + c)^* d$ can be defined in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$ with $|A^{ext} \setminus A| = 5$, even if one restricts to handshaking and the actions in A are not subject to communication.*

Proof. Define A^{ext} as the extension of A with the following five fresh atomic actions:

$$in, \quad r_j, s_j \ (j = 0, 1).$$

Let $r_j \mid s_j \stackrel{\text{def}}{=} in$ (for $j = 0, 1$) be the only communications defined (so we have handshaking, and the actions in A are not subject to communication). Furthermore,

$$\begin{array}{lll} P & \text{abbreviates} & (a(a^{\$}b) + c)s_1 + d, \\ Q & \text{abbreviates} & (r_1 P)^{\$} r_0, \\ R & \text{abbreviates} & (r_1 s_1)^{\$} s_0. \end{array}$$

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

6.2 Expressing a Stack

We provide recursive specifications of a stack over a finite data type in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$, with the help of a regular control process and two counters. Let $D = \{\mathtt{d}_1, \ldots, \mathtt{d}_N\}$ for some $N \geq 1$ be a finite set of data elements, ranged over by $\mathtt{d}$. Let furthermore D^* be the set of finite strings over D, ranged over by σ, and let ϵ denote the empty string. The stack $S(\epsilon)$ over D with empty-testing and termination option is defined by the infinite recursive specification

$$\begin{array}{rcl} S(\epsilon) & = & (\sum_{j=1}^{N} r(\mathtt{d}_j) S(\mathtt{d}_j)) + s(\mathit{empty}) S(\epsilon) + r(\mathit{stop}) \\ S(\mathtt{d}\sigma) & = & (\sum_{j=1}^{N} r(\mathtt{d}_j) S(\mathtt{d}_j \mathtt{d}\sigma)) + s(\mathtt{d}) S(\sigma). \end{array}$$

Here the contents of the stack is represented by the argument of S: $S(\mathtt{d}\sigma)$ is the stack that contains $\mathtt{d}\sigma$ with $\mathtt{d}$ on top. Action $r(\mathtt{d}_i)$ (receive $\mathtt{d}_i$) models the push of $\mathtt{d}_i$ onto the stack, and action $s(\mathtt{d}_i)$ (send $\mathtt{d}_i$) represents deletion of $\mathtt{d}_i$ 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 summand. In case $N = 1$ ($D = \{\mathtt{d}_1\}$), the recursive equations above specify a counter: the stack contents then models the counter value.

The following theorem is the second (and last) cornerstone of the universal expressivity result for $\mathrm{ACP}^{\$}(A, |, \tau)$.

Theorem 6.2.1. *Each stack over a finite data type D with actions from A can be expressed in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$ with A^{ext} a finite extension of A, even if one restricts to handshaking and the actions in A are not subject to communication.*

Proof. Let a stack $S(\epsilon)$ be given as described above. Without loss of generality, assume $D = \{\mathtt{d}_1, \ldots, \mathtt{d}_N\}$ for some $N > 1$ (if $N = 1$, then 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 $\ulcorner \cdot \urcorner : D^* \to \mathbb{N}$:

$$\begin{array}{lcl} \ulcorner \epsilon \urcorner & \stackrel{\text{def}}{=} & 0, \\ \ulcorner \mathtt{d}_j \sigma \urcorner & \stackrel{\text{def}}{=} & j + N \cdot \ulcorner \sigma \urcorner. \end{array}$$

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

$$decode(n) \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{if } n = 0, \\ \mathtt{d}_N \star decode(\frac{n-N}{N}) & \text{if } n \neq 0,\ n \bmod N = 0, \\ \mathtt{d}_{(n \bmod N)} \star decode(\frac{n-(n \bmod N)}{N}) & \text{otherwise.} \end{cases}$$

For example, if $N = 3$, then $\ulcorner \mathtt{d}_3\mathtt{d}_1\mathtt{d}_2 \urcorner = 24$ and $decode(32) = \mathtt{d}_2\mathtt{d}_1\mathtt{d}_3 \in \{\mathtt{d}_1, \mathtt{d}_2, \mathtt{d}_3\}^*$.

Next, we define two counters to specify $S(\epsilon)$ in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$:

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

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

$$\begin{aligned} C_j(0) &\stackrel{\text{def}}{=} C_j, \\ C_j(n+1) &\stackrel{\text{def}}{=} (\overline{a}_j{}^{\$}\overline{b}_j)C_j(n). \end{aligned}$$

We further define a regular control process X_ϵ with actions $a_j, b_j, c_j, d_j \in A^{ext} \setminus A$ 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 recursive specification.

$$X_\epsilon = (\textstyle\sum_{j=1}^{N} r(\mathtt{d}_j)a_1^j X_j) + s(empty)X_\epsilon + r(stop)d_1d_2,$$

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

$$\begin{aligned} X_k &= (\textstyle\sum_{j=1}^{N} r(\mathtt{d}_j)\mathrm{Push}_j) + s(\mathtt{d}_k)\mathrm{Pop}_k \\ \mathrm{Push}_k &= (\mathrm{Shift\,1\,to\,2})a_1^k(N\,\mathrm{Shift\,2\,to\,1})X_k \\ \mathrm{Pop}_k &= b_1^k(\tfrac{1}{N}\,\mathrm{Shift\,1\,to\,2})\mathrm{Test}_\epsilon \\ \mathrm{Shift\,1\,to\,2} &= (b_1a_2)^*c_1 && \text{(shift the contents of } C_1 \text{ to } C_2) \\ N\,\mathrm{Shift\,2\,to\,1} &= (b_2a_1^N)^*c_2 && \text{(shift the } N\text{-fold of } C_2 \text{ to } C_1) \\ \tfrac{1}{N}\,\mathrm{Shift\,1\,to\,2} &= (b_1^Na_2)^*c_1 && \text{(shift the number of } N\text{-folds of } C_1 \text{ to } C_2) \\ \mathrm{Test}_\epsilon &= b_2a_1\mathrm{Test}_1 + c_2X_\epsilon && \text{(determine whether the stack is empty,} \\ \mathrm{Test}_1 &= b_2a_1\mathrm{Test}_2 + c_2X_1 && \text{ or which } D\text{-element is on top)} \\ \mathrm{Test}_2 &= b_2a_1\mathrm{Test}_3 + c_2X_2 \\ &\vdots \\ \mathrm{Test}_N &= b_2a_1\mathrm{Test}_1 + c_2X_N. \end{aligned}$$

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

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

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

$$\begin{aligned}
&\tau_{\{in\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)) \\
&= (\textstyle\sum_{j=1}^{N} r(\mathtt{d}_j)\tau_{\{in\}} \circ \partial_H(a_1^j X_j \parallel C_1(0) \parallel C_2(0))) \\
&\quad + s(empty)\tau_{\{in\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)) \\
&\quad + r(stop)\tau_{\{in\}} \circ \partial_H(d_1 d_2 \parallel C_1(0) \parallel C_2(0)) \\
&= (\textstyle\sum_{j=1}^{N} r(\mathtt{d}_j)\tau^{j+1}\tau_{\{in\}} \circ \partial_H(X_j \parallel C_1(j) \parallel C_2(0))) \\
&\quad + s(empty)\tau_{\{in\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)) \\
&\quad + r(stop)\tau\tau \\
&= (\textstyle\sum_{j=1}^{N} r(\mathtt{d}_j)\tau_{\{in\}} \circ \partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j \urcorner) \parallel C_2(0))) \\
&\quad + s(empty)\tau_{\{in\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)) + r(stop). \qquad (3)
\end{aligned}$$

We are done if $\tau_{\{in\}} \circ \partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0))$ behaves as $S(\mathtt{d}_j\sigma)$ for some $\sigma \in D^*$. We prove this by first omitting the $\tau_{\{in\}}$-operation, and analysing the behaviour of $\partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0))$. This analysis is arranged in a graphical style in Fig. 1, where $P \xrightarrow{a} Q$ represents the statement $P = aQ$ for some $a \in A$, $P \xrightarrow{\sigma} \!\!\!\!\rightarrow Q$ represents $P = \sigma Q$, and branching represents an application of $+$. So the uppermost expression in Fig. 1 with its arrows and resulting expressions represents the obviously derivable equation

$$\begin{aligned}
\partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0)) \;=\; &(\textstyle\sum_{k=1}^{N} r(\mathtt{d}_k)\partial_H(\mathrm{Push}_k \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0))) \\
&+ s(\mathtt{d}_j)\partial_H(\mathrm{Pop}_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0)).
\end{aligned}$$

By the axiom (B1), identity (3) above, and the derivation displayed in Fig. 1 it follows that

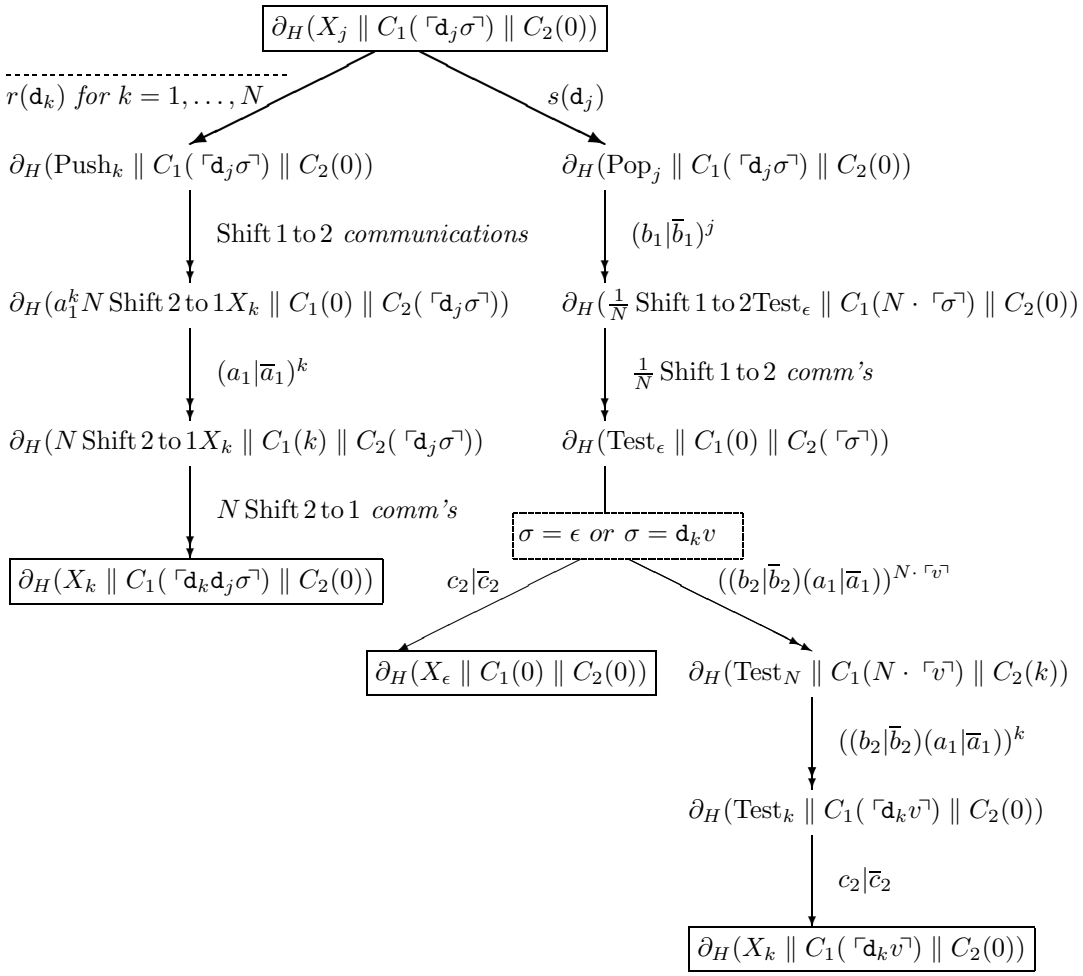
$$\tau_{\{in\}} \circ \partial_H(X_\epsilon \parallel C_1(0) \parallel C_2(0)), \text{ and } \tau_{\{in\}} \circ \partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0))$$

satisfy the recursive equations for $S(\epsilon)$ and $S(\mathtt{d}_j\sigma)$, respectively ($j = 1, \ldots, N$ and $\sigma \in D^*$). By (RSP) it follows that

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

By Theorem 6.1.1 and Lemma 6.1.2 it follows that once D is fixed, X_ϵ and hence the empty stack $S(\epsilon)$ can be expressed in $\mathrm{ACP}^{\$}(A^{ext}, |, \tau)$ with handshaking for some $A^{ext} \supseteq A$. □

Figure 1: Calculations with $\partial_H(X_j \parallel C_1(\ulcorner \mathtt{d}_j\sigma \urcorner) \parallel C_2(0))$.



Baeten, Bergstra, and Klop [10] showed that Turing machines can be specified in process algebra by means of two stacks and a regular control process. In view of Theorem 6.2.1, this yields that $\mathrm{ACP}^{\$}(A,|,\tau)$ is universally expressive; see [26] for details.

6.3 Undecidability Results

We now sketch the undecidability result mentioned above for $\mathrm{ACP}^{*\$}(A,|)$. The idea is that in this signature one can 'implement' register machine computability in the following way.

1. Registers (counters) have a straightforward definition in $\mathrm{ACP}^{*\$}(A,|)$, namely $(a(a^{\$}b)+c)^*d$ (cf. Section 6.1).

2. Starting from a universal programming language for two-register machines (cf. Minsky in [64]), one can define a process algebraic representation of each program in $\mathrm{BPA}^*(A)$ (using a third register for I/O, and a fourth one as "program-line counter").

3. Defining encapsulation in an appropriate way, this yields for any computable function $f:\mathbb{N}\to\mathbb{N}$ a process term P and a computable function $g:\mathbb{N}\to\mathbb{N}\backslash\{0\}$ such that the equation

$$\partial_H(Px \parallel C_0(n) \parallel C_1 \parallel C_2 \parallel C_3) \;=\; in^{g(n)}\partial_H(x \parallel C_0(f(n)) \parallel C_1 \parallel C_2 \parallel C_3)$$

   can be derived from the axioms for $\mathrm{ACP}^{*\$}(A,|)$ if and only if $f(n)$ is defined, and the left-hand side equals an infinite *in*-trace otherwise. Here *in* is the result of a communication between the program (process term) P and the registers.

Now let W_{e_1}, W_{e_2} be recursively inseparable sets, and let $f:\mathbb{N}\to\mathbb{N}$ be the partial recursive function defined by

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

Choose P as described in item 3 above, and let P_1', P_2' and $P_1(n), P_2(n)$ be defined by

$$\begin{aligned} P_1' &= P({s_3}^*d_3)({s_2}^*d_2)({s_1}^*d_1)({s_0}^*d_0),\\ P_2' &= P({s_3}^*d_3)({s_2}^*d_2)({s_1}^*d_1)(({s_0}({s_0}^*c_0))^*d_0),\\ P_i(n) &= \partial_H(P_i' \parallel C_0(n) \parallel C_1 \parallel C_2 \parallel C_3) \qquad (i=1,2).\end{aligned}$$

Then we find

$$n \in W_{e_1} \;\Rightarrow\; f(n) = 0 \;\Rightarrow\; P_1(n) = P_2(n) \qquad (= in^{g(n)+4}),$$
$$n \in W_{e_2} \;\Rightarrow\; f(n) = 1 \;\Rightarrow\; P_1(n) \neq P_2(n)\ (in^{g(n)+5} \neq in^{g(n)+6}).$$

As to the latter implication: assume otherwise, i.e., $in^k = in^{k+1}$ for some $k \geq 1$. Then by Lemma 2.2.2, $in^k \leftrightarrows in^{k+1}$, which clearly is a contradiction.

Thus, decidability of $P_1(n) = P_2(n)$ provides a recursive separation of W_{e_1} and W_{e_2}, which is contradictory. All details and a more precise explanation can be found in [26]. A similar proof strategy can be applied for $\mathrm{ACP}^{*\sharp}(A, |)$ and $\mathrm{ACP}^{*\leftrightarrows}(A, |)$, where counter-like processes are used instead.

7 Special Constants

This section provides some last comments on two particular constants. First we shortly consider the silent step τ in relation to *fairness*, dealing with infinite τ-traces (cf. Definition 2.3.3 and Theorems 4.5.1–4.5.4). Finally, we briefly discuss the *empty process* in the context of iteration.

7.1 Silent Step and Fairness

Due to the character of τ, one would want to be able to abstract from infinite sequences of τ steps. Depending on the kind of process semantics adopted, different solutions have been found. In the case of rooted branching bisimulation, with next to (B1) the extra axiom

$$\text{(B2)} \quad x(\tau(y+z)+y) \;=\; x(y+z)$$

a general solution is provided by *Koomen's fair abstraction rule* [55, 10]. For each n and each set of equations, there is a version KFAR_n^b that is valid in rooted branching bisimulation. For example, the axiom KFAR_1^b reads as follows:

$$\frac{x = ix + y \quad (i \in I)}{\tau\tau_I(x) = \tau\tau_I(y)}$$

(so the infinite τ sequence induced by ix is reduced to a single τ step). By definition of BKS we now have an immediate representation of the process in the premise of KFAR_1^b, namely i^*y. Henceforth we can represent KFAR_1^b by the law

$$\tau\tau_I(i^*y) = \tau\tau_I(y) \qquad (i \in I).$$

Given the distribution law

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

(see Table 8), we can even represent KFAR_1^b simply by

$$(\mathrm{FIR}_1^b) \quad \tau\tau^* x = \tau x.$$

(taking x for $\tau_I(y)$), where FIR abbreviates Fair Iteration Rule.

Example 7.1.1. *A particular consequence of FIR_1^b is the case where x as above is replaced by τx:*

$$\tau^*(\tau x) = \tau x, \tag{4}$$

the proof of which is trivial: $\tau^*(\tau x) = \tau(\tau^*(\tau x)) + \tau x \overset{FIR_1^b}{=} \tau\tau x + \tau x = \tau x$.

As a small example of the use of FIR_1^b consider a statistic experiment which models the tossing of a coin until head comes up (cf. [28]). This process can be described by:

$$(\mathtt{throw}\ \mathtt{tail})^*\mathtt{throw}\ \mathtt{head}.$$

for actions `throw`*,* `tail`*, and* `head`*. We assume that the probability of tossing heads is larger than 0. Thus we exclude the infinite trace that alternately executes* `throw` *and* `tail`*. Abstracting from just the two atomic actions in $I \overset{\mathrm{def}}{=} \{\mathtt{throw}, \mathtt{tail}\}$, FIR_1^b yields*

$$\tau_I((\mathtt{throw}\ \mathtt{tail})^*\mathtt{throw}\ \mathtt{head}) = \tau\,\mathtt{head}.$$

First, observe $\tau_I(\mathtt{throw}\ \mathtt{tail}) = \tau$. Then, using (4), it easily follows that

$$\tau_I((\mathtt{throw}\ \mathtt{tail})^*\mathtt{throw}\ \mathtt{head}) = \tau\,\mathtt{head}.$$

This expresses that `head` *eventually comes up, and thus excludes the infinite sequence of τ-steps present in $\tau_I((\mathtt{throw}\ \mathtt{tail})^*\mathtt{throw}\ \mathtt{head})$.*

7.2 Empty Process

Let the symbol ε denote the *empty process*, introduced as a unit for sequential composition by Koymans and Vrancken in [58] (see also [28, 74]). Obvious as ε may be (being a unit for $\cdot$), its introduction is nontrivial because at the same time it must be a unit for $\|$ as well. In the design of BPA, PA, ACP and related axiom systems, it has proved useful to study versions of the theory, both with and without ε. Just for this reason the star operation with its (original) defining equation as given by Kleene in [54] was introduced in process algebra.

Taking $y = \varepsilon$ in $x^* y$, one obtains $x^*\varepsilon$ which satisfies

$$x^*\varepsilon = x(x^*\varepsilon) + \varepsilon. \tag{5}$$

The unary operation $\_^*\varepsilon$ is a plausible candidate for the unary version of Kleene's star operation in process algebra. Moreover, taking $x = \delta$ in (5) implies that $\delta^*\varepsilon = \varepsilon$ (by the identities $\delta x = \delta$ and $\delta + x = x$), and hence that $\_^*\varepsilon$ cannot be used in a setting without having ε available as a separate process (once δ is accepted as one). So with ε, the interdefinability of the unary and the binary star, noted in [34], is preserved.

MILNER [61] formulated an axiomatisation for the unary Kleene star in BPA with deadlock and empty process, modulo strong bisimulation equivalence. It remains an open question whether this axiomatisation is complete. FOKKINK showed that MILNER's axiomatisation adapted to no-exit iteration (NEI, see Section 4.1) is complete modulo strong bisimulation equivalence, in the presence of empty process.

A particular consequence of MILNER's axiomatisation is (in our notation, using binary Kleene star)

$$\varepsilon^* x = x,$$

which seems a natural identity. Turning to the non-regular operations (see Section 6), the identity $\varepsilon^\sharp x = x$ seems as natural. The other two non-regular operations, i.e. the push-down \$ and the back and forth operation $\leftrightarrows$, have a more surprising effect when combined with ε. Using recursive specifications we find that $\varepsilon^\$ a$ is a solution of the recursive equation

$$X = X^2 + a,$$

and $\varepsilon^{\leftrightarrows} a$ is a solution of

$$X = Xa + a.$$

Both these recursive specifications are easily associated with infinitely branching processes. The (unguarded) specification $X = Xa + a$ occurs in [28] as an example specification that has two distinct solutions: $\sum_{i=1}^{\omega} a^i$ and $a^\omega + \sum_{i=1}^{\omega} a^i$. The transition rules for recursive specifications (see Table 3) as well as those for $\leftrightarrows$ yield the first solution. The interplay of recursive operations with empty process is apparantly nontrivial and deserves further study.

References

[1] L. Aceto and W.J. Fokkink. An equational axiomatization for multi-exit iteration. *Information and Computation*, 137(2):121–158, 1997.

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

[3] L. Aceto, W.J. Fokkink, and A. Ingólfsdóttir. On a question of A. Salomaa: the equational theory of regular expressions over a singleton alphabet is not finitely based. *Theoretical Computer Science*, 209(1/2):163–178, 1998.

[4] L. Aceto, W.J. Fokkink, and A. Ingólfsdóttir. A menagerie of non-finitely based process semantics over BPA*: from ready simulation to completed traces. *Mathematical Structures in Computer Science*, 8(3):193–230, 1998.

[5] L. Aceto, W.J. Fokkink, and A. Ingólfsdóttir. A Cook's tour of equational axiomatizations for prefix iteration. In M. Nivat, ed., *Proceedings 1st Conference on Foundations of Software Science and Computation Structures (FoSSaCS'98)*, Lisbon, LNCS 1378, pp. 20-34. Springer-Verlag, 1998.

[6] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. Chapter 1.3 in this handbook.

[7] L. Aceto and J.F. Groote. A complete equational axiomatization for MPA with string iteration. *Theoretical Computer Science*, 211(1/2):339–374, 1999.

[8] L. Aceto and A. Ingólfsdóttir. An equational axiomatization of observation congruence for prefix iteration. In M. Wirsing and M. Nivat, eds., *Proceedings 5th Conference on Algebraic Methodology and Software Technology (AMAST'96)*, Munich, LNCS 1101, pp. 195–209. Springer-Verlag, 1996.

[9] D.N. Arden. Delayed logic and finite state machines. In *Theory of Computing Machine Design*, pp. 1–35. University of Michigan Press, 1960.

[10] 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.

[11] 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.

[12] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration. Report P9314, Programming Research Group, University of Amsterdam, 1993.

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

[14] S.L. Bloom and Z. Ésik. Equational axioms for regular sets. *Mathematical Structures in Computer Science*, 3(1):1–24, 1993.

[15] S.L. Bloom, Z. Ésik, and D.A. Taubner. Iteration theories of synchronization trees. *Information and Computation*, 102(1):1–55, 1993.

[16] J.C.M. Baeten and R.J. van Glabbeek. Another look at abstraction in process algebra. In T. Ottmann, ed., *Proceedings 14th Colloquium on Automata, Languages and Programming (ICALP'87)*, Karlsruhe, LNCS 267, pp. 84–94. Springer-Verlag, 1987.

[17] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. Grid protocols based on synchronous communication. *Science of Computer Programming*, 29(1/2):199–233, 1997.

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

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

[20] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, eds., *Proceedings Spring School on Mathematical Methods of Specification and Synthesis of Software Systems '85*, Wendisch-Rietz, LNCS 215, pp. 9–23. Springer-Verlag, 1986.

[21] J.A. Bergstra, J.W. Klop, and J.V. Tucker. Algebraic tools for system construction. In E. Clarke and D. Kozen, eds., *Proceedings 4th Workshop on Logics of Programs*, Pittsburgh, LNCS 164, pp. 34–44. Springer-Verlag, 1984.

[22] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[23] J. Boselie. Expressiveness Results for Process Algebra with Iteration. *Master's Thesis*, University of Amsterdam, 1995.

[24] D.J.B. Bosscher. *Grammars Modulo Bisimulation*. PhD Thesis, University of Amsterdam, 1997.

[25] J.A. Bergstra and A. Ponse. Two recursive generalizations of iteration in process algebra. Report P9808, Programming Research Group, University of Amsterdam, 1998. To appear in *Theoretical Computer Science*.

[26] J.A. Bergstra and A. Ponse. Register Machine Based Processes. Programming Research Group, University of Amsterdam, June 13, 1999. In *Proceedings Marktoberdorf 1999*. Extended version to appear.

BT84

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

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

[29] D. Caucal. Graphes canoniques de graphes algébriques. *Theoretical Informatics and Applications*, 24(4):339–352, 1990.

[30] S. Crvenković, I. Dolinka, and Z. Ésik. A note on equations for commutative regular languages. *Information Processing Letters*, 70(6):265–267, 1999.

[31] F. Corradini, R. De Nicola, and A. Labella. Fully abstract models for nondeterministic regular expressions. In I. Lee and S.A. Smolka, eds., *Proceedings 6th Conference on Concurrency Theory (CONCUR'95)*, Philadelphia, LNCS 962, pp. 130–144. Springer-Verlag, 1995.

[32] F. Corradini, R. De Nicola, and A. Labella. Models of nondeterministic regular expressions. *Journal of Computer and System Sciences*, 59(3):412–449, 1999.

[33] F. Corradini, R. De Nicola, and A. Labella. A finite axiomatization of nondeterministic regular expressions. *Theoretical Informatics and Applications*, 33(4):447–466, 1999.

[34] I.M. Copi, C.C. Elgot, and J.B. Wright. Realization of events by logical nets. *Journal of the ACM*, 5:181–196, 1958.

[35] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[36] W.J. Fokkink and R.J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules, *Information and Computation*, 126(1):1–10, 1996.

[37] W.J. Fokkink. A complete equational axiomatization for prefix iteration. *Information Processing Letters*, 52(6):333-337, 1994.

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

[39] W.J. Fokkink. On the completeness of the equations for the Kleene star in bisimulation. In M. Wirsing and M. Nivat, eds., *Proceedings 5th Conference on Algebraic Methodology and Software Technology (AMAST'96)*, Munich, LNCS 1101, pp. 180–194. Springer-Verlag, 1996.

[40] W.J. Fokkink. Axiomatizations for the perpetual loop in process algebra. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, eds., *Proceedings 24th Colloquium on Automata, Languages and Programming (ICALP'97)*, Bologna, LNCS 1256, pp. 571–581. Springer-Verlag, 1997.

[41] W.J. Fokkink. Language preorder as a precongruence. Report CSR 11-98, University of Wales Swansea, 1998. To appear in *Theoretical Computer Science*.

[42] W.J. Fokkink. Rooted branching bisimulation as a congruence. Report CSR 16-98, University of Wales Swansea, 1998. To appear in *Journal of Computer and System Sciences*.

[43] W.J. Fokkink. *Introduction to Process Algebra*. Springer-Verlag, 2000.

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

[45] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, eds., *Proceedings 1st Conference on Concurrency Theory (CONCUR'90)*, Amsterdam, LNCS 458, pp. 278–297. Springer-Verlag, 1990.

[46] R.J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In A. Borzyszkowski and S. Sokołowski, eds., *Proceedings 18th Symposium on Mathematical Foundations of Computer Science (MFCS'93)*, Gdansk, LNCS 711, pp. 473–484. Springer-Verlag, 1993.

[47] R.J. van Glabbeek. The linear time – branching time spectrum II: the semantics of sequential systems with silent moves. In E. Best, ed., *Proceedings 4th Conference on Concurrency Theory (CONCUR'93)*, Hildesheim, LNCS 715, pp. 66–81. Springer-Verlag, 1993.

[48] R.J. van Glabbeek. Axiomatizing flat iteration. In A. Mazurkiewicz and J. Winkowski, eds., *Proceedings 8th Conference on Concurrency Theory (CONCUR'98)*, Warsaw, LNCS 1243, pp. 228–242. Springer-Verlag, 1997.

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

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

[51] M. Hennessy. A term model for synchronous processes. *Information and Control*, 51(1):58–75, 1981.

[52] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, ed., *Computational Problems in Abstract Algebra*, pp. 263–297. Pergamon Press, 1970.

[53] S.C. Kleene. Representation of events in nerve nets and finite automata. Research Memorandum RM-704, U.S. Air Force Project RAND, The RAND Cooperation, 15 December 1951.

[54] S.C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, eds., *Automata Studies*, pp. 3–41. Princeton University Press, 1956.

[55] C.J. Koomen. *A Structure Theory for Communication Network Control*. PhD Thesis, Delft Technical University, 1982.

[56] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[57] D. Krob. Complete systems of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, 1991.

[58] C.P.J. Koymans and J.L.M. Vrancken. Extending process algebra with the empty process ϵ. Logic Group Preprint Series Nr. 1, CIF, Utrecht University, 1985.

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

[60] R. Milner. A modal characterisation of observable machine-behaviour. In E. Astesiano and C. Böhm, eds., *Proceedings 6th Colloquium on Trees in Algebra and Programming (CAAP'81)*, Genoa, LNCS 112, pp. 25–34. Springer-Verlag, 1981.

[61] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984.

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

[63] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[64] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[65] W.S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[66] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, ed., *Proceedings 5th GI (Gesellschaft für Informatik) Conference*, Karlsruhe, LNCS 104, pp. 167–183. Springer-Verlag, 1981.

[67] D.A. Plaisted. Equational reasoning and term rewriting systems. In D. Gabbay and J. Siekmann, eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, Volume 1, pp. 273–364. Oxford University Press, 1993.

[68] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[69] V.N. Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964. In Russian.

[70] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the ACM*, 13(1):158–169, 1966.

[71] A. Salomaa. *Theory of Automata*. International Series of Monographs in Pure and Applied Mathematics 100. Pergamon Press, 1969.

[72] P.M. Sewell. Nonaxiomatisability of equivalences over finite state processes. *Annals of Pure and Applied Logic*, 90(1/3):163–191, 1997.

[73] D.R. Troeger. Step bisimulation is pomset equivalence on a parallel language without explicit internal choice. *Mathematical Structures in Computer Science*, 3(1):25–62, 1993.

[74] J.L.M. Vrancken. The algebra of communicating processes with empty process. *Theoretical Computer Science*, 177(2):287-328, 1997.

[75] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1/2):89–105, 1995.

[76] M.B. van der Zwaag. Some verifications in process algebra with iota. In J.F. Groote, S.P. Luttik, and J.J. van Wamel, eds., *Proceedings 3rd Workshop on Formal Methods for Industrial Critical Systems (FMICS'98)*, Amsterdam, pp. 347–368. Stichting Mathematisch Centrum, 1998.