Concrete process algebra

J.C.M. Baeten and C. Verhoef
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
NL-5600 MB Eindhoven
The Netherlands
email: {chrisv, josb}@win.tue.nl

Note: J.C.M. Baeten received partial support and C. Verhoef received full support from ESPRIT basic research action 7166, CONCUR2.

Note: This report will appear (in a slightly different version) as a chapter in the forthcoming Volume 4 of the Handbook of Logic in Computer Science (eds. S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, Oxford University Press.

Contents

> Algebra may be considered, in its most general form, as *the* science which treats of the combinations of arbitrary signs and symbols by means of defined through arbitrary laws ... [Peackock, 1830].

1 Introduction

Concurrency theory is the branch of computer science that studies the mathematics of concurrent or distributed systems. In concurrency theory, the design of such mathematics is studied and issues concerning the specification and verification of such systems are analysed. Often, a concurrent system is called a process. In order to analyse a large and complex process it is desirable to be able to describe it in terms of smaller and simpler processes. Thus, it seems natural to have some simple processes—the ones that are not subject to further investigation—and operators on them to compose larger ones thus resulting in an algebraic structure. In order to reason about large processes it is often useful to have a set of basic identities between processes at one's disposal. The most relevant identities among them are normally called axioms. The axiomatic and algebraic point of view on concurrency theory is widely known as process algebra.

The most well-known algebraic concurrency theories are the ones known by the acronyms CCS, CSP, and ACP. CCS is the Calculus of Communicating Systems of [Milner, 1980]. Theoretical CSP originates from [Brookes *et al.*, 1984]; the acronym CSP stands for Communicating Sequential Processes. ACP is the Algebra of Communicating Processes; the original reference to ACP is [Bergstra and Klop, 1984a]; we note that recently the full version of [Bergstra and Klop, 1984a] has appeared; see [Bergstra and Klop, 1995]. Of these three, (theoretical) CSP is the most abstract (it identifies more processes than the other two), and tends in the direction of a specification language. The other two, CCS and ACP, are based on the same notion of equivalence (bisimulation equivalence), and are more operationally oriented than CSP. They tend in the direction of a programming language. Of the two, CCS has links to logic and λ-calculus, and ACP may be best characterized as a purely algebraical approach.

In this survey we focus on concrete process algebra. Concrete process algebra is the area of algebraic concurrency theory that does not incorporate a notion called abstraction. Abstraction is the ability to hide information, to abstract information away. The reason that we refrain from incorporating this important issue is that concrete process algebra is already such a large part of the theory that it justifies its own survey. Moreover, it is more and more recognized that for the understanding of issues in large languages it is often convenient first to study such issues in a basic language, a language with less features. For instance, some decidability results in process algebra are obtained in this way. Other examples of such basic languages are Milner's basic CCS [Milner, 1980], BCCSP [Glabbeek, 1990], ASTP [Nicollin and Sifakis, 1994], TCCS_0 [Moller and Tofts, 1990], BPP [Christensen *et al.*, 1993], and the pair BPA/PA [Bergstra and Klop, 1982]. The results that may be obtained for a basic language are almost always useful when the language is extended with additional constructs. Most of the time, these basic languages are concrete. In this survey we will see many examples where a result for a basic language is very useful for an extended version of the language.

Another reason to focus on concrete process algebra is that it is indeed purely algebraically a neat theory. On the other hand, the theory with a form of abstraction and thus with some special constant such as Milner's silent action (τ) or the empty process ε of [Koymans and Vrancken, 1985] is not (yet) stabilized. That is, there are many variants of the theory and it is not clear if there exists a superior variant. For instance, there are two closely related competitive equivalences for the theory with so-called τ abstraction: observational congruence [Milner, 1980] and branching bisimulation equivalence [Glabbeek and Weijland, 1989].

To obtain a uniform notation, since the majority of the available concrete process algebras are ACP-like ones, and since the ACP approach is the most algebraical approach, we survey the algebraical part in the ACP-style process algebra of [Bergstra and Klop, 1984a; Bergstra and Klop, 1995]. As for the semantics of the various languages we deviate from the approach of [Bergstra and Klop, 1984a; Bergstra and Klop, 1995] since nowadays many process algebras have an operational semantics in the style of [Plotkin, 1981]. So, we equip all the languages with such an operational semantics. In the articles [Bergstra and Klop, 1982], [Bergstra and Klop, 1984b], BPA, PA, and ACP were introduced with a semantics in terms of projective limit models. When we restrict ourselves to guarded recursion, projective limit models identify exactly the (strongly) bisimilar processes. The projective limit models are an algebraic reformulation of the topological structures used in [Bakker and Zucker, 1982]. Regarding syntax as well as semantics, [Bergstra and Klop, 1982] reformulates [Bakker and Zucker, 1982] in order to allow more efficient algebraic reasoning.

For those readers who want to know more about possibly other approaches to process algebra (with abstraction), we refer to the following four text books in the area [Baeten and Weijland, 1990], [Hennessy, 1988], [Hoare, 1985], and [Milner, 1989]; see also section 4.

Finally, we briefly review what can be expected in this survey. The survey is organized into three sections (not counting this section).

The first section (2) describes concrete sequential processes; that is, in this section we even refrain from discussing parallelism. In this section, we will meet and tackle many problems that accompany the design of any algebraic language. Since the languages are simple, it is relatively easy to explain the solutions. The solutions that we obtain for the concrete sequential processes turn out to be useful for other languages, too. In particular, we will use these solutions in section 3 where we discuss concrete concurrent processes. Lastly, section 4 gives directions for further reading.

Acknowledgements We thank the proof readers, Jan Bergstra (University of Amsterdam and Utrecht University) and Jan Willem Klop (CWI and Free University Amsterdam) for their useful comments. Also comments by Twan Basten (Eindhoven University of Technology), Kees Middelburg (PTT Research and Utrecht University), Alban Ponse (University of Amsterdam), and Michel Reniers (Eindhoven University of Technology) were appreciated. Special thanks go to Joris Hillebrand (University of Amsterdam) for his essential help in the final stages of the preparation of the document.

2 Concrete sequential processes

In this section we will introduce some basic concepts that can be found in process algebra. We will do this in a modular way. That is, first we treat a basic theory that is the kernel for all the other theories that we will discuss subsequently. The basic theory describes finite, concrete, sequential nondeterministic processes. Then we add features to this kernel that are known to be important in process algebra: for instance, deadlock or recursion to mention a few. Such features make the kernel more powerful for both theoretical and practical purposes. We also show that each feature is a so-called conservative extension of the original theory; thus, we may argue that our approach is modular.

2.1 Introduction

In this subsection we give the reader an idea of what can be expected in the subsections of the sequential part of this survey.

We start with the basic language. Once we have treated this language, we will extend it in the following subsections with important features. We discuss the notions of recursion in subsection 2.3, projection in 2.4, deadlock (or inaction) in 2.5, empty process in 2.6, and we discuss the following operators: renaming operators in subsection 2.7, state operators in 2.8 and 2.9, the priority operator in 2.10, and Kleene's binary star operator in 2.11. Next, we focus in subsection 2.12 on an extension with time. Then subsection 2.13 follows with pointers to extensions that we do not discuss in this survey. Finally, we discuss decidability and expressiveness issues in subsection 2.14 for some of the languages introduced.

2.2 Basic process algebra

First, we list some preliminaries. Then we treat the basic language of this chapter. Next, we devote subsection 2.2.2 to term rewriting analysis; we discuss a powerful method that we will frequently need subsequently. In the next, and last, subsection 2.2.3 we discuss an operational semantics for our basic language. In 2.2.3 we also treat a meta-theorem on operational semantics that we will often use in the rest of this survey.

We assume that we have an infinite set V of variables with typical elements $x, y, z, \ldots$. A (single sorted) signature Σ is a set of function symbols together with their arity. If the arity of a function symbol $f \in \Sigma$ is zero we say that f is a constant symbol. The notion of a term (over Σ) is defined as expected: $x \in V$ is a term; if $t_1, \ldots, t_n$ are terms and if $f \in \Sigma$ is n-ary then $f(t_1, \ldots, t_n)$ is a term. A term is also called an open term; if it contains no variables we call it closed. We denote the set of closed terms by $C(\Sigma)$ and the set of open terms by $O(\Sigma)$ (note that a closed term is also open). We also want to speak about the variables occurring in terms: let $t \in O(\Sigma)$ then $var(t) \subseteq V$ is the set of variables occurring in t.

A substitution σ is a map from the set of variables into the set of terms over a given signature. This map can easily be extended to the set of all terms by substituting for each variable occurring in an open term its σ-image.

2.2.1 The theory Basic Process Algebra

We will give the theory Basic Process Algebra or BPA in terms of an equational specification. BPA is due to [Bergstra and Klop, 1982].

Definition 2.2.1. An *equational specification* (Σ, E) consists of a set Σ that is a signature and a set of equations of the form $t_1 = t_2$ where t_1 and t_2 are (open) terms. The equations in E are often referred to as *axioms*.

We define the definition of derivability of an equation from an equational specification.

Definition 2.2.2. Let (Σ, E) be an equational specification. We define inductively when an equation $s = t$ is *derivable* from the equational specification (Σ, E) (with s and t terms over Σ). When an equation $s = t$ is derivable from (Σ, E) we write $(\Sigma, E) \vdash s = t$ or provided that no confusion arises $E \vdash s = t$. We call the symbol $\vdash$ the derivability relation.

(i) $\quad s = t \in E \Longrightarrow E \vdash s = t$

(ii) $\quad E \vdash s = t \Longrightarrow E \vdash t = s$

(iii) $\quad E \vdash t = t$

(iv) $\quad E \vdash t = s$ and $E \vdash s = u \Longrightarrow E \vdash t = u$

Let σ be a substitution.

(v) $\quad E \vdash s = t \Longrightarrow E \vdash \sigma(s) = \sigma(t)$

Let $C[\_]$ be a context.

(vi) $\quad E \vdash s = t \Longrightarrow E \vdash C[s] = C[t]$

We recall that a *context* is a term with a hole in it; contexts can be defined inductively in the obvious way.

Table 1. BPA.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y)z = xz + yz$	A4
$(xy)z = x(yz)$	A5

Now we give the theory BPA $= (\Sigma_{\mathrm{BPA}}, E_{\mathrm{BPA}})$.

We begin with the signature Σ_{BPA}. There are two binary operators present in Σ_{BPA}; they are denoted $+$ and $\cdot$. The signature Σ_{BPA} also contains a number of constants with typical names $a, b, c, \ldots$. We will use the capital letter A for the set of constants. The set A can be seen as a parameter of the theory BPA: for each application the set A will be specified. For now it is only important that there are constants. This ends our discussion of the signature.

The set of equations E_{BPA} consists of the five equations A1–5 in table 1. The variables x, y, and z in table 1 are universally quantified. They stand for elements of some arbitrary model of BPA. These elements are often called *processes*.

Remark 2.2.3. Terms will be denoted according to the same conventions as the usual ones for summation and multiplication. We will often omit the centered dot in a product. The centered dot binds stronger than the plus. Thus, $xy + z$ means $(x \cdot y) + z$ and the brackets in $x(y + z)$ cannot be omitted.

Intuition We will give an intuitive meaning of the signature and the axioms respectively. Formal semantics can be found in 2.2.3.

The constants $a, b, c, \ldots$ are called *atomic actions* or *steps*. We consider them as processes, which are not subject to any investigation whatsoever.

We think of the centered dot $(\cdot)$ as *sequential composition*. The process xy is the process that first executes the process x and when (and if) it is completed y starts.

The sum or *alternative composition* $x + y$ of two processes x and y represents the process that either executes x or y but not both.

Now we will discuss the axioms of table 1.

Axiom A1 expresses the *commutativity* of the alternative composition. It says that the choice between x and y is the same as the choice between y and x.

Axiom A2 expresses the *associativity* of the plus. It says that first choosing between $x + y$ and z and then (possibly) a choice between x and y

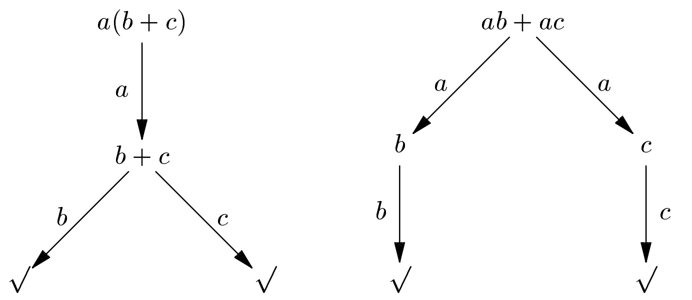

Fig. 1. Two deduction graphs with the same execution paths ab and ac but with different choice moments.

is the same as choosing between x and $y+z$ and then (possibly) a choice between y and z.

Axiom A3 expresses the *idempotency* of the alternative composition. A choice between x and x is the same as a choice for x.

Axiom A4 expresses the *right distributivity* of the sequential composition over the alternative composition. A choice between x and y followed by z is the same as a choice between x followed by z and y followed by z.

Axiom A5 expresses the associativity of the sequential composition. First xy followed by z is the same as first x followed by yz.

Full distributivity We will explain why only right distributivity is presented in table 1. An axiom that does not appear in BPA is the axiom that expresses the *left distributivity* (LD) of the sequential composition over the alternative composition:

$$\text{LD} \quad x(y+z) = xy + xz.$$

Axioms A4 and LD together would give full distributivity. Axiom LD is not included on intuitive grounds. In the left-hand side of LD the moment of choice is later than in the right-hand side. For in $a(b+c)$ the choice between b and c is made after the execution of a, whereas in $ab+ac$ first the choice between ab and ac must be made and then the chosen term can be executed, as in figure 1, where we depict two deduction graphs. See definition 2.2.24 later on for a formal definition of a deduction graph.

The right-hand side of LD is often called a *non-deterministic* choice, which is a subject of research on its own.

Structural induction Structural induction is a basic proof technique in process algebra when closed terms are involved. We will inductively define the class of basic terms. It will turn out that every closed term can be written as a basic term.

Definition 2.2.4. An atomic action is a *basic* term. If t is a basic term and $a \in A$, then $a \cdot t$ is a basic term. If t and s are basic terms, then $t + s$ is a basic term.

Remark 2.2.5. If we consider terms to be identical that only differ in the order of the summands, we can see that basic terms have the following form

$$\sum_{i=1}^{n} a_i \cdot t_i + \sum_{j=1}^{m} b_j,$$

where $a_i, b_j \in A$, $1 \leq i \leq n, 1 \leq j \leq m$, $n + m \geq 1$, and the t_i again basic for $1 \leq i \leq n$. Here,

$$\sum_{i=1}^{k} p_i$$

is an abbreviation of $p_1 + \cdots + p_k$, and if $n = 0, m \geq 1$ we have a term of the form $b_1 + \cdots + b_m$. Similarly if $m = 0, n \geq 1$.

In the next proposition we see that if we want to prove a statement correct for all closed terms (see subsection 2.2 for the definition of a closed term), it suffices to prove it for basic terms. Since they are inductively defined we can use structural induction.

Proposition 2.2.6. *Let t be a closed* BPA *term. Then there is a basic term s such that* BPA $\vdash t = s$.

Proof. We will use term rewriting analysis to prove 2.2.6. In the next subsection (2.2.2) we will give a short introduction to this theory. We will use the proof of the fact that the term rewriting system of table 2 is strongly normalizing as a running example. Once we know that the term rewriting system of table 2 has this property, it is not difficult to see that given a closed BPA term t, there is a normal form s, which is a basic term, and that BPA $\vdash t = s$, which proves the proposition. ∎

2.2.2 Term rewriting systems

In this subsection we will introduce a result from the field of term rewriting systems that is a powerful tool in process algebra. We will do this by means of an example: we will prove the essential step of proposition 2.2.6 using the result. General references to this theory are [Dershowitz and Jouannaud, 1990] and [Klop, 1992].

Definition 2.2.7. A term rewriting system or term reduction system is a pair (Σ, R) with Σ a signature and R a set of rewriting (or reduction) rules. The reduction rules are of the form $s \to t$, where s and t are terms over the signature Σ; we denote this set by $O(\Sigma)$ with O for open terms over Σ. For these terms we have two constraints

- s is not a variable.

- all variables that occur in t must also occur in s.

Often, we give reduction rules a name, as in our example below. The one-step reduction relation on terms, also denoted $\rightarrow$, is the smallest relation on terms containing the rules that is closed under substitutions and contexts. We denote the transitive-reflexive closure of the one-step reduction relation $\rightarrow$ by $\twoheadrightarrow$.

Example 2.2.8. We give an example of a term reduction system. Let T be the term reduction system with as signature that of BPA and as a set of reduction rules those in table 2. Note that we do not have rules corresponding to axioms A1 or A2, as these axioms have no clear direction.

A useful property for a term reduction system is that there are no infinite reductions possible. Below we define some more notions.

Definition 2.2.9. Let (Σ, R) be a term reduction system and let s be a Σ term. We say that s is a normal form if there is no term t such that $s \rightarrow t$. A term s has a normal form if there exists a normal form t with $s \twoheadrightarrow t$.

A term s_0 is called strongly normalizing or terminating (SN) if there exists no infinite series of reductions beginning in s_0:

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots .$$

A term reduction system is called strongly normalizing if every term of it is SN.

Example 2.2.10. For our running example we have that the process a is a normal form, the process $(ab)c$ is not a normal form but has one, $a(bc)$ (use RA5), and there are no infinite reductions possible. To prove the last statement, we introduce the method of the recursive path ordering following [Klop, 1992].

Definition 2.2.11. Let (Σ, R) be a term reduction system. We define $O^*(\Sigma)$ to be the superset of $O(\Sigma)$ where some function (and constant) symbols may be marked with an asterisk $(*)$.

Example 2.2.12. A typical element of the superset O^* of our running example is

$$a \cdot^* (b \cdot c^*).$$

Table 2. A term reduction system for BPA.

$x + x \rightarrow x$	RA3
$(x+y)z \rightarrow xz + yz$	RA4
$(xy)z \rightarrow x(yz)$	RA5

Definition 2.2.13. Let Σ be a signature and let $>$ be a well-founded partial ordering on Σ. Let $\to$ be the reduction relation that is defined in the clauses RPO1–5 in table 3.

Let $s, t \in O^*(\Sigma)$. We write $s \cong t$ if s can be obtained from t by permuting the arguments of t.

Let $s, t \in O(\Sigma)$. We write $s >_{rpo} t$ if there exists a $u \in O(\Sigma)$ such that $u \cong t$ and $s \to^+ u$. With $\to^+$ we mean the transitive closure of $\to$.

Example 2.2.14. Suppose that we have the following ordering on the signature of our running example: $\cdot > +$. With this choice of $>$ we can execute the following reduction:

$$\begin{array}{lll}(x+y)\cdot z & >_{rpo} & (x+y)\cdot^* z \\ & >_{rpo} & (x+y)\cdot^* z + (x+y)\cdot^* z \\ & >_{rpo} & (x+^* y)\cdot z + (x+^* y)\cdot z \\ & >_{rpo} & x\cdot z + y\cdot z.\end{array}$$

In the following theorem (due to [Dershowitz, 1987]) we will see that if we have such a reduction for each rewrite rule we have a strongly normalizing term rewriting system.

Theorem 2.2.15. *Let (Σ, R) be a term rewriting system with finitely many rewriting rules and let $>$ be a well-founded ordering on Σ. If $s >_{rpo} t$ for each rewriting rule $s \to t \in R$, then the term rewriting system (Σ, R) is strongly normalizing.*

The method of the recursive path ordering is not convenient for rewriting rules such as $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$. We will discuss a variant of the above method which is known as the lexicographical variant of the recursive path

Table 3. The recursive path ordering.

RPO1. Mark head symbol ($k \geq 0$)

$$H(t_1, \ldots, t_k) \to H^*(t_1, \ldots, t_k)$$

RPO2. Make copies under smaller head symbol ($H > G,\ k \geq 0$)

$$H^*(t_1, \ldots, t_k) \to G\big(H^*(t_1, \ldots, t_k), \ldots, H^*(t_1, \ldots, t_k)\big)$$

RPO3. Select argument ($k \geq 1,\ 1 \leq i \leq k$)

$$H^*(t_1, \ldots, t_k) \to t_i$$

RPO4. Push $*$ down ($k \geq 1,\ l \geq 0$)

$$H^*\big(t_1, \ldots, G(s_1, \ldots, s_l), \ldots, t_k\big) \to H\big(t_1, \ldots, G^*(s_1, \ldots, s_l), \ldots, t_k\big)$$

RPO5. Handling contexts

$$s \to t \Longrightarrow H(\ldots, s, \ldots) \to H(\ldots, t, \ldots)$$

ordering. The idea is that we give certain function symbols the so-called lexicographical status (the remaining function symbols have the multiset status). The function symbols with a lexicographical status have, in fact, an arbitrary but fixed argument for which this status holds. For instance, we give the sequential composition the lexicographical status for the first argument.

We also have an extra rule to cope with function symbols with a lexicographical status. For a k-ary function symbol H with the lexicographical status for the ith argument we have the following extra rule in table 4. The idea behind this rule is that if the complexity of a dedicated argument is reduced and the complexity of the other arguments increases (but not unboundedly) the resulting term will be seen as less complex as a whole.

Definition 2.2.16. Let $s, t \in O(\Sigma)$. We write $s >_{lpo} t$ if $s \to^{+} t$ with $\to^{+}$ this time the transitive closure of the reduction relation defined by the rules RPO1–5 and LPO.

Example 2.2.17. Suppose that we have the following ordering on the signature of our running example $\cdot > +$. We give the symbol $\cdot$ the lexicographical status for the first argument. Consider the following reduction:

$$\begin{array}{lll}(x \cdot y) \cdot z & >_{lpo} & (x \cdot y) \cdot^{*} z \\ & >_{lpo} & (x \cdot^{*} y) \cdot \big((x \cdot y) \cdot^{*} z\big) \\ & >_{lpo} & x \cdot \big((x \cdot^{*} y) \cdot z\big) \\ & >_{lpo} & x \cdot (y \cdot z).\end{array}$$

Note that we did not use permutation of arguments in the deduction of example 2.2.14. This means that we also have

$$(x + y) \cdot z >_{lpo} x \cdot z + y \cdot z.$$

In the following theorem (due to [Kamin and Lévy, 1980]) we will see that if we have such a reduction for each rewrite rule we also have a strongly normalizing term rewriting system.

Theorem 2.2.18. *Let (Σ, R) be a term rewriting system with finitely many*

Table 4. The extra rule for the lexicographical variant of the recursive path ordering. We have that H has the lexicographical status for the ith argument.

LPO Reduce ith argument ($k \geq 1,\ 1 \leq i \leq k,\ l \geq 0$)
Let $t \equiv H^{*}\big((t_1, \ldots, t_{i-1}, G(s_1, \ldots, s_l), t_{i+1}, \ldots, t_k\big)$
Then $t \to H\big(t, \ldots, t, G^{*}(s_1, \ldots, s_l), t, \ldots, t\big)$

rewriting rules and let $>$ be a well-founded ordering on Σ. If $s >_{lpo} t$ for each rewriting rule $s \to t \in R$, then the term rewriting system (Σ, R) is strongly normalizing.

So with the aid of the above theorem we conclude that the term rewriting system of table 2 is strongly normalizing (we leave the case RA3 to the reader). To prove strong normalization we will henceforth confine ourselves to giving a partial ordering $>$ on the signature and to saying which function symbols do have the lexicographical status (and for which argument).

2.2.3 Semantics of basic process algebra

In this subsection we will give an operational semantics of BPA in the style of Plotkin; see [Plotkin, 1981]. The usual procedure to give an operational semantics is to only give a table with so-called transition rules; see, for instance, table 5. In this subsection we will make a small excursion to the so-called general theory of structured operational semantics because in that framework we can formulate general theorems that hold for large classes of languages. The reason for this detour is that we will use such general results many times in this chapter.

To start with, we define the notion of a term deduction system, which is taken from [Baeten and Verhoef, 1993]. It is a modest generalization of the notion of a transition system specification that originates from [Groote and Vaandrager, 1992]. The idea of a term deduction system is that it can be used not only to define a transition relation but also to define unary predicates on states that turn out to be useful. See table 5 for a typical term deduction system; it is a definition of both transition relations $\xrightarrow{a}$ and unary predicates $\xrightarrow{a}\surd$ for each $a \in A$.

First we list some preliminaries for completeness sake.

We recall that the meaning or *semantics* of an equation specification (Σ, E) is given by a *model* or an *algebra* $\mathcal{A}$. Such an algebra consists of a set of elements U (called the *universe* or *domain* of $\mathcal{A}$) together with constants in U and functions from U^n to U. We call $\mathcal{A}$ a Σ-algebra when there is a correspondence between the constant symbols in Σ and the constants in U, and between the function symbols in Σ and the functions in $\mathcal{A}$ (of the same arity). We call such a correspondence an *interpretation*. Now if $\mathcal{A}$ is a Σ-algebra of the equational specification (Σ, E), then an equation $s = t$ over (Σ, E) has a meaning in $\mathcal{A}$, when we interpret the constant and function symbols in s and t by the corresponding constants and functions in $\mathcal{A}$. Moreover, the variables in s and t are universally quantified. So when for all variables in s and t we have that the statement $s = t$ is true in $\mathcal{A}$ we write $\mathcal{A} \models s = t$ and we say $\mathcal{A}$ satisfies $s = t$ or $s = t$ holds in $\mathcal{A}$. We call $\models$ the satisfiability relation. If a Σ-algebra $\mathcal{A}$ satisfies all equations $s = t$ over (Σ, E) we write $\mathcal{A} \models (\Sigma, E)$ (or $\mathcal{A} \models E$) and we say that $\mathcal{A}$ is an algebra for E, or a model of E. We also say that E is a sound axiomatization of $\mathcal{A}$. See remark 2.2.34 and theorem 2.2.35 for an example.

Definition 2.2.19. A term deduction system is a structure (Σ, D) with Σ a signature and D a set of deduction rules. The set $D = D(T_p, T_r)$ is parameterized with two sets, which are called respectively the set of predicate symbols and the set of relation symbols. Let $P \in T_p$ and $R \in T_r$, and $s, t, u \in O(\Sigma)$. We call expressions Ps and tRu formulas. A deduction rule $d \in D$ has the form

$$\frac{H}{C}$$

with H a set of formulas and C a formula. We call the elements of H the hypotheses of d and we call the formula C the conclusion of d. If the set of hypotheses of a deduction rule is empty we call such a rule an axiom. We denote an axiom simply by its conclusion provided that no confusion can arise. Notions such as "substitution", "*var*", or "closed" extend to formulas and deduction rules as expected.

Example 2.2.20. Let $T(\mathrm{BPA})$ be the term deduction system consisting of the signature of BPA and the deduction rules of table 5. As relation symbols we have the transition relations and as predicate symbols we have the successful termination predicates. The intuitive idea of $s \xrightarrow{a} s'$ is that, for example, a machine in state s can evolve into state s' by executing step a. The intended meaning of $s \xrightarrow{a} \surd$ is that this machine in state s can terminate successfully by executing a; the symbol $\surd$ (pronounced tick) stands for successful termination.

Next, we give the definition of a proof of a formula from a set of deduction rules. This definition is taken from [Groote and Vaandrager, 1992].

Definition 2.2.21. Let $T = (\Sigma, D)$ be a term deduction system. A proof of a formula ψ from T is a well-founded upwardly branching tree of which the nodes are labelled by formulas such that the root is labelled with ψ and if χ is the label of a node q and $\{\chi_i : i \in I\}$ is the set of labels belonging

Table 5. Derivation rules of $T(\mathrm{BPA})$.

$$a \xrightarrow{a} \surd$$

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

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

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

$$\begin{tikzcd} a \xrightarrow{a} \surd \arrow[d] \\ a + b \xrightarrow{a} \surd \arrow[d] \\ (a+b)c \xrightarrow{a} c \end{tikzcd}$$

Fig. 2. A proof

to the nodes $\{q_i : i \in I\}$ directly above q (I some is index set) then there is a deduction rule

$$\frac{\{\phi_i : i \in I\}}{\phi}$$

and a substitution $\sigma : V \longrightarrow O(\Sigma)$ such that $\sigma(\phi) = \chi$ and $\sigma(\phi_i) = \chi_i$ for $i \in I$.

If a proof of ψ exists, we say that ψ is provable from T, notation $T \vdash \psi$.

Example 2.2.22. It is not difficult to verify that the tree in figure 2 is a proof of the transition $(a+b)c \xrightarrow{a} c$.

Next, we define the notion of a deduction graph. It generalizes the well-known notion of a labelled state transition diagram in the sense that it can also handle unary predicates on states. First, we need a reachability definition.

Definition 2.2.23. Let T be a term deduction system and let s and t be terms. We define a binary relation ρ as the transitive reflexive closure of the binary relation $\{(s,t) \mid \exists R : T \vdash sRt\}$. If $s\rho t$ we say that from s we can reach t, or that t is reachable from s.

Definition 2.2.24. Let T be a term deduction system. The deduction graph of a closed term s is a labelled graph that is obtained as follows. The nodes of this graph are the terms that can be reached from s; the labels of nodes are sets of predicate symbols. The edges of a deduction graph are labelled by relation symbols. Let t be a node of the deduction graph of s; then there is a label $\{P : T \vdash Pt\}$ attached to this node. Let t and t' be nodes of the deduction graph of s. Then there exists an edge labelled by R from t to t' in the deduction graph of s if and only if we have $T \vdash tRt'$. See figure 3 for examples. Observe that we are a bit sloppy there: we identify the edges of the graph with its labels; that is, we render an edge

$$\xrightarrow{\xrightarrow{a}}$$

simply as $\xrightarrow{a}$. Moreover, we depict a predicate $\xrightarrow{b}\surd$ as a b-labelled edge to a node $\surd$.

Next, we define the notion of a structured state system [Baeten and Verhoef, 1993]. It is a generalization of the well-known notion of a labelled transition system.

Definition 2.2.25. A structured state system is a triple (S, S_p, S_r) where S is a set of states, S_p is a subset of the power set of S and S_r is a subset of the power set of $S \times S$. The sets S_p and S_r are called respectively the set of predicates and the set of relations.

A term deduction system induces in a natural way a structured state system.

Definition 2.2.26. Let $T = (\Sigma, D)$ be a term deduction system and let $D = D(T_p, T_r)$. The structured state system G induced by T has as its set of states $S = C(\Sigma)$; the predicates and relations are the following.

$$\begin{aligned} S_p &= \Big\{ \{ t \in C(\Sigma) \mid T \vdash Pt \} \Bigm| P \in T_p \Big\}, \\ S_r &= \Big\{ \{ (s,t) \in C(\Sigma) \times C(\Sigma) \mid T \vdash sRt \} \Bigm| R \in T_r \Big\}. \end{aligned}$$

Example 2.2.27. Let $L = L(\text{BPA})$ be the structured state system induced by the term deduction system $T(\text{BPA})$ from example 2.2.20. In figure 3 we depict two deduction graphs of the terms $ab + a(b+b)$ and ab.

Both terms represent the same behaviour: first a is executed, then b, and then both systems terminate successfully. However, they do not have the same deduction graphs as we can see in figure 3. So the set of deduction graphs is not directly a model of BPA since in that system we want the alternative composition to be idempotent.

Many different equivalence notions have been defined in order to identify states that have the same behaviour; see [Glabbeek, 1990] and [Glabbeek, 1993] for a systematic approach. The finest among them is the so-called strong bisimulation equivalence of [Park, 1981]. We will take the formulation of [Baeten and Verhoef, 1993] for this well-known notion.

It will turn out that the two deduction graphs of example 2.2.27 are bisimilar.

Definition 2.2.28. Let $G = (S, S_p, S_r)$ be a structured state system. A relation $B \subseteq S \times S$ is called a (strong) bisimulation if for all $s, t \in S$ with sBt the following conditions hold. For all $R \in S_r$

$$\begin{aligned} &\forall s' \in S(sRs' \Rightarrow \exists t' \in S : tRt' \wedge s'Bt'), \\ &\forall t' \in S(tRt' \Rightarrow \exists s' \in S : sRs' \wedge s'Bt'), \end{aligned}$$

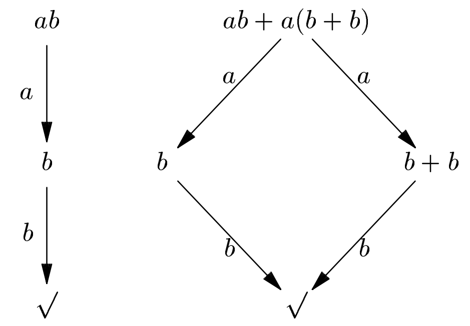

Fig. 3. Two deduction graphs.

and for all $P \in S_p$

$$Ps \Leftrightarrow Pt.$$

The first two conditions are known as the transfer property. Two states s and t in S are bisimilar in the structured state system G if there exists a bisimulation relation containing the pair (s, t). The notation is $s \sim_G t$ or $s \sim t$ provided that no confusion can arise.

Note that bisimilarity is an equivalence relation, called a bisimulation equivalence.

Example 2.2.29. Let $L = L(\mathrm{BPA})$ be the structured state system of example 2.2.27. It is not hard to see that the two states $ab + a(b + b)$ and ab are bisimilar. We graphically depict the bisimulation relation by connecting its pairs with a dashed line as in figure 4.

When two states in a deduction graph are bisimilar, we also call the corresponding terms bisimilar.

An example of two deduction graphs that are *not* bisimilar can be found in figure 1.

Considered as deduction graphs, the two processes $x = ab$ and $y = ab + a(b + b)$ are not equal, but from a process algebraic point of view, we want them to be. That is, they both first execute the atomic process a and then the b. So we would like to have a model for which $ab = ab + a(b + b)$. The usual approach to obtain this is to work with an equivalence relation and to identify equivalent objects. We then say that the objects are equal modulo this equivalence relation. If x is a process and $\sim$ denotes bisimulation equivalence, the equivalence class is defined $[x] = \{y : x \sim y\}$. So, in the above example the two processes $x = ab$ and $y = ab + a(b + b)$ are equal modulo bisimulation equivalence: $[x] = [y]$. Now it would be very nice if the equivalence class is independent of the chosen representative. If this is the case, we can easily define our process algebra operators on these classes.

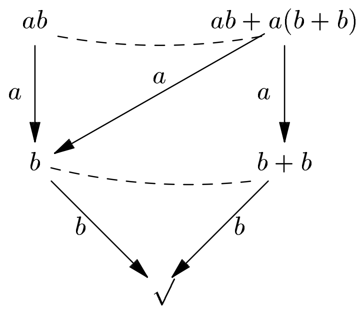

Fig. 4. Bisimilar deduction graphs.

For instance, the alternative composition can be defined as $[x]+[y] = [x+y]$. In general, the assumption that a relation is an equivalence relation is too weak for this purpose. The additional property that does the job is called congruency. In the next definition we define this well-known notion.

Definition 2.2.30. Let Σ be a signature. An equivalence relation R on the set of closed Σ terms is called a congruence if for all n-ary function symbols $f \in \Sigma$ we have

$$x_1 \; R \; y_1, \ldots, x_n \; R \; y_n \Longrightarrow f(x_1, \ldots, x_n) \; R \; f(y_1, \ldots, y_n)$$

where $x_1, y_1, \ldots, x_n, y_n$ are closed Σ terms.

Next, we define some syntactical constraints on the rules of a term deduction system for which it can be proved that if a term deduction system satisfies these constraints then strong bisimulation equivalence will always be a congruence. Below we discuss the so-called *path* format; this stands for "predicates and *tyft/tyxt* hybrid format" and is proposed by [Baeten and Verhoef, 1993]. It is a modest generalization of the *tyft/tyxt* format originating from [Groote and Vaandrager, 1992]. The name *tyft/tyxt* refers to the syntactical form of the deduction rules.

We refer to [De Simone, 1985] for the first paper that discusses syntactical constraints on operational rules. Nowadays, the syntactical constraints formulated in that paper are often referred to as the "De Simone format".

Definition 2.2.31. Let $T = (\Sigma, D)$ be a term deduction system with $D = D(T_p, T_r)$. Let I and J in the following be index sets of arbitrary cardinality, let $t_i, s_j, t \in O(\Sigma)$ for all $i \in I$ and $j \in J$, let $P_j, P \in T_p$ be predicate symbols for all $j \in J$, and let $R_i, R \in T_r$ be relation symbols for all $i \in I$. A deduction rule $d \in D$ is in *path* format if it has one of the following four forms:

$$\frac{\{P_js_j : j \in J\} \cup \{t_iR_iy_i : i \in I\}}{f(x_1, \dots, x_n)Rt}$$

with $f \in \Sigma$ an n-ary function symbol, $X = \{x_1, \dots, x_n\}$, $Y = \{y_i : i \in I\}$, and $X \cup Y \subseteq V$ a set of distinct variables;

$$\frac{\{P_js_j : j \in J\} \cup \{t_iR_iy_i : i \in I\}}{xRt}$$

with $X = \{x\}$, $Y = \{y_i : i \in I\}$, and $X \cup Y \subseteq V$ a set of distinct variables;

$$\frac{\{P_js_j : j \in J\} \cup \{t_iR_iy_i : i \in I\}}{Pf(x_1, \dots, x_n)}$$

with $f \in \Sigma$ an n-ary function symbol, $X = \{x_1, \dots, x_n\}$, $Y = \{y_i : i \in I\}$, and $X \cup Y \subseteq V$ a set of distinct variables or

$$\frac{\{P_js_j : j \in J\} \cup \{t_iR_iy_i : i \in I\}}{Px}$$

with $X = \{x\}$, $Y = \{y_i : i \in I\}$, and $X \cup Y \subseteq V$ a set of distinct variables.

If in the above four cases $var(d) = X \cup Y$ we say that the deduction rule d is pure.

We say that a term deduction system is in *path* format if all its deduction rules are in *path* format. We say that a term deduction system is pure if all its rules are pure.

Next, we formulate the congruence theorem for the *path* format. It is taken from [Baeten and Verhoef, 1993]. There the so-called well-founded subcase is proved. [Fokkink, 1994] showed that this requirement is not necessary, thus yielding the following result. Note that we do not use the notion pure in the theorem below. We just define it since we will need this notion later on when we will have our second excursion into the area of general SOS theory.

Theorem 2.2.32. *Let $T = (\Sigma, D)$ be a term deduction system in path format. Then strong bisimulation equivalence is a congruence for all function symbols occurring in Σ.*

Lemma 2.2.33. *Let T(BPA) be the term deduction system that we defined in example 2.2.20. Then bisimulation equivalence is a congruence on the set of closed* BPA *terms.*

Proof. It is easy to see that the operational semantics given in table 5 is in *path* format, so with theorem 2.2.32 we immediately find the proof of this lemma. ∎

Remark 2.2.34. It follows from lemma 2.2.33 that we can take the quotient of the algebra of closed BPA terms with respect to bisimulation equivalence and that the operators of BPA can be defined on this quotient by taking representatives. Next, we will show that this quotient algebra is a model of BPA. We recall that this property is called soundness; that is, if two closed terms x and y are provably equal, $\mathrm{BPA} \vdash x = y$, then we also have that x and y are bisimilar, $x \sim y$.

Theorem 2.2.35. *The set of closed* BPA *terms modulo bisimulation equivalence is a model of* BPA.

Proof. Since bisimulation equivalence is a congruence, we only need to verify the soundness of each separate axiom. We check A1 (see table 1). Let x and y be closed BPA terms. We have to show that $x + y$ is bisimilar with $y + x$ (using the rules of table 5). It is not hard to see that the relation that contains the pair $(x + y, y + x)$ and the diagonal of $S \times S$ is a bisimulation. The cases A2–4 are treated analogously. It remains to check A5. It is easy to see that the relation containing all the pairs of the form $\big((xy)z, x(yz)\big)$ and the diagonal of $S \times S$ is a bisimulation. ∎

Next we will show that BPA is a complete axiomatization of the set of closed terms modulo bisimulation equivalence. We recall that an axiomatization is complete if bisimilar x and y are provably equal with these axioms. Note that we only talk about closed process terms here: complete axiomatizations for open terms are much more difficult, see, e.g. [Groote, 1990a]. But first we will need some preliminaries to prove this; they are listed in the next lemma.

Lemma 2.2.36. *Let x and y be closed* BPA *terms and let $n(z)$ be the number of symbols of a closed* BPA *term z. Then we have:*

(*i*) $T(\mathrm{BPA}) \vdash x \xrightarrow{a} \surd \Longrightarrow \mathrm{BPA} \vdash x = a + x$,

(*ii*) $T(\mathrm{BPA}) \vdash x \xrightarrow{a} y \Longrightarrow \mathrm{BPA} \vdash x = ay + x$,

(*iii*) $T(\mathrm{BPA}) \vdash x \xrightarrow{a} y \Longrightarrow n(x) > n(y)$.

Proof. Easy. Use induction on the depth of the proof. ∎

Theorem 2.2.37. *The axiom system* BPA *is a complete axiomatization of the set of closed* BPA *terms modulo bisimulation equivalence.*

Proof. Let x and y be bisimilar closed BPA terms, notation $x \sim y$. We have to prove that $\mathrm{BPA} \vdash x = y$. With the aid of proposition 2.2.6 and theorem 2.2.35 it is enough to prove this for basic terms. By symmetry, it is even enough to prove that for basic terms x and y

$$x + y \sim y \Longrightarrow \mathrm{BPA} \vdash x + y = y.$$

We will prove this with induction on $n = n(x) + n(y)$. First, let $x = a$. Then $y \xrightarrow{a} \surd$, so with lemma 2.2.36 we find that $x + y = y$. This proves the basis of our induction. Now suppose that $x = ax'$. Then $x + y \xrightarrow{a} x'$, so there is a y' with $y \xrightarrow{a} y'$ and $x' \sim y'$. But then also $x' + y' \sim y'$ and $y' + x' \sim x'$

and with induction we find $x' + y' = y'$ and $y' + x' = x'$. So $x' = y'$. Now $x + y = ax' + y = ay' + y = y$ with lemma 2.2.36. Finally, suppose that $x = x' + x''$. Since $x + y \sim y$, we also have $x' + y \sim y$ and $x'' + y \sim y$. By induction $x' + y = y$ and $x'' + y = y$. So $x + y = x' + x'' + y = y + y = y$. ■

2.3 Recursion in BPA

In this subsection we will add recursion to the theory BPA.

Definition 2.3.1. Let V be a set of variables. A *recursive specification* $E = E(V)$ is a set of equations

$$E = \{X = s_X(V) : X \in V\},$$

where each $s_X(V)$ is a BPA term that only contains variables of V. These equations are called *recursion* equations. A recursion equation is called a *recursive* equation if it has the form $X = s(X)$ where $s(X)$ only contains the variable X. By convention, we use capital letters $X, Y, \ldots$ for variables bound in a recursive specification.

Example 2.3.2. $E_1 = \{X = Xa + a\}$ and $E_2 = \{Y = aY\}$ are examples of recursive specifications.

Definition 2.3.3. A *solution* of a recursive equation is a process in some model of BPA such that its substitution in the recursive equation yields a true statement in that model.

A *solution* $\{\langle X \mid E \rangle : X \in V\}$ of a recursive specification $E(V)$ is a set of processes in some model of BPA such that replacing each variable X by $\langle X \mid E \rangle$ in the recursion equations of $E(V)$ yields true statements in that model. Mostly, we are interested in one particular variable $X \in V$. Abusing terminology we call $\langle X \mid E \rangle$ the solution of E. Moreover, abusing notation we often write X for $\langle X \mid E \rangle$.

Remark 2.3.4. In CCS [Milner, 1980; Milner, 1989] and CSP [Hoare, 1985] the so-called μ notation is used: if $x = t(x)$ is a recursive equation, then $\mu x.t(x)$ is a process satisfying this equation.

Definition 2.3.5. Let $E = E(V)$ be a recursive specification and let t be an open BPA term. Then $\langle t \mid E \rangle$ is the process t with all variables X both occurring in t and V replaced by the processes $\langle X \mid E \rangle$.

At this point we have all the necessary definitions to define the equational specification BPArec, in which rec is an abbreviation for recursion. The signature of BPArec consists of the signature of BPA plus for all recursive specifications $E(V)$ and for all $X \in V$ a constant $\langle X \mid E \rangle$. The axioms of BPArec consist of the axioms of BPA plus for all recursive specifications $E(V) = \{X = s_X : X \in V\}$ and for all $X \in V$ an equation $\langle X \mid E \rangle = \langle s_X \mid E \rangle$.

Definition 2.3.6. Let s be a term over BPA containing a variable X. We call an occurrence of X in s *guarded* if s has a subterm of the form $a \cdot t$ with t a BPA term containing this occurrence of X; in this case we call the atomic action $a \in A$ a guard (of X in s). Otherwise we call the occurrence of X in s *unguarded*.

Definition 2.3.7. We call a term *completely guarded* if all occurrences of all its variables are guarded.

We call a term *guarded* if we can rewrite it to a completely guarded term by use of the axioms. Otherwise, a term is called *unguarded*.

Example 2.3.8. The term $aX + bX$ is completely guarded, $(a+b)X$ is guarded but not completely guarded, and $Xa + Xb$ is unguarded.

Definition 2.3.9. We call a recursive specification *completely guarded* if the right-hand sides of its recursion equations are completely guarded.

We call a recursive specification *guarded* if we can rewrite it to a completely guarded recursive specification by use of the axioms and/or its recursion equations. Otherwise, a recursive specification is called *unguarded*.

Example 2.3.10. In example 2.3.2, the recursive specification E_1 is unguarded and E_2 is completely guarded.

The next two definitions are taken from [Bergstra and Klop, 1986].

Definition 2.3.11. The *(restricted) recursive definition principle* is the assumption that every (guarded) recursive specification has a solution. We refer to this assumption as $\mathrm{RDP}^{(-)}$.

Definition 2.3.12. The *recursive specification principle* (RSP) is the assumption that every guarded recursive specification has at most one solution.

Note that RSP contains the guardedness demand from the beginning (as opposed to AIP and RDP); this is due to the fact that having at most one solution is not feasible for unguarded specifications, take for example $\{X = X\}$.

Together, RDP^- and RSP say that a guarded recursive specification has a *unique* solution.

Semantics The semantics of BPArec can be given completely analogously to the semantics for BPA that we gave in subsection 2.3.

We consider the term deduction system $T(\mathrm{BPArec})$ with as signature the signature of BPArec and as rules the rules in table 5 together with those in table 6. Bisimulation equivalence is a congruence on the structured state system $L(\mathrm{BPArec})$ induced by $T(\mathrm{BPArec})$; see 2.2.32. So on the quotient of the algebra of closed BPArec terms with respect to bisimulation equivalence we can define the operators of BPArec by taking representatives. The next theorem states that this quotient algebra is a model of BPArec.

Theorem 2.3.13. *The set of closed* BPArec *terms modulo bisimulation*

Table 6. Derivation rules for recursion ($X = s_X \in E$).

$$\frac{\langle s_X \mid E\rangle \xrightarrow{a} \surd}{\langle X \mid E\rangle \xrightarrow{a} \surd} \qquad \frac{\langle s_X \mid E\rangle \xrightarrow{a} y}{\langle X \mid E\rangle \xrightarrow{a} y}$$

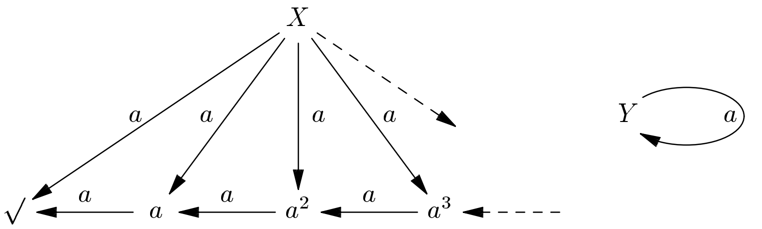

Fig. 5. Two deduction graphs of the processes X and Y that can be found in example 2.3.2.

equivalence is a model of BPArec. *Moreover, it satisfies* RDP *and* RSP.

Proof. As bisimulation equivalence is a congruence we only need to check the soundness of each axiom. The axioms A1–5 are treated in exactly the same way as in theorem 2.2.35. Equations concerning recursion are treated in the same way as A1.

Let $E(V)$ be a recursive specification. Then $\{[\langle X \mid E\rangle] : X \in V\}$ is a solution. See below example 2.2.29 for the $[\cdot]$ notation.

The proof that this model satisfies RSP will be postponed until we have discussed the combination of recursion and so-called projections. See theorem 2.4.36 for a proof. ∎

Example 2.3.14. In figure 5, we depict two deduction graphs of the solutions of the two recursive specifications of example 2.3.2. It is not hard to see that $X \xrightarrow{a} \surd$ and $X \xrightarrow{a} a^n$ for all $n \geq 1$. We can think of the process X as the infinite sum $\sum_{n<\omega} a^n$ and we can think of the process Y as the infinite product a^ω. Note that $X + Y$ and X are not bisimilar since $X + Y$ can do infinitely many a steps, whereas X can perform only finitely many a steps.

2.4 Projection in BPA

In subsection 2.3 we introduced guarded recursive specifications. They are mainly used to specify infinite processes such as a counter: see example 2.14.5. With the principles RDP and RSP we can prove statements involving such infinite processes. See example 3.6.1 for an application of RSP. In this subsection we will introduce another method for this purpose. The idea is that we approximate an infinite process by its finite projections. The

Table 7. Projection.

$\pi_n(a) = a$	PR1
$\pi_1(ax) = a$	PR2
$\pi_{n+1}(ax) = a\pi_n(x)$	PR3
$\pi_n(x+y) = \pi_n(x) + \pi_n(y)$	PR4

finite projections for their part turn out to be closed terms and they can therefore be taken care of by structural induction (see also 2.2 for structural induction). This material is based on the paper [Bergstra and Klop, 1982].

We will define the equational specification BPA + PR, in which PR is an abbreviation for projection.

The signature of BPA + PR consists of the signature of BPA plus for each $n \geq 1$ a unary function π_n that is called a *projection operator of order* n or the nth projection. The axioms of BPA + PR consist of the axioms of table 7 and the axioms of BPA; we call the axioms PR1–4. In table 7 we have $n \geq 1$, the letter a ranges over all the atomic actions, and the variables x and y are universally quantified.

We will now discuss the axioms of table 7.

The idea of projections is that we want to be able to cut off a process at a certain depth. An atomic action is intrinsically the most "shallow" process so we cannot cut off more. Axiom PR1 expresses this: a projection operator is invariant on the set of atomic actions.

The subscript n of the projection operator serves as a counter for the depth of a process. Axioms PR2 and PR3 illustrate how this counter can be decremented.

Axiom PR4 says that the projection operator distributes over the alternative composition: choosing an alternative does not alter the counter of the projection operator.

Proof rule We will discuss a proof rule expressing that a process is determined by its finite projections. This rule is due to [Bergstra and Klop, 1986].

Definition 2.4.1. Let x and y be processes. The *approximation induction principle* (AIP) is the following assumption. If for all $n \geq 1$ we have $\pi_n(x) = \pi_n(y)$ then $x = y$.

Remark 2.4.2. In the presence of recursion we will define a more restrictive version of this principle; see definition 2.4.28.

The following theorem states that projection operators can be eliminated from closed terms. To prove this we will use a method that we briefly explained in subsection 2.2.2. First, we define what we mean by the elimination of operators.

Definition 2.4.3. Let $L = (\Sigma, E)$ and $L_0 = (\Sigma_0, E_0)$ be two equational specifications with $\Sigma_0 \subseteq \Sigma$. If for all $s \in C(\Sigma)$ there is a $t \in C(\Sigma_0)$ such that $L \vdash s = t$ we say that L has the elimination property (for L_0).

Theorem 2.4.4. *The equational specification* BPA + PR *has the elimination property for* BPA.

Proof. It is not hard to show that the term rewriting system of table 8 is strongly normalizing with the lexicographical variant of the recursive path ordering that we treated in subsection 2.2.2. We confine ourselves to giving a partial ordering $<$ on the elements of the signature of BPA + PR.

$$+ < \cdot < \pi_1 < \pi_2 < \ldots .$$

Moreover, we give the sequential composition $\cdot$ the lexicographical status for the first argument. It is straightforward to show that the left-hand side of each rewrite rule is $>_{lpo}$ than its right-hand side. Now, apply theorem 2.2.18.

Now it is not hard to see that a normal form (with respect to the term rewriting system in table 8) of a closed BPA+PR term must be a basic BPA term, which proves the theorem. ∎

Next, we formulate a traditional theorem in process algebra. It states that the term rewriting system associated to the equational specification BPA + PR behaves neatly: it terminates modulo the equations without a clear direction (viz. the commutativity and the associativity of alternative composition) and it is confluent modulo these equations. In term rewriting theory, this is expressed by saying that the term rewriting system is complete. Incidentally, note that we proved in theorem 2.4.4 that the associated term rewriting system terminates, but not that it terminates modulo the equations A1 and A2.

Table 8. A term rewriting system for BPA + PR.

$x + x \to x$	RA3
$(x + y)z \to xz + yz$	RA4
$(xy)z \to x(yz)$	RA5
$\pi_n(a) \to a$	RPR1
$\pi_1(ax) \to a$	RPR2
$\pi_{n+1}(ax) \to a\pi_n(x)$	RPR3
$\pi_n(x + y) \to \pi_n(x) + \pi_n(y)$	RPR4

The main application of the next result is that it is usually used to prove the conservativity of BPA + PR over BPA (an extension is conservative if no new identities can be derived between original terms in the extended system). The proof of this term rewriting theorem requires much term rewriting theory, which is beyond the scope of this chapter. For more information on these term rewriting techniques we refer to [Jouannaud and Muñoz, 1984] and [Jouannaud and Kirchner, 1986]. Nevertheless, we want to mention the theorem anyway, since it has an importance of its own, for instance for implementational purposes. We will prove the conservativity with an alternative method that we will explain in the next subsection; see subsection 2.4.1.

Theorem 2.4.5. *The term rewriting system of table 8 is confluent modulo the equations A1 and A2. Therefore, it has unique normal forms modulo the equations A1 and A2.*

The next theorem states that for a closed term s the sequence

$$\pi_1(s), \pi_2(s), \ldots$$

converges to the term s itself. It is a nice example of the use of structural induction.

Proposition 2.4.6. *Let t be a closed* BPA + PR *term. Then there is an $n \geq 1$ such that for all $k \geq n$ we have* $\mathrm{BPA} + \mathrm{PR} \vdash \pi_k(t) = t$.

Proof. It suffices to prove this proposition for basic BPA terms (use theorem 2.4.4). We will use the technique of structural induction that we discussed on page 8. So, we will use the inductive definition of a basic term (2.2.4).

Let t be an atomic action. Take $n = 1$ and use axiom PR1.

Let $t = a \cdot s$ with $a \in A$ and s. Suppose that the proposition holds for s. Then there is an $n' \geq 1$ with $\pi_k(s) = s$ for all $k \geq n'$. Take $n = n' + 1$ and use axiom PR3.

Let $t = s+r$. Suppose that the proposition holds for s and r. Then there are n' and n'' with the desired properties for s and r. Take $n = \max(n', n'')$ and use axiom PR4. ∎

Semantics The semantics of BPA + PR can be given in the same way as the semantics for BPA.

Table 9. Derivation rules for projections.

$$\frac{x \xrightarrow{a} \surd}{\pi_n(x) \xrightarrow{a} \surd} \qquad \frac{x \xrightarrow{a} x'}{\pi_1(x) \xrightarrow{a} \surd} \qquad \frac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$$

We consider the term deduction system $T(\mathrm{BPA}+\mathrm{PR})$ with as signature the signature of BPA + PR and as deduction rules the rules in tables 5 and 9. Bisimulation equivalence is a congruence on the structured state system induced by $T(\mathrm{BPA} + \mathrm{PR})$; see 2.2.32. So the quotient of closed BPA + PR terms with respect to bisimulation equivalence is well-defined; that is, the operators of BPA+PR can be defined on this quotient by taking representatives. The following theorem states that this quotient is a model of BPA + PR.

Theorem 2.4.7. *The set of closed* BPA + PR *terms modulo bisimulation equivalence is a model of* BPA + PR *and* AIP.

Proof. Strong bisimulation equivalence is a congruence, so to prove the soundness of the axiomatization BPA+PR we just need to check the soundness of the separate axioms. The axioms A1–5 are treated exactly the same as in theorem 2.2.35. So we only need to check the axioms of table 7. The relation between $\pi_n(a)$ and a is a bisimulation, so PR1 holds. Axiom PR2 is treated analogously. Axioms PR3 and PR4 are treated as A1.

With proposition 2.4.6 we see that AIP holds. ∎

2.4.1 Conservativity

Here, we will explain how to prove the conservativity of BPA + PR over BPA without using theorem 2.4.5. We recall that the main application of theorem 2.4.5 is that we can prove that adding the projection operators does not yield new identities between BPA terms. This important property is called conservativity. We did not give a proof of theorem 2.4.5, but instead we will provide an alternative powerful method for proving the conservativity of BPA + PR over BPA (and all the other conservativity theorems in this chapter). This method is based on the format of the operational rules of both systems rather than on a term rewriting analysis. So, we will make a second expedition into the area of general theory on structured operational semantics and we will illustrate the theory with a running example. This example will yield, of course, the conservativity of BPA + PR over BPA. We will base ourselves on [Verhoef, 1994b].

First we formalize how we can join two given signatures.

Definition 2.4.8. Let Σ_0 and Σ_1 be two signatures. If for all operators $f \in \Sigma_0 \cap \Sigma_1$ the arity of f in Σ_0 is the same as its arity in Σ_1 then the sum of Σ_0 and Σ_1, notation $\Sigma_0 \oplus \Sigma_1$, is defined and is equal to the signature $\Sigma_0 \cup \Sigma_1$.

Example 2.4.9. Let $\Sigma_0 = A \cup \{+, \cdot\}$ and $\Sigma_1 = \Sigma_0 \cup \{\pi_n : n \geq 1\}$ be signatures. Then the sum $\Sigma_0 \oplus \Sigma_1$ is defined and equals the signature of BPA + PR. Note that these signatures are not disjoint.

Next, we define how to "add" two operational semantics.

Definition 2.4.10. Let $T^i = (\Sigma_i, D_i)$ be term deduction systems with predicate and relation symbols T_p^i and T_r^i respectively ($i = 0, 1$). Let $\Sigma_0 \oplus \Sigma_1$ be defined. The sum of T^0 and T^1, notation $T^0 \oplus T^1$, is the term deduction system $(\Sigma_0 \oplus \Sigma_1, D_0 \cup D_1)$ with predicate and relation symbols $T_p^0 \cup T_p^1$ and $T_r^0 \cup T_r^1$ respectively.

Example 2.4.11. Let T_0 be the term deduction system with Σ_0 of our running example and with the rules of table 5. Let T_1 be the term deduction system with Σ_1 of the running example and with deduction rules that can be found in table 9. The sum $T_0 \oplus T_1$ is defined and is the operational semantics of the theory comprising basic process algebra and projections: BPA + PR.

Next, we define what we will call operational conservativity. This definition is taken from [Verhoef, 1994b], but this notion is already defined by [Groote and Vaandrager, 1992] for the case without extra predicates on states.

Definition 2.4.12. Let $T^i = (\Sigma_i, D_i)$ be term deduction systems ($i = 0, 1$) with $T = (\Sigma, D) := T^0 \oplus T^1$ defined. Let $D = D(T_p, T_r)$. The term deduction system T is called an operationally conservative extension of T^0 if for all $s, u \in C(\Sigma_0)$, for all relation symbols $R \in T_r$ and predicate symbols $P \in T_p$, and for all $t \in C(\Sigma)$ we have

$$T \vdash sRt \iff T^0 \vdash sRt$$

and

$$T \vdash Pu \iff T^0 \vdash Pu.$$

Before we can continue with a theorem that gives sufficient conditions when a term deduction system is an operationally conservative extension of another such system, we need one more definition. This definition originates from [Groote and Vaandrager, 1992].

Definition 2.4.13. Let $T = (\Sigma, D)$ be a term deduction system and let F be a set of formulas. The variable dependency graph of F is a directed graph with variables occurring in F as its nodes. The edge $x \longrightarrow y$ is an edge of the variable dependency graph if and only if there is a relation $tRs \in F$ with $x \in var(t)$ and $y \in var(s)$.

The set F is called well-founded if any backward chain of edges in its variable dependency graph is finite. A deduction rule is called well-founded if its set of hypotheses is so. A term deduction system is called well-founded if all its deduction rules are well-founded.

Example 2.4.14. Definition 2.4.13 expresses that a rule is well-founded if the set of premises does not contain cyclic references to variables (in case this set is finite). So, for instance, if there is a premise xRx then there is a cyclic reference to the variable x. Also the two premises xRy and ySx comprise a cyclic reference to x. Since we do not have such premises in the operational semantics of BPA, it is not hard to verify that the deduction rules of table 5 are well-founded.

Now, we are in a position to state a theorem providing us with sufficient conditions so that a term deduction system is an operationally conservative extension of another one. This theorem is based on a more general theorem of [Verhoef, 1994b]. A more restrictive version of this theorem was first formulated by [Groote and Vaandrager, 1992].

Theorem 2.4.15. *Let $T^0 = (\Sigma_0, D_0)$ be a pure well-founded term deduction system in path format. Let $T^1 = (\Sigma_1, D_1)$ be a term deduction system in path format. If there is a conclusion sRt or Ps of a rule $d_1 \in D_1$ with $s = x$ or $s = f(x_1, \ldots, x_n)$ for an $f \in \Sigma_0$, we additionally require that d_1 is pure, well-founded, $t \in O(\Sigma_0)$ for premises tRy of d_1, and that there is a premise containing only Σ_0 terms and a new relation or predicate symbol. Now if $T = T^0 \oplus T^1$ is defined then T is an operationally conservative extension of T_0.*

Example 2.4.16. We already gave the definition of a pure term deduction system in definition 2.2.31. It is not hard to see that the term deduction system T_0 of our running example is pure. It is also not difficult to see that T_1 of our running example is in *path* format. Moreover, since there is no deduction rule in T_1 with an old function symbol or a variable on the vital position, we do not need to check the additional requirements. So, since the sum is defined we conclude with the above theorem that $T_0 \oplus T_1$ is an operationally conservative extension of T_0.

Now that we have the operational conservativity, we need to make the connection with the usual conservativity. Following [Verhoef, 1994b], henceforth we will call this well-known property equational conservativity to exclude possible confusion with the already introduced notion of operational conservativity. As an intermediate step, we will first define the notion of operational conservativity up to φ equivalence. Here, φ equivalence is some (semantical) equivalence that is defined in terms of relation and predicate symbols only. Strong bisimulation equivalence is an example of an equivalence that is definable exclusively in terms of relation and predicate symbols. This definition was first formulated by [Groote and Vaandrager, 1992] for the case of operational conservativity up to strong bisimulation equivalence. Roughly, if original terms s and t are bisimilar in the extended system, if and only if they are bisimilar in the original system we call the large system a conservative extension up to bisimulation equivalence of the original one. The next definition expresses this for any equivalence.

Definition 2.4.17. Let $T^i = (\Sigma_i, D_i)$ be term deduction systems ($i = 0, 1$) with $T = (\Sigma, D) := T^0 \oplus T^1$ defined. If we have for all $s, t \in C(\Sigma_0)$

$$s =_{\varphi}^{\oplus} t \iff s =_{\varphi}^{0} t$$

we say that T is an operationally conservative extension of T_0 up to φ equivalence, where φ is some semantical equivalence that is defined in terms of relation and predicate symbols only. By $s =_\varphi t$ we mean that s and t are in the same φ equivalence class. The superscripts $\oplus$ and 0 are to express the system in which this holds.

Remark 2.4.18. Many equivalences are definable in terms of relation and predicate symbols only: for instance, trace equivalence, completed trace equivalence, failure equivalence, readiness equivalence, failure trace equivalence, ready trace equivalence, possible future equivalence, simulation equivalence, complete simulation equivalence, ready simulation equivalence, nested simulation equivalence, strong bisimulation equivalence, weak bisimulation equivalence, η bisimulation equivalence, delay bisimulation equivalence, branching bisimulation equivalence, and more equivalences. We refer to Van Glabbeek's linear time – branching time spectra [Glabbeek, 1990; Glabbeek, 1993] for more information on these equivalences.

Next, we formulate a theorem stating that if a large system is an operationally conservative extension of a small system, then it is also an operationally conservative extension up to any equivalence that is definable in terms of relation and predicate symbols only. This theorem is taken from [Verhoef, 1994b]. This theorem was formulated by [Groote and Vaandrager, 1992] for the case of strong bisimulation equivalence.

Theorem 2.4.19. *Let $T^i = (\Sigma_i, D_i)$ be term deduction systems ($i = 0, 1$) and let $T = T^0 \oplus T^1$ be defined. If T is an operationally conservative extension of T_0 then it is also an operationally conservative extension up to φ equivalence, where φ is an equivalence relation defined exclusively in terms of predicate and relation symbols.*

Example 2.4.20. For our running example it will be clear that the term deduction system of the sum $T_0 \oplus T_1$ is an operationally conservative extension up to strong bisimulation equivalence of the base system T_0.

Now that we have the intermediate notion of operational conservativity up to some equivalence, we will come to the well-known notion that in this chapter we will call equational conservativity. We recall that an equational specification is a pair consisting of a signature and a set of equations over this signature. First we define how we combine equational specifications into larger ones.

Definition 2.4.21. Let $L_i = (\Sigma_i, E_i)$ be equational specifications ($i = 0, 1$). Let $\Sigma_0 \oplus \Sigma_1$ be defined. Then the sum $L_0 \oplus L_1$ of L_0 and L_1 is the equational specification $(\Sigma_0 \oplus \Sigma_1, E_0 \cup E_1)$.

Example 2.4.22. Let L_0 be the equational specification that consists of the signature Σ_0 of our running example and the equations E_0 of BPA that we already listed in table 1: the axioms A1–A5. Let L_1 be the equational specification with as signature Σ_1 of our running example and with equations that we presented in table 7: PR1–PR4. Now the sum $L_0 \oplus L_1$ is defined and equals the equational specification that we baptized BPA + PR.

Next, we define the notion of equational conservativity.

Definition 2.4.23. Let $L_i = (\Sigma_i, E_i)$ be equational specifications ($i = 0, 1$) and let $L = L_0 \oplus L_1$ be defined. We say that L is an equationally conservative extension, or simply a conservative extension of L_0, if for all $s, t \in C(\Sigma_0)$

$$L \vdash s = t \iff L_0 \vdash s = t.$$

We recall that $\vdash$ stands for derivability in equational logic as defined in definition 2.2.2.

Now we have all the prerequisites to formulate the equational conservativity theorem. This theorem is taken from [Verhoef, 1994b].

Theorem 2.4.24. *Let $L_i = (\Sigma_i, E_i)$ be equational specifications ($i = 0, 1$) and let $L = (\Sigma, E) = L_0 \oplus L_1$ be defined. Let $T_i = (\Sigma_i, D_i)$ be term deduction systems and let $T = T_0 \oplus T_1$. Let φ be an equivalence that is definable in terms of predicate and relation symbols only. Let E_0 be a complete axiomatization with respect to the φ equivalence model induced by T_0 and let E be a sound axiomatization with respect to the φ equivalence model induced by T. If T is an operationally conservative extension of T_0 up to φ equivalence then L is an equationally conservative extension of L_0.*

Now, we can apply the equational conservativity theorem to prove the conservativity of BPA + PR over BPA.

Theorem 2.4.25. *If t and s are closed* BPA *terms, then we have*

$$\mathrm{BPA} \vdash t = s \iff \mathrm{BPA} + \mathrm{PR} \vdash t = s.$$

Proof. On the way to this proof we checked all the conditions of the theorem in the example paragraphs except for the soundness and completeness of BPA and the soundness of BPA + PR. Fortunately, we already proved these conditions. The soundness and completeness of BPA is proved in theorems 2.2.35 and 2.2.37 respectively and the soundness of BPA + PR is proved in theorem 2.4.7. So, we can apply theorem 2.4.24 and conclude that BPA + PR is an equationally conservative extension of BPA. ∎

Now that we have the conservativity of BPA + PR, we can immediately prove its completeness from the completeness of the subsystem BPA. We will not do this directly, but we will formulate a general completeness theorem that can be found in [Verhoef, 1994b]; it is a simple corollary of the equational conservativity theorem. This completeness theorem states that the combination of conservativity, elimination of extra operators, and the completeness of the subsystem yields the completeness of the extension. For the formulation of the next theorem, we stick to the notations and assumptions stated in theorem 2.4.24.

Theorem 2.4.26. *If in addition to the conditions of theorem 2.4.24 the equational specification L has the elimination property for L_0 (see definition 2.4.3) then we have that E is a complete axiomatization with respect to the φ equivalence model induced by the term deduction system T.*

Theorem 2.4.27. *The axiom system* BPA + PR *is a complete axiomatization of the set of closed* BPA + PR *terms modulo bisimulation equivalence.*

Proof. We apply theorem 2.4.26. We already know that the conditions of the conservativity theorem are satisfied. So we only need to check the additional one. According to theorem 2.4.4 the elimination condition is satisfied. So the conditions of theorem 2.4.26 are satisfied and we are done. ∎

2.4.2 Recursion and projection

In this subsection we will add recursion and projections to the theory BPA.

We will define the equational specification BPArec + PR by means of a combination of BPArec and BPA + PR.

The signature of BPArec + PR consists of the signature of BPArec plus for each $n \geq 1$ a unary function π_n (projections). The axioms of BPArec + PR are the axioms of BPArec and the axioms of table 7.

Proof rule We will discuss a proof rule expressing that a process that can be specified with the aid of a guarded recursive specification is determined by its finite projections. This rule is a restricted version of the rule that is defined in definition 2.4.1 and is also due to [Bergstra and Klop, 1986].

Definition 2.4.28. Let x and y be processes such that x or y (or both) can be specified with the aid of a guarded recursive specification. The *restricted approximation induction principle* (AIP^-) is the following assumption. If for all $n \geq 1$ we have $\pi_n(x) = \pi_n(y)$ then $x = y$.

Remark 2.4.29. The principle AIP^- is more restrictive than necessary. A more general approximation induction principle is defined in [Glabbeek, 1987].

Theorem 2.4.30. *Let x be a solution of a guarded recursive specification. Then $\pi_n(x)$ can be rewritten into a closed* BPA *term for all $n \geq 1$.*

Proof. Without loss of generality we may assume that x is a solution of a completely guarded recursive specification. Let the right-hand side of the recursion equation of x be called s_1. Suppose that we have defined s_n. Then we obtain s_{n+1} as follows: substitute for each variable in s_n the right-hand side of its recursion equation. It is not hard to see that for every $n \geq 1$

we have $\pi_n(x) = \pi_n(s_n)$ and that the latter term can be rewritten into a closed BPA term. ∎

The following corollary is called *the projection theorem.*

Corollary 2.4.31. *Suppose that we have two solutions x_1 and x_2 of a guarded recursive specification belonging to the same recursion equation. Then for all $n \geq 1 : \pi_n(x_1) = \pi_n(x_2)$.*

Proof. This follows immediately from the proof of theorem 2.4.30. ∎

Theorem 2.4.32. *The principle* AIP$^-$ *implies the principle* RSP.

Proof. This follows immediately from corollary 2.4.31. ∎

Semantics The semantics of BPArec + PR is given by means of a term deduction system T(BPArec+PR) with as signature the signature of BPArec+ PR and as rules the rules in tables 5, 6, and 9. Bisimulation equivalence is a congruence on the structured state system L(BPArec + PR) induced by T(BPArec+PR); see 2.2.32. So the quotient of closed BPArec+PR terms modulo strong bisimulation equivalence is well-defined; that is, the operators of BPArec + PR can be defined on this quotient by taking representatives. The next theorem states that this quotient is a model of BPArec+PR.

Theorem 2.4.33. *The set of closed* BPArec + PR *terms modulo bisimulation equivalence is a model of* BPArec + PR.

Proof. Since strong bisimulation equivalence is a congruence, we only need to verify the soundness of each axiom. This has been done in the proofs of theorems 2.3.13 and 2.4.7. ∎

Theorem 2.4.34. *The model of closed* BPArec + PR *terms modulo bisimulation equivalence satisfies* RDP, AIP$^-$, *and* RSP.

Proof. The principle RDP is proved as in theorem 2.2.35. With the aid of theorem 2.4.32 it suffices to prove that the model satisfies AIP$^-$. This proof is taken from [Glabbeek, 1987].

So let x and y be closed BPArec + PR terms such that for all $n \geq 1$ we have $\pi_n(x) \sim \pi_n(y)$ (the symbol $\sim$ stands for the bisimulation relation). Suppose that y can be specified with the aid of a guarded recursive specification. We have to prove that x and y are bisimilar. We relate u and v, notation $u\ R\ v$, if and only if v can be specified with the aid of a guarded recursive specification and if for all $n \geq 1 : \pi_n(u) \sim \pi_n(v)$. Note that $x\ R\ y$. We will prove that R is a bisimulation relation. So we have to distinguish three cases.

Suppose that $u\ R\ v$ and $u \xrightarrow{a} u'$. Define for $n \geq 1$

$$S_n = \{v^* \mid v \xrightarrow{a} v^*,\ \pi_n(u') \sim \pi_n(v^*)\}.$$

To prove that there is a v' with $v \xrightarrow{a} v'$ and $u'\ R\ v'$ it suffices to show that the intersection of the S_n contains an element. Firstly, every S_n contains an element since $\pi_{n+1}(u) \sim \pi_{n+1}(v)$. Secondly, each S_n is finite. For v can be specified with the aid of a guarded recursive specification, so there is a completely guarded term t with $v = t$. We can rewrite this term using the rewrite rules RA3, RA4, and RA5 of table 8 to a sum of terms. The summands are either atomic actions or products $t' \cdot t''$ and t' is not a sum or a product. Since t is completely guarded it must be an atomic action. So v has the form

$$v = \sum_{i=1}^{n} a_i \cdot v_i + \sum_{j=1}^{m} b_j.$$

This means that the set S_n is finite. Thirdly, we have $S_{n+1} \subseteq S_n$ for all $n \geq 1$, since $\pi_{n+1}(u') \sim \pi_{n+1}(v^*)$ implies $\pi_n(u') \sim \pi_n(v^*)$. From these three observations we can conclude that the sequence $S_1, S_2, \ldots$ must remain constant from some index onwards. Thus, the intersection of the S_n is not empty.

Now suppose $u \; R \; v$ and $v \xrightarrow{a} v'$. Define, as above, for all $n \geq 1$

$$S_n = \{u^* \mid u \xrightarrow{a} u^*,\ \pi_n(u^*) \sim \pi_n(v')\}.$$

We can again observe that every S_n contains an element and that the sequence $S_1, S_2, \ldots$ is decreasing (but not that each S_n is finite). So we can choose for each $n \geq 1$ an element $u_n \in S_n$. By the first part of the proof we know that there are v_n with $v \xrightarrow{a} v_n$ and $u_n \; R \; v_n$. But since v can be specified with the aid of a guarded recursive specification there is a v^* that occurs infinitely many times in the sequence $v_1, v_2, \ldots$. Let $v^* = v_k$ for some index k. We show that $u_k \; R \; v'$, which proves the second case. So fix an $n \geq 1$. Then there is an index $m > n$ with $v^* = v_m$ and we find $\pi_n(u_m) \sim \pi_n(v')$, since $u_m \in S_m \subseteq S_n$. Moreover, we have $u_k \; R \; v^*$ and $u_m \; R \; v^*$. So $\pi_n(u_k) \sim \pi_n(u_m)$ and we find $\pi_n(u_k) \sim \pi_n(v')$. So we have $u_k \; R \; v'$, since $n \geq 1$ was arbitrarily chosen.

Finally, note that if $u \; R \; v$ then we have

$$u \xrightarrow{a} \surd \iff \pi_2(u) \xrightarrow{a} \surd \iff \pi_2(v) \xrightarrow{a} \surd \iff v \xrightarrow{a} \surd.$$

So R is a bisimulation. This finishes the proof. ∎

Theorem 2.4.35. *The principle* AIP *does not hold in the model of closed* BPArec + PR *terms modulo bisimulation equivalence.*

Proof. Consider the two recursive specifications that we defined in example 2.3.2. Let $x = \langle X \mid E_1 \rangle$ and $y = \langle Y \mid E_2 \rangle$. With the aid of figure 5 we can see that for all $n \geq 1$ we have that $\pi_n(x+y) \sim \pi_n(x)$ but that $x+y \not\sim x$. Note that E_1 is not guarded so $x + y$ and x are not specified with the aid of a guarded recursive specification. ∎

The following theorem concerns the theory BPArec. The result was already stated in theorem 2.3.13. We postponed the proof of this until now, since we want to use the fact that RSP holds in BPArec + PR.

Theorem 2.4.36. *The model of closed* BPArec *terms modulo bisimulation equivalence satisfies* RSP.

Proof. Let $E(V)$ be a guarded recursive specification and suppose that we have two solutions, say x and y, belonging to the same recursion equation. We have to show that $x \sim y$. Since x and y are also BPArec + PR terms we find with the aid of theorem 2.4.34 that $x \sim y$, which proves the theorem. ∎

2.5 Deadlock

Usually deadlock stands for a process that has stopped its executions and cannot proceed. In this subsection we will extend the theory BPA with a process named *deadlock*. We can distinguish between successful and unsuccessful termination in the presence of deadlock. This subsection is based on [Bergstra and Klop, 1984a; Bergstra and Klop, 1995].

Let $x \cdot y$ be a sequential composition of two processes. The process y starts if x has terminated. But if x reaches a state of inaction due to deadlock, we do not want y to start: we want it only to start when x terminates successfully. We will axiomatize the behaviour of a deadlocked process called δ in table 10.

The signature of the equational specification BPA_δ is the signature of BPA extended with a constant $\delta \notin A$ called deadlock, or inaction. The axioms of the equational specification BPA_δ are the axioms of BPA in table 1 plus the two axioms in table 10 (A6 and A7). We will discuss them now.

Equation A6 expresses that δ is a neutral element with respect to the alternative composition; it says that no deadlock will occur as long as there is an alternative that can proceed. Axiom A7 says that the constant δ is a left-zero element for the sequential composition. It says that after a deadlock has occurred, no other actions can possibly follow. Actually, *inaction* would be a better name for the constant δ, for a process $a+\delta$ ($a \in A$) contains no deadlock. Deadlock only occurs if there is no alternative to δ, as in $a \cdot \delta$.

So using the process δ we can distinguish between successful and unsuccessful termination. Thus, the process $a\delta$ terminates unsuccessfully whereas a terminates successfully.

Structural induction In BPA_δ we can use the technique of structural induction just like in BPA (compare page 8). We will adjust the definition of a basic term (see definition 2.2.4) and we will mention the result that every closed term over BPA_δ can be written as a basic term.

Definition 2.5.1. The constant δ is a basic term over BPA_δ. An atomic action is a basic term. If t is a basic term and $a \in A$, then $a \cdot t$ is a basic term. If t and s are basic terms, then $t + s$ is a basic term.

We recall that a closed term over BPA_δ is a BPA_δ term without variables.

Remark 2.5.2. If we consider terms t and s to be identical if

$$A1, A2 \vdash t = s,$$

we can see that basic terms have the following form:

$$\sum_{i=1}^{n} a_i \cdot t_i + \sum_{j=1}^{m} b_j,$$

where $a_i \in A$, $b_j \in A \cup \{\delta\}$, $1 \leq i \leq n, 1 \leq j \leq m$, and $n + m \geq 1$.

Proposition 2.5.3. *Let t be a closed* BPA_δ *term. Then there is a basic term s such that* $\mathrm{BPA}_\delta \vdash t = s$.

Proof. The proof of this proposition can be given along the same lines as the proof of proposition 2.2.6. ∎

Semantics As usual, we give the semantics by means of a term deduction system. Take for the signature of $T(\mathrm{BPA}_\delta)$ the signature of BPA_δ and for the set of rules just the ones of BPA of table 5. Since bisimulation equivalence is a congruence (2.2.32), the quotient of closed BPA_δ terms modulo bisimulation equivalence is well-defined so the operators of BPA_δ can be defined on this quotient using representatives. This quotient is a model of BPA_δ.

Theorem 2.5.4. *The set of closed* BPA_δ *terms modulo bisimulation equivalence is a model of* BPA_δ.

Proof. It suffices to check the soundness of each axiom, since bisimulation equivalence is a congruence. Axioms A1–A5 are treated as in theorem 2.2.35 since there are no transitions for δ. Axiom A6 is treated as A1. For axiom A7 take the relation that only relates δx and δ. ∎

Theorem 2.5.5. *The axiom system* BPA_δ *is a complete axiomatization of the set of closed* BPA_δ *terms modulo bisimulation equivalence.*

Proof. Since there are no new transitions for the constant δ, this is proved as theorem 2.2.37. ∎

Table 10. Deadlock.

$x + \delta = x$	A6
$\delta x = \delta$	A7

2.5.1 Extensions of BPA_δ

In this subsection we will discuss the extensions of BPA_δ with recursion and/or projections.

Recursion We can add recursion to BPA_δ in exactly the same way as we did for BPA. The equational specification $\mathrm{BPA}_\delta\mathrm{rec}$ contains the signature of BPArec and a constant $\delta \notin A$. The axioms of $\mathrm{BPA}_\delta\mathrm{rec}$ are the axioms of BPArec plus the axioms of table 10.

Note that $\delta \notin A$ so it cannot serve as a guard: δX is not completely guarded but it is guarded since $\delta X = \delta$.

The semantics of $\mathrm{BPA}_\delta\mathrm{rec}$ can be given by means of the term deduction system $T(\mathrm{BPA}_\delta\mathrm{rec})$ that has as its signature the signature of $\mathrm{BPA}_\delta\mathrm{rec}$ and as rules the rules of tables 5 and 6. Since bisimulation equivalence is a congruence (2.2.32), the operators of $\mathrm{BPA}_\delta\mathrm{rec}$ can be defined on the quotient algebra of closed $\mathrm{BPA}_\delta\mathrm{rec}$ terms with respect to bisimulation equivalence. This quotient is a model of $\mathrm{BPA}_\delta\mathrm{rec}$ and it satisfies RDP. To prove this, combine the proofs of theorems 2.3.13 and 2.5.4. Moreover this model satisfies RSP, which can be proved when we combine $\mathrm{BPA}_\delta\mathrm{rec}$ with projections.

Projection We can extend BPA_δ with projections in exactly the same way as we did for BPA. The equational specification $\mathrm{BPA}_\delta + \mathrm{PR}$ has as its signature the signature of $\mathrm{BPA} + \mathrm{PR}$ and a constant $\delta \notin A$. Its axioms are the axioms of $\mathrm{BPA} + \mathrm{PR}$ and the ones concerning deadlock (tables 1, 7, and 10). We assume that a ranges over $A \cup \{\delta\}$ in table 7 on projections.

The following theorem states that projection operators can be eliminated from closed terms.

Theorem 2.5.6. *For every closed* $\mathrm{BPA}_\delta + \mathrm{PR}$ *term* t *there is a basic* BPA_δ *term* s *such that* $\mathrm{BPA}_\delta + \mathrm{PR} \vdash t = s$.

Proof. Use the term rewriting system that consists of the rules in table 8 and in addition the rewrite rule $\delta \cdot x \to \delta$ and show that this term rewriting system is terminating. Hint: use theorem 2.2.18. The rest of the proof is straightforward and therefore omitted. ∎

Proposition 2.5.7. *Let* t *be a closed* $\mathrm{BPA}_\delta + \mathrm{PR}$ *term. Then there is an* $n \geq 1$ *such that for all* $k \geq n$ *we have* $\mathrm{BPA}_\delta + \mathrm{PR} \vdash \pi_k(t) = t$.

Proof. With theorem 2.5.6 it suffices to prove the proposition for basic BPA_δ terms t. So we can use structural induction on t to prove the proposition. ∎

The semantics of $\mathrm{BPA}_\delta + \mathrm{PR}$ can be given with the term deduction system $T(\mathrm{BPA}_\delta + \mathrm{PR})$ that has as its signature the signature of $\mathrm{BPA}_\delta + \mathrm{PR}$ and as rules the rules of tables 5 and 9. Since bisimulation equivalence is a congruence (2.2.32), it follows that the operators of $\mathrm{BPA}_\delta + \mathrm{PR}$ can be defined on the quotient algebra of closed $\mathrm{BPA}_\delta + \mathrm{PR}$ terms with respect to bisimulation equivalence. This quotient is a model of $\mathrm{BPA}_\delta + \mathrm{PR}$ and it satisfies AIP. To prove this, combine the proofs of theorems 2.4.7 and 2.5.4. Moreover, according to theorem 2.4.24 we have that $\mathrm{BPA}_\delta + \mathrm{PR}$ is a conservative extension of BPA_δ. So with theorem 2.4.26 we find that $\mathrm{BPA}_\delta + \mathrm{PR}$ is a complete axiomatization of this model (use also theorem 2.5.6).

Recursion and projection Here we extend BPA_δ with recursion and projections. The equational specification $\mathrm{BPA}_\delta\mathrm{rec} + \mathrm{PR}$ has as its signature the signature of $\mathrm{BPA}_\delta\mathrm{rec}$ plus for each $n \geq 1$ a unary function π_n. The axioms are the axioms of $\mathrm{BPA}_\delta\mathrm{rec}$ and the axioms concerning projections (table 7). We assume that a ranges over $A \cup \{\delta\}$ in this table.

The standard facts (and their proofs) of subsection 2.4.2 are easily transposed to the present situation. Their translation is, in short: every projection of a solution of a guarded recursive specification can be rewritten into a closed BPA_δ term; the projection theorem 2.4.31 holds; and AIP^- implies RSP.

The semantics of $\mathrm{BPA}_\delta\mathrm{rec} + \mathrm{PR}$ is given by means of a term deduction system $T(\mathrm{BPA}_\delta\mathrm{rec}+\mathrm{PR})$. Its signature is the signature of $\mathrm{BPA}_\delta\mathrm{rec}+\mathrm{PR}$ and its rules are those of $T(\mathrm{BPA}_\delta+\mathrm{PR})$ plus the rules concerning recursion that are presented in table 6. Since bisimulation equivalence is a congruence, the quotient algebra of closed $\mathrm{BPA}_\delta\mathrm{rec}+\mathrm{PR}$ terms with respect to bisimulation equivalence is well-defined, and the operators of $\mathrm{BPA}_\delta\mathrm{rec} + \mathrm{PR}$ can be defined on this quotient. This quotient is a model of $\mathrm{BPA}_\delta\mathrm{rec} + \mathrm{PR}$ and it satisfies RDP, RSP, and AIP^-, but not its unrestricted version AIP. We can prove that $\mathrm{BPA}_\delta\mathrm{rec}$ satisfies RSP. This is proved in the same way as theorem 2.4.36.

2.6 Empty process

In many situations it is useful to have a constant process that stands for immediate successful termination. In this subsection we will extend the equational specifications BPA and BPA_δ with a process that is only capable of terminating successfully. We will call such a process the *empty process* and we will denote it by ε. This constant originates from [Koymans and Vrancken, 1985]. Another reference to this constant is [Vrancken, 1986].

The empty process is a counterpart of the process deadlock. The process deadlock stands for immediate *unsuccessful* termination while the empty process stands for immediate *successful* termination. Moreover, the combi-

Table 11. Empty process.

$x\varepsilon = x$	A8
$\varepsilon x = x$	A9

nation of the two axioms A8 and A9 of table 11 express that ε is a neutral element with respect to the *sequential* composition whereas axioms A1 and A6 express that δ is a neutral element with respect to the *alternative* composition. Note that successful termination (not in a sum context) after the execution of at least one action can already be expressed in systems without ε, as $a \cdot \varepsilon = a$ ($a \in A$).

The equational specifications BPA_ε and $\mathrm{BPA}_{\delta\varepsilon}$ are defined as follows.

The signature of BPA_ε consists of the signature of BPA extended with a constant $\varepsilon \notin A$ called the empty process. The equations of BPA_ε are the axioms of BPA and the axioms A8 and A9 of table 11.

The signature of $\mathrm{BPA}_{\delta\varepsilon}$ consists of the signature of BPA_δ extended with a constant $\varepsilon \notin A \cup \{\delta\}$. The axioms of $\mathrm{BPA}_{\delta\varepsilon}$ are the ones of BPA_δ plus A8 and A9.

Structural induction In BPA_ε and $\mathrm{BPA}_{\delta\varepsilon}$ we can use the technique of structural induction just like in BPA or BPA_δ since every closed term can be written as a basic term. We will adjust the definition of a basic term to the present situation and we will mention that closed terms over BPA_ε or $\mathrm{BPA}_{\delta\varepsilon}$ can be written as basic terms.

Definition 2.6.1. A basic term over BPA_ε is defined as follows.

An atomic action is a basic term over BPA_ε. The constant ε is a basic term over BPA_ε. If t is a basic term over BPA_ε and $a \in A$, then $a \cdot t$ is a basic term over BPA_ε. If t and s are basic terms over BPA_ε, then $t + s$ is a basic term over BPA_ε.

A basic term over $\mathrm{BPA}_{\delta\varepsilon}$ is defined as follows.

An atomic action is a basic term over $\mathrm{BPA}_{\delta\varepsilon}$. The constants δ and ε are basic terms over $\mathrm{BPA}_{\delta\varepsilon}$. If t is a basic term over $\mathrm{BPA}_{\delta\varepsilon}$ and $a \in A$, then $a \cdot t$ is a basic term over $\mathrm{BPA}_{\delta\varepsilon}$. If t and s are basic terms over $\mathrm{BPA}_{\delta\varepsilon}$, then $t + s$ is a basic term over $\mathrm{BPA}_{\delta\varepsilon}$.

We recall that a closed term over $\mathrm{BPA}_\varepsilon/\mathrm{BPA}_{\delta\varepsilon}$ is a $\mathrm{BPA}_\varepsilon/\mathrm{BPA}_{\delta\varepsilon}$ term without variables.

Remark 2.6.2. If we consider terms identical that only differ in the order of the summands, basic terms over BPA_ε or $\mathrm{BPA}_{\delta\varepsilon}$ are of the form

$$\sum_{i=1}^{n} a_i \cdot t_i + \sum_{j=1}^{m} b_j,$$

where $a_i \in A$, $b_j \in A \cup \{(\delta,)\varepsilon\}$, $1 \leq i \leq n, 1 \leq j \leq m$, and $n + m \geq 1$.

Proposition 2.6.3. *Let t be a closed* $\mathrm{BPA}_\varepsilon/\mathrm{BPA}_{\delta\varepsilon}$ *term. Then there is a basic term s such that* $\mathrm{BPA}_\varepsilon/\mathrm{BPA}_{\delta\varepsilon} \vdash t = s$.

Proof. The proof of this proposition can be given along the same lines as the proof of proposition 2.2.6. ∎

Semantics We give the semantics of BPA_ε and $\mathrm{BPA}_{\delta\varepsilon}$ by means of term deduction systems. We handle both cases at the same time. Take for the signature of $T(\mathrm{BPA}_{(\delta)\varepsilon})$ the signature of $\mathrm{BPA}_{(\delta)\varepsilon}$ and for the set of rules the ones that are presented in table 12. This operational semantics is taken from [Baeten and Glabbeek, 1987].

The term deduction systems that we consider here differ from the ones that we treated before: instead of successful termination predicates $\cdot \xrightarrow{a} \surd$ we now have a termination option predicate; it is denoted postfix: $\cdot\downarrow$. Since they both are unary predicates on states we can still use the general theory on structured operational semantics that we treated in subsection 2.2.3. In particular we can use theorem 2.2.32 to prove that bisimulation equivalence is a congruence. So, the quotient algebra of closed $\mathrm{BPA}_{(\delta)\varepsilon}$ terms with respect to bisimulation equivalence is well-defined for the operators of $\mathrm{BPA}_{(\delta)\varepsilon}$. This quotient is a model for $\mathrm{BPA}_{(\delta)\varepsilon}$.

Theorem 2.6.4. *The set of closed* $\mathrm{BPA}_{(\delta)\varepsilon}$ *terms modulo bisimulation equivalence is a model of* $\mathrm{BPA}_{(\delta)\varepsilon}$.

Proof. Easy. A1–A7 are treated as usual, A8 as A5, and A9 as A1. ∎

Theorem 2.6.5. *The axiom system* $\mathrm{BPA}_{(\delta)\varepsilon}$ *is a complete axiomatization of the set of closed* $\mathrm{BPA}_{(\delta)\varepsilon}$ *terms modulo bisimulation equivalence.*

Proof. This is proved along the same lines as theorem 2.2.37 if we rephrase lemma 2.2.36 as follows. First, redefine the function n: $n(\varepsilon) = n(\delta) = 1$, $n(a) = 2$ for all $a \in A$, and $n(x + y) = n(xy) = n(x) + n(y)$. Secondly, replace the first case of 2.2.36 by $T(\mathrm{BPA}_{(\delta)\varepsilon}) \vdash x\downarrow \implies \mathrm{BPA}_{(\delta)\varepsilon} \vdash x = \varepsilon + x$. ∎

2.6.1 Conservativity

In this subsection we will explain how to prove that BPA_ε is a conservative extension of BPA. We cannot immediately use the theory of subsection 2.4.1, since the operational semantics of BPA_ε presented in table 12 is not an operationally conservative extension of the operational semantics of BPA that we listed in table 5. For we can prove in the extended system that $a \xrightarrow{a} \varepsilon$, whereas in the subsystem we can prove that $a \xrightarrow{a} \surd$. So we can

Table 12. Derivation rules of $T(\mathrm{BPA}_{(\delta)\varepsilon})$.

$$a \xrightarrow{a} \varepsilon \qquad \varepsilon\downarrow$$

$$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x\downarrow}{(x+y)\downarrow} \qquad \frac{y\downarrow}{(x+y)\downarrow}$$

$$\frac{x \xrightarrow{a} x'}{xy \xrightarrow{a} x'y} \qquad \frac{x\downarrow,\, y \xrightarrow{a} y'}{xy \xrightarrow{a} y'} \qquad \frac{x\downarrow,\, y\downarrow}{(xy)\downarrow}$$

"reach" a new term if we start with an original term. A possible solution for this problem is to give an alternative operational semantics for BPA_ε than the one that we present in table 12.

The special behaviour of the constant ε is expressed in the operational semantics of BPA_ε as it is presented in table 12. Another possibility is to express the special behaviour of the empty process with the aid of the equivalence relation. A well-known example of this kind is observational congruence due to [Milner, 1980]. There, Milner's silent action τ is treated as a normal atomic action in the operational rules and its special behaviour is expressed with the equivalence relation: observational congruence. In the case of the empty process a similar approach is reported on by [Koymans and Vrancken, 1985]. In that paper a graph model was constructed featuring the empty process as an ordinary atomic action. A notion called ε bisimulation was defined to express the special behaviour of the empty process. With the approach of [Koymans and Vrancken, 1985] we can use the theory of subsection 2.4.1 to prove the conservativity of BPA_ε over BPA. We will sketch the idea and leave the details as an exercise to the interested reader. For the operational rules we just take the operational semantics of BPA where we let a also range over ε. This means that we have, for instance, the rule $\varepsilon \xrightarrow{\varepsilon} \surd$. Note that this adds a new relation $\xrightarrow{\varepsilon}$ and a new predicate $\xrightarrow{\varepsilon}\surd$ to the operational rules for BPA. Now with the aid of theorem 2.4.15 it is not hard to see that this term deduction system is an operationally conservative extension of the term deduction system in table 5. By way of an example we will check the conditions of theorem 2.4.15 for one deduction rule in the extended system:

$$\frac{x \xrightarrow{\varepsilon} x'}{x + y \xrightarrow{\varepsilon} x'}.$$

The crucial place to look at is the left-hand side of the conclusion: $x + y$. There an original function symbol occurs: $+$. Now we need to check that this rule is pure and well-founded. This is easy. Also the terms x and x' must be original terms; this is the case since they are variables. And there must be a premise containing only original terms and a new relation or predicate symbol. This is also the case. The other rules are treated equally simply. So, we may apply theorem 2.4.15 and find the operational conservativity. Now this notion termed ε bisimulation can be defined exclusively in terms of relation and predicate symbols. So with theorem 2.4.19 we find that the term deduction system belonging to BPA_ε is an operationally conservative extension up to ε bisimulation equivalence of the term deduction system belonging to BPA (note that ε bisimulation becomes normal strong bisimulation for BPA where no ε is present). Now we can apply the equational conservativity theorem 2.4.24 if we know in addition that the model induced by the operational rules modulo ε bisimulation equivalence is sound with respect to the axioms of BPA_ε that we listed in tables 1 and 11. This is shown for the graph model by [Koymans and Vrancken, 1985] and this proof transposes effortlessly to the situation with operational rules that we sketched above. This proves that BPA_ε is a conservative extension of BPA.

2.6.2 Extensions of $\mathrm{BPA}_{(\delta)\varepsilon}$

In this subsection we will discuss the extensions of $\mathrm{BPA}_{(\delta)\varepsilon}$ with recursion and/or projections.

Recursion We can add recursion to $\mathrm{BPA}_{(\delta)\varepsilon}$ in exactly the same way as we did for $\mathrm{BPA}_{(\delta)}$. The equational specification $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ contains the signature of BPArec and $(\delta,)\varepsilon \notin A$. The axioms are the ones of BPArec and the axioms of table 11 (and table 10).

Since $\delta, \varepsilon \notin A$, they cannot serve as a guard. For instance, εX is neither completely guarded nor guarded.

The semantics of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ can be given by means of a term deduction system $T(\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec})$ that has as its signature the signature of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ and as its rules the ones of $T(\mathrm{BPA}_{(\delta)\varepsilon})$ plus the rules of table 13. Since bisimulation equivalence is a congruence (2.2.32), we can define the operators of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ on the quotient algebra of closed $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ terms with respect to bisimulation equivalence. This quotient is a model of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ and it satisfies RDP and RSP.

Projection We extend the theory $\mathrm{BPA}_{(\delta)\varepsilon}$ with projections. The equational specification $\mathrm{BPA}_{(\delta)\varepsilon}+\mathrm{PR}$ has as its signature the one of $\mathrm{BPA}_{(\delta)}+\mathrm{PR}$ plus a constant $\varepsilon \notin A$ (and $\varepsilon \neq \delta$). The axioms of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ are the axioms of $\mathrm{BPA}_{(\delta)} + \mathrm{PR}$ plus the axioms of table 11. Moreover, we assume for axiom PR1 (table 7) that a may also be ε.

The results that we inferred for $\mathrm{BPA}_{(\delta)} + \mathrm{PR}$ also hold for $\mathrm{BPA}_{(\delta)\varepsilon} +$ PR: we can eliminate projections occurring in closed terms and the sequence $\pi_1(t), \pi_2(t), \ldots$ has $t, t, \ldots$ as its tail. We can also prove many conservativity results using subsection 2.4.1. We can, for instance, show that $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ is conservative over $\mathrm{BPA}_{(\delta)\varepsilon}$. We already showed that $\mathrm{BPA}_{(\delta)\varepsilon}$ is a conservative extension of BPA, so with transitivity we find that $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ is a conservative extension of BPA. We use the transitivity argument here since the proof that $\mathrm{BPA}_{(\delta)\varepsilon}$ is a conservative extension of BPA uses another semantics. See subsection 2.6.1 for more information.

The semantics of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ can be given by means of a term deduc-

Table 13. Derivation rules for recursion and empty process.

$\dfrac{\langle s_X \mid E\rangle \downarrow}{\langle X \mid E\rangle \downarrow}$	$\dfrac{\langle s_X \mid E\rangle \xrightarrow{a} y}{\langle X \mid E\rangle \xrightarrow{a} y}$

Table 14. Derivation rules for projections with empty process.

$\dfrac{x\downarrow}{\pi_n(x)\downarrow}$	$\dfrac{x \xrightarrow{a} x'}{\pi_1(x) \xrightarrow{a} \varepsilon}$	$\dfrac{x \xrightarrow{a} x'}{\pi_{n+1}(x) \xrightarrow{a} \pi_n(x')}$

tion system $T(\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR})$. Its signature is the signature of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ and its rules are the rules of tables 12 and 14. We can define the quotient algebra of closed $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ terms with respect to bisimulation equivalence and the operators of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ as usual. The quotient is a model of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ and it satisfies AIP. The theory $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR}$ is complete.

Recursion and projection Here we discuss the combination of recursion, projection, and the empty process. The theory $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ has as its signature the signature of $\mathrm{BPA}_{(\delta)}\mathrm{rec} + \mathrm{PR}$ and a constant $\varepsilon \notin A$ (and $\varepsilon \neq \delta$). The axioms of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ are the ones of $\mathrm{BPA}_{(\delta)}\mathrm{rec} + \mathrm{PR}$ and the axioms of table 11. Moreover, we assume for axiom PR1 (table 7) that a may also be ε.

The standard facts (and their proofs) of subsection 2.4.2 are easily translated to the present situation.

The semantics of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ is given by means of a term deduction system $T(\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR})$. Its signature is the signature of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ and its rules are those of $T(\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{PR})$ plus the rules concerning recursion that are presented in table 13. Since bisimulation equivalence is a congruence, we can define the operators of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ on the quotient algebra of closed $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ terms with respect to bisimulation equivalence. This quotient is a model of $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec} + \mathrm{PR}$ and it satisfies RDP, RSP, and AIP^-, but not its unrestricted version AIP. As a consequence, we can prove that $\mathrm{BPA}_{(\delta)\varepsilon}\mathrm{rec}$ satisfies RSP. This is proved in the same way as theorem 2.4.36.

2.6.3 CCS termination

A variant of the empty process is given by the CCS process NIL [Milner, 1980]. We can extend the signature of BPA by the constant NIL, and formulate the operational rules in table 15. These rules are taken from [Aceto and Hennessy, 1992].

The crucial difference between the *necessary termination* predicate $\surd$ and the termination option predicate $\downarrow$ is in the rule for $+$: for $\surd$, *both* components must terminate in order for the sum to terminate. As a result, NIL satisfies the laws for ε but at the same time the law $x + \mathrm{NIL} = x$. A consequence is that the law A4 (distributivity of $\cdot$ over $+$) does not hold for all processes, and so $\mathrm{BPA}_{\mathrm{NIL}}$ cannot be axiomatized using the axioms of BPA. The following complete axiomatization is taken from [Baeten and Vaandrager, 1992]; for more information, we refer to this paper.

Table 15. Derivation rules of $T(\mathrm{BPA_{NIL}})$.

$$a \xrightarrow{a} \mathrm{NIL} \qquad \mathrm{NIL}\surd$$

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

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

As before, we can add δ without any operational rules. Its axioms have to be adapted in the presence of NIL, though. We show this in table 17.

2.7 Renaming in BPA

Sometimes it is useful to have the possibility of *renaming* atomic actions. The material of this subsection is based on [Baeten and Bergstra, 1988a] with improvements by [Vaandrager, 1990a]. Renaming operators occur in most concurrency theories; see, for example, [Milner, 1980; Milner, 1989], [Hennessy, 1988], and [Hoare, 1985].

The signature of the equational specification BPA + RN consists of the signature of BPA plus for each function f from the set of atomic actions to itself a unary operator ρ_f called a *renaming* operator. Such a function f is called a renaming function. The axioms of BPA + RN are the ones for BPA plus the axioms concerning renaming operators displayed in table 18.

Structural induction In BPA + RN we can use structural induction just as in BPA, since closed BPA+RN terms can be rewritten into basic BPA terms. To that end, we will first prove that the term rewriting system associated to BPA + RN is strongly normalizing. We display the rewrite rules concerning the renaming operators in table 19. Note that, in rule RRN1, $f(a)$ stands

Table 16. $\mathrm{BPA_{NIL}}$.

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(ax + by + z)w = axw + (by + z)w$	A4*
$(xy)z = x(yz)$	A5
$x + \mathrm{NIL} = x$	A6*
$x \cdot \mathrm{NIL} = x$	A8*
$\mathrm{NIL} \cdot x = x$	A9*

Table 17. δ in the presence of NIL.

$ax + \delta = ax$	A6**
$\delta \cdot x = \delta$	A7

for the atomic action that a is renamed into.

Theorem 2.7.1. *The term rewriting system associated to* BPA + RN *is strongly normalizing. The rewrite rules are those of table 2 and the rules in table 19.*

Proof. We will apply theorem 2.2.18 to prove that the rewrite rules are terminating. For that, we first give a partial ordering of the signature.

$$\forall f, a \in A : \rho_f > \cdot > +, \rho_f > a.$$

Moreover, we give sequential composition the lexicographical status for the first argument. Now straightforward calculations will show that each left-hand side of a rewrite rule is strictly greater in the $>_{lpo}$ ordering than its right-hand side. ∎

With the aid of the above termination result, we can show the elimination theorem for basic process algebra with renaming operators.

Theorem 2.7.2. *For every closed* BPA + RN *term t there is a basic* BPA *term s such that* BPA + RN $\vdash t = s$.

Proof. Consider the term rewriting system presented in table 19. According to theorem 2.7.1, this term rewriting system is strongly normalizing. Now let t be a closed BPA + RN term and rewrite this into a normal form s with respect to the term rewriting system of table 19. With proposition 2.2.6 it suffices to show that s is a closed BPA term. Suppose that s contains a renaming operator and consider the smallest subterm containing this occurrence. The subterm has the form $\rho_f(u)$ with u a closed BPA term. This contradicts the normality of s, since now we can rewrite the subterm using RRN1, RRN2, or RRN3. So s is a closed BPA term. ∎

Proposition 2.7.3. *Let $f, g : A \longrightarrow A$. Let x be a closed* BPA + RN *term. Then we have the following:*

Table 18. Renaming.

$\rho_f(a) = f(a)$	RN1
$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$	RN2
$\rho_f(xy) = \rho_f(x)\rho_f(y)$	RN3

(i) $\rho_I(x) = x$,

(ii) $\rho_f\big(\rho_g(x)\big) = \rho_{f\circ g}(x)$.

Here, the function $f \circ g : A \longrightarrow A$ *is defined by* $f \circ g(a) = f\big(g(a)\big)$ *and the function* $I : A \longrightarrow A$ *is defined by* $I(a) = a$ *for all* $a \in A$.

Proof. With the aid of theorem 2.7.2 it suffices to prove the theorem for basic BPA terms. For these terms the proof is trivial. ∎

Semantics We give the semantics for BPA + RN by means of a term deduction system $T(\mathrm{BPA}+\mathrm{RN})$, whose signature is the one of BPA + RN and whose rules are the rules of tables 5 and 20. Bisimulation equivalence is a congruence, so the quotient of closed BPA + RN terms modulo bisimulation equivalence is well-defined. This means that the operators of BPA + RN can be defined on this quotient, which is a model of BPA + RN.

Theorem 2.7.4. *The set of closed* BPA + RN *terms modulo bisimulation equivalence is a model of* BPA + RN.

Proof. Axioms A1–A5 are treated as in 2.2.35. For RN1 take the relation that only relates $\rho_f(a)$ and $f(a)$. RN2 goes like A1. RN3 goes like A5. ∎

At this point we have all the ingredients necessary to state and prove that BPA + RN is a conservative extension of BPA.

Theorem 2.7.5. *The equational specification* BPA + RN *is a conservative extension of the equational specification* BPA. *That is, if* t *and* s *are closed* BPA *terms, then we have*

$$\mathrm{BPA} \vdash t = s \iff \mathrm{BPA} + \mathrm{RN} \vdash t = s.$$

Proof. The operational semantics of BPA can be operationally conservatively added to the operational rules concerning the renaming operator. This follows immediately from theorem 2.4.15. The sum of these

Table 19. A term rewriting system for BPA + RN.

$\rho_f(a) \to f(a)$	RRN1
$\rho_f(x+y) \to \rho_f(x) + \rho_f(y)$	RRN2
$\rho_f(xy) \to \rho_f(x)\rho_f(y)$	RRN3

Table 20. Derivation rules concerning renaming operators.

$$\frac{x \xrightarrow{a} \surd}{\rho_f(x) \xrightarrow{f(a)} \surd} \qquad \frac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')}$$

operational rules is precisely the operational semantics of BPA + RN. Now with theorem 2.4.19 we find that modulo strong bisimulation equivalence BPA + RN is an operationally conservative extension of BPA. So with theorem 2.4.24 we find with the soundness of the equational specification BPA + RN and the soundness and completeness of BPA that basic process algebra with renamings is an equationally conservative extension of BPA. ∎

With the aid of the above conservativity result and the elimination theorem for BPA + RN, we find the completeness of BPA + RN.

Theorem 2.7.6. *The axiom system* BPA + RN *is a complete axiomatization of the set of closed* BPA + RN *terms modulo bisimulation equivalence.*

Proof. Apply theorem 2.4.26. ∎

2.7.1 Extensions of BPA + RN

In this subsection we will discuss the extensions of BPA+RN with recursion and/or projections.

Recursion We can add recursion to BPA + RN in the same way as we added recursion to BPA. The equational specification BPArec + RN has as its signature the signature of BPArec plus for all functions f from A to A a renaming operator ρ_f. The equations of BPArec + RN are the axioms of BPArec plus the axioms concerning renaming; see table 18. We can turn the set of closed BPArec + RN terms into a model of BPArec + RN that satisfies RDP and RSP as usual.

Projection Projections can be added in an obvious way to BPA+RN: just add the projection functions and their axioms to the equational specification BPA + RN to obtain BPA + RN + PR. The standard facts that hold for BPA + PR also hold for BPA + RN + PR. As we did for BPA + PR we can infer that BPA + RN + PR is sound and complete, and that AIP is valid.

Recursion and projection The extension with both recursion and projection of BPA + RN, called BPArec + RN + PR, can be obtained just like in the case of BPA.

2.7.2 Renaming in basic process algebra with deadlock

In this subsection we will extend the BPA_δ family with renaming operators. We begin with BPA_δ itself. The equational specification $\mathrm{BPA}_\delta + \mathrm{RN}$ has as its signature the one of BPA_δ and for each function $f : A \cup \{\delta\} \to A \cup \{\delta\}$, with $f(\delta) = \delta$, a unary operator ρ_f called a renaming operator. The axioms of BPA_δ + RN are the axioms of BPA + RN plus the axioms concerning deadlock; see table 10. We assume for axiom RN1 (table 18) that a ranges over $A \cup \{\delta\}$. Note that we have $\rho_f(\delta) = f(\delta) = \delta$, for all renaming operators. This is necessary: it is easy to derive a contradiction if we allow δ to be renamed into an atomic action.

Structural induction We can use structural induction as before, since closed $\mathrm{BPA}_\delta + \mathrm{RN}$ terms can be rewritten into basic BPA_δ terms. This follows from the next elimination theorem.

Theorem 2.7.7. *For every closed* $\mathrm{BPA}_\delta + \mathrm{RN}$ *term* t *there is a basic* BPA_δ *term* s *such that* $\mathrm{BPA}_\delta + \mathrm{RN} \vdash t = s$.

If t *and* s *are closed* BPA_δ *terms, then we have*

$$\mathrm{BPA}_\delta \vdash t = s \iff \mathrm{BPA}_\delta + \mathrm{RN} \vdash t = s.$$

Proof. Add the extra rewrite rule $f(\delta) \to \delta$ to table 19 and reiterate the proof of theorem 2.7.2. ∎

Remark 2.7.8. Note that proposition 2.7.3 also holds for $\mathrm{BPA}_\delta + \mathrm{RN}$.

Semantics The semantics for $\mathrm{BPA}_\delta + \mathrm{RN}$ can be given just like the semantics for $\mathrm{BPA} + \mathrm{RN}$. Let $T(\mathrm{BPA}_\delta + \mathrm{RN})$ be the term deduction system with the signature of $\mathrm{BPA}_\delta + \mathrm{RN}$ and with the rules of tables 5 and 20. In the latter table we further assume that $f(a) \in A$. Since bisimulation equivalence is a congruence, the quotient of the set of closed $\mathrm{BPA}_\delta + \mathrm{RN}$ terms with respect to bisimulation equivalence is well-defined. This quotient is a model of $\mathrm{BPA}_\delta + \mathrm{RN}$; from this, the completeness of BPA_δ, and the elimination result, the completeness of $\mathrm{BPA}_\delta + \mathrm{RN}$ follows.

2.7.3 Extensions of $\mathrm{BPA}_\delta + \mathrm{RN}$ and BPA_δ

In this subsection we discuss the extensions of $\mathrm{BPA}_\delta + \mathrm{RN}$ with recursion and/or projections and we discuss the extension of BPA_δ with a particular renaming operator.

Recursion and/or projection The extensions of $\mathrm{BPA}_\delta + \mathrm{RN}$ with recursion, projection, or a combination of both are obtained in the same way as these extensions without deadlock; see section 2.7.1.

Encapsulation In most concurrency theories in which a form of deadlock is present there usually is the notion of an *encapsulation* or *restriction* operator; this is a renaming operator that renames certain atomic actions into δ. The notion of encapsulation and the notation ∂_H stem from [Bergstra and Klop, 1984b]. The notion named restriction is due to [Milner, 1980].

In this subsection we will add the encapsulation operator to BPA_δ.

The equational specification $\mathrm{BPA}_\delta + \partial_H$ has as its signature the one of BPA_δ plus for each $H \subseteq A$ a unary operator ∂_H called the encapsulation operator. The axioms of $\mathrm{BPA}_\delta + \partial_H$ are the equations of BPA_δ and the equations defining ∂_H in table 21. We assume in this table that a ranges over $A \cup \{\delta\}$, so in particular we find with D1 that $\partial_H(\delta) = \delta$.

Table 21. The encapsulation operator.

$\partial_H(a) = a, \text{ if } a \notin H$	D1
$\partial_H(a) = \delta, \text{ if } a \in H$	D2
$\partial_H(x+y) = \partial_H(x) + \partial_H(y)$	D3
$\partial_H(xy) = \partial_H(x)\partial_H(y)$	D4

The semantics of $\mathrm{BPA}_\delta + \partial_H$ can be derived just like in the case of $\mathrm{BPA}_\delta +$ RN. We can take $\partial_H = \rho_{f_H}$ with

$$f_H(a) = \begin{cases} a & \text{if } a \notin H; \\ \delta & \text{otherwise.} \end{cases}$$

For completeness sake, we give the operational rules for the encapsulation operator in table 22.

It is also straightforward to extend $\mathrm{BPA}_\delta + \partial_H$ with recursion and/or projections.

2.7.4 Renaming in basic process algebra with empty process

In this subsection we will add renaming operators to both BPA_ε and $\mathrm{BPA}_{\delta\varepsilon}$ with extensions. We will simultaneously refer to both of them as before with parentheses: $\mathrm{BPA}_{(\delta)\varepsilon}$. Arbitrary combinations of renaming operators and the empty process introduce a form of abstraction, which is beyond the scope of concrete process algebra. Therefore, we will restrict ourselves to the concrete subcase that prohibits renaming into the empty process.

The signature of the equational specification $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ is the signature of $\mathrm{BPA}_{(\delta)} + \mathrm{RN}$ plus a constant $\varepsilon \notin A$ ($\varepsilon \neq \delta$). We do not allow renaming into ε so for the functions f we assume (moreover) that $f(a) \in A_{(\delta)}$ if $a \in A_{(\delta)}$ and $\big(f(\delta) = \delta,\big)$ $f(\varepsilon) = \varepsilon$. The axioms of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ are the ones of $\mathrm{BPA}_{(\delta)\varepsilon}$ and the equations for renaming; see table 18. Note that $\rho_f(\varepsilon) = \varepsilon$ (and $\rho_f(\delta) = \delta$).

Abstraction We have an abstraction mechanism if we allow renaming into the empty process. For instance, suppose that we have two atomic actions, say a and b. Let $f(a) = a$ and $f(b) = \varepsilon$. Then $\rho_f(ab) = a$ and we have abstracted from b.

Table 22. Derivation rules for the encapsulation operator.

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

Structural induction We can use structural induction as before, since closed $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ terms can be rewritten into basic $\mathrm{BPA}_{(\delta)\varepsilon}$ terms. This can be shown along the same lines as the elimination theorem for the theory without the empty process; see, for instance, theorem 2.7.2.

Note that proposition 2.7.3 also holds for $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$.

Semantics The semantics of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ will be given by means of a term deduction system. Let $T(\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN})$ be the term deduction system with $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ as its signature and with rules displayed in tables 12 and 23. Bisimulation equivalence is a congruence, so we can define the operators of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ on the quotient of closed $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ terms modulo bisimulation equivalence. It is straightforward to prove that this is a model of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$. The completeness of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ is also standard to prove.

Look-ahead If we allow renaming into the empty process, we need two more derivation rules that concern renaming; they introduce a look-ahead as can be seen in table 24. The operational rules that we list in this table are due to [Baeten and Glabbeek, 1987].

We will give an example. Suppose that $f(a) = \varepsilon$ and $f(b) = b$. Then we can derive $\rho_f(a^n b) \xrightarrow{b} \varepsilon$, for each $n \geq 1$.

2.7.5 Extensions of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ and $\mathrm{BPA}_{(\delta)\varepsilon}$

In this subsection we discuss the extensions of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ with recursion and/or projections and we discuss the extension of $\mathrm{BPA}_{(\delta)\varepsilon}$ with a particular renaming operator.

Recursion and/or projection The extensions of $\mathrm{BPA}_{(\delta)\varepsilon} + \mathrm{RN}$ with recursion and/or projection can be obtained just like before. However, if we allow renaming into the empty process the definition of a guarded recursive specification has to be adapted. We will show in an example that RSP no longer holds with the present definition. This example is taken from [Baeten *et al.*, 1987]. Suppose that we have at least three elements in

Table 23. Derivation rules for renaming operators and empty process.

$$\frac{x\downarrow}{\rho_f(x)\downarrow} \qquad \frac{x \xrightarrow{a} x'}{\rho_f(x) \xrightarrow{f(a)} \rho_f(x')},\ f(a) \in A$$

Table 24. Extra rules when we allow renaming into ε.

$$\frac{x \xrightarrow{a} y,\ \rho_f(y)\downarrow}{\rho_f(x)\downarrow},\ f(a) = \varepsilon \qquad \frac{x \xrightarrow{a} y,\ \rho_f(y) \xrightarrow{b} x'}{\rho_f(x) \xrightarrow{b} x'},\ f(a) = \varepsilon$$

the set of atomic actions, say a, i, and j. Let ε_i resp. ε_j be the renaming operators that rename i resp. j into ε and further do nothing. Then the guarded recursive specification

$$\{X = i \cdot \varepsilon_j(Y), Y = j \cdot \varepsilon_i(X)\}$$

has the solution $\{ia^n, ja^n\}$ for all $n \geq 1$. So RSP cannot hold.

A possible solution can be to prohibit the occurrences of renaming operators in the body of guarded recursive specifications. Also more sophisticated solutions can be obtained in terms of restrictions on the renaming operators that do occur in the body of a guarded recursive specification.

Encapsulation The extension of $\text{BPA}_{\delta\varepsilon}$ with the encapsulation operator can be obtained in the same way as in the case without the empty process; see subsection 2.7.3.

2.8 The state operator

In this subsection we extend BPA with the (simple) state operator, which is a generalization of a renaming operator. It is a renaming operator with a memory to describe processes with an independent global state. We denote the state operator by λ_s; the subscript is the memory cell containing the current state s. This subsection is based on [Baeten and Bergstra, 1988a]. Another treatment of state operators can be found in [Verhoef, 1992].

Next, we will discuss the signature of BPA_λ. It consists of the usual signature of BPA extended with for each $m \in M$ and $s \in S$ a unary operator λ_s^m called the (simple) state operator. M, S, and A are mutually disjoint. The set S is the state space and M is the set of object names; the M stands for machine.

We describe the state operator by means of two total functions *action* and *effect*. The function *action* describes the renaming of the atomic actions and the function *effect* describes the contents of the memory. We have

$$\textit{action} : A \times M \times S \longrightarrow A, \qquad \textit{effect} : A \times M \times S \longrightarrow S.$$

Mostly, we write $a(m, s)$ for $\textit{action}(a, m, s)$ and $s(m, a)$ for $\textit{effect}(a, m, s)$.

Intuitively, we think of the process $\lambda_s^m(x)$ as follows: m represents a machine (say a computer), s describes its state (say the contents of its memory), x is its input (say a program). Now $\lambda_s^m(x)$ describes what happens when x is presented to machine m in state s.

Now we discuss the equations of BPA_λ. They are the axioms of BPA (see table 1) and the axioms of table 25. The first axiom SO1 gives the renaming part of the state operator. The second axiom SO2 shows the effect of renaming an atomic action on the current state. Note that if a renaming has no effect on states we obtain an ordinary renaming oper-

Table 25. The axioms defining the state operator.

$\lambda_s^m(a) = a(m,s)$	SO1
$\lambda_s^m(ax) = a(m,s)\lambda_{s(m,a)}^m(x)$	SO2
$\lambda_s^m(x+y) = \lambda_s^m(x) + \lambda_s^m(y)$	SO3

ator. Axiom SO3 expresses that the state operator distributes over the alternative composition.

2.8.1 Termination and elimination

Next, it is our aim to show that the state operator can be eliminated. Therefore, we will use that the term rewriting system associated to BPA_λ is strongly normalizing. We will prove the latter fact with the aid of the method of the recursive path ordering. However, we cannot apply this method immediately. This is due to the fact that we cannot hope to find a strict partial ordering on the signature of BPA_λ that does the job. The problematical rule is the rewrite rule RSO2 (see table 26). Suppose that we have one atomic action a. Let us have two different states, which shall remain nameless. Take an inert action function, that is it does nothing, and let the effect function act as a switch. This yields the following instantiation for the rewrite rule RSO2:

$$\begin{aligned} \lambda(ax) &\rightarrow a\lambda'(x), \\ \lambda'(ax) &\rightarrow a\lambda(x). \end{aligned}$$

For the first rewrite rule the ordering that works is $\lambda > \lambda'$. But for the second rule, the ordering should be the opposite, thus yielding an inconsistency. We solve this by giving the state operator a rank; the rank of a state operator depends on the weight of its operand. This idea is taken from [Verhoef, 1992]; he based this idea on a method that [Bergstra and Klop, 1985] give for the termination of a concurrent system (see theorem 3.2.3 where we treat their method).

Definition 2.8.1. Let x and y be terms and let a be an atomic action.

Table 26. The rewrite rules for the simple state operator.

$\lambda_s^m(a) \rightarrow a(m,s)$	RSO1
$\lambda_s^m(ax) \rightarrow a(m,s)\lambda_{s(m,a)}^m(x)$	RSO2
$\lambda_s^m(x+y) \rightarrow \lambda_s^m(x) + \lambda_s^m(y)$	RSO3

The weight of a term x, notation $|x|$, is defined inductively as follows.

- $|a| = 1$,
- $|x + y| = \max\{|x|, |y|\}$,
- $|x \cdot y| = |x| + |y|$,
- $|\lambda_s^m(x)| = |x|$.

Definition 2.8.2. The rank of a state operator is the weight of the subterm of which it is the leading operator. So, if $|x| = n$, we write $\lambda_{n,s}^m(x)$.

Theorem 2.8.3. *The term rewriting system associated to the equational specification of* BPA_λ *is strongly normalizing. The rewrite rules are the ones listed in tables 2 and 26.*

Proof. Take the following precedence for the elements of the signature of BPA_λ:

$$\forall n \geq 1, m \in M, s, s' \in S, a \in A: \; \lambda_{n+1,s}^m > \lambda_{n,s'}^m > \cdot > +, \lambda_{n,s}^m > a.$$

Moreover, give the sequential composition the lexicographical status for the first argument. Now it is not hard to see that each left-hand side of the rewrite rules is strictly greater than its right-hand side in the $>_{lpo}$ ordering. We will treat an example. Let $a' = a(m,s)$, $\lambda_n = \lambda_{n,s}^m$, and $\lambda_n' = \lambda_{n,s(m,a)}^m$. Suppose that $|x| = n$.

$$\begin{array}{rll}
\lambda_{n+1}(ax) & >_{lpo} & \lambda_{n+1}^*(ax) \\
& >_{lpo} & \lambda_{n+1}^*(ax) \cdot \lambda_{n+1}^*(ax) \\
& >_{lpo} & a' \cdot \lambda_n'\left(\lambda_{n+1}^*(ax)\right) \\
& >_{lpo} & a' \cdot \lambda_n'(ax) \\
& >_{lpo} & a' \cdot \lambda_n'^*(ax) \\
& >_{lpo} & a' \cdot \lambda_n'(a \cdot^* x) \\
& >_{lpo} & a' \cdot \lambda_n'(x).
\end{array}$$

The other inequalities are checked analogously. With theorem 2.2.18 it follows that the system is terminating. ∎

Now, we can state the elimination theorem for basic process algebra with the state operator.

Theorem 2.8.4. *For every closed* BPA_λ *term* t *there is a basic* BPA *term* s *such that* $\mathrm{BPA}_\lambda \vdash t = s$.

Proof. Straightforward. ∎

Semantics We give the semantics for BPA_λ by means of a term deduction system $T(\mathrm{BPA}_\lambda)$. Its signature is that of BPA_λ and its rules are the rules of tables 5 and 27. According to theorem 2.2.32 bisimulation equivalence

Table 27. Derivation rules of $T(\mathrm{BPA}_\lambda)$.

$\dfrac{x \xrightarrow{a} \surd}{\lambda_s^m(x) \xrightarrow{a(m,s)} \surd}$	$\dfrac{x \xrightarrow{a} x'}{\lambda_s^m(x) \xrightarrow{a(m,s)} \lambda_{s(m,a)}^m(x')}$

is a congruence so the operators of BPA_λ can be defined on the quotient of the closed BPA_λ terms modulo bisimulation equivalence. Moreover, it is a model of BPA_λ. With the aid of the method explained in subsection 2.4.1, it is not hard to see that BPA_λ is a conservative extension of BPA. Then it easily follows with theorem 2.4.26 that the axioms in tables 1 and 25 constitute a complete axiomatization of BPA_λ.

Extensions The extensions of BPA_λ with recursion and/or projection are obtained in the same way as those of BPA.

The extension of BPA_δ with the state operator is obtained in the same way as the extension of BPA with it. We also allow $a(m,s) = \delta$ so the action function can rename into δ. There is only one extra axiom: we need to know what the state operator should do with the extra constant δ. Therefore, we need to know how the functions *action* and *effect* are extended to $A \cup \{\delta\}$. We define $\delta(m,s) = \delta$ and $s(m,\delta) = s$. The extra axiom is

$$\lambda_s^m(\delta) = \delta.$$

The extensions of $\mathrm{BPA}_{\delta\lambda}$ with recursion and/or projection are obtained in the same way as those of BPA_δ.

The following example is due to Alban Ponse [Ponse, 1993].

Example 2.8.5. We describe an edit session with the aid of the state operator. We will use the theory BPA_λ with recursion.

The characters that can be typed are the lower case characters $a, b, \ldots, z$ with the usual meaning, and two special characters D and P. We call the set of characters that can be typed C. The character D stands for the deletion of the last character from the memory; if the memory is empty pressing the D will cause a *beep*. The P sends the contents of the memory to a printer device. We have a user U that wants to type characters from C. The user is specified as follows:

$$U = \sum_{c \in C} \mathit{type}(c) \cdot U + \sum_{c \in C} \mathit{type}(c).$$

The state space is $S = \{a, b, \ldots, z\}^*$; we denote the empty word by ε. The set A of atomic actions is

$$\{\mathit{type}(c), \mathit{typed}(c), \mathit{deleted}(c) : c \in C\} \cup \{\mathit{printed}(\sigma) : \sigma \in S\} \cup \{\mathit{beep}\}.$$

We give the *action* and *effect* functions implicitly, by giving the relevant axioms for our specific state operator. We assume that $c \in \{a, b, \ldots, z\}$ and $\sigma \in S$.

$$\begin{aligned}
\lambda_\varepsilon(\mathit{type}(D) \cdot x) &= \mathit{beep} \cdot \lambda_\varepsilon(x) \\
\lambda_{\sigma c}(\mathit{type}(D) \cdot x) &= \mathit{deleted}(c) \cdot \lambda_\sigma(x) \\
\lambda_\sigma(\mathit{type}(c) \cdot x) &= \mathit{typed}(c) \cdot \lambda_{\sigma c}(x) \\
\lambda_\sigma(\mathit{type}(P) \cdot x) &= \mathit{printed}(\sigma) \cdot \lambda_\sigma(x).
\end{aligned}$$

For the other atomic actions in A we define the *action* and *effect* functions to be inert. Now the process $\lambda_\varepsilon(U)$ describes an edit session. Since we have only one object name, we left out the superscripts.

Note that the following choice for the last equation of the above display also works:

$$\lambda_\sigma(\mathit{type}(P) \cdot x) = \mathit{printed}(\sigma) \cdot \lambda_\varepsilon(x).$$

However, we did not choose this option to separate different concerns: if we want to empty the memory, it may be more appropriate to define an atomic action that empties the memory.

2.9 The extended state operator

In the following, we will discuss BPA with the extended state operator, which is a generalization of the simple state operator. We denote the extended state operator by Λ_s^m. The difference with the (simple) state operator is that we can rename an atomic action into a closed term of a particular form, namely a finite sum of atomic actions. With this extra feature it is possible to translate an instruction like `read(x)` into process algebra.

This subsection is based on [Baeten and Bergstra, 1988a].

We discuss the signature of BPA_Λ. It consists of the usual signature of BPA extended for each $m \in M$ and $s \in S$ with a unary operator Λ_s^m called the extended state operator. M, S, and A are mutually disjoint. The set S is the state space and M is the set of object names; the M stands for machine.

We describe the extended state operator by means of two functions *action* and *effect*. The function *action* describes the renaming of the atomic actions and the function *effect* describes the contents of the memory. We have

$$\mathit{action} : A \times M \times S \longrightarrow 2^A \setminus \{\emptyset\}, \qquad \mathit{effect} : A \times M \times S \times A \longrightarrow S.$$

We write $a(m, s)$ for $\mathit{action}(a, m, s)$ and $s(m, a, b)$ for $\mathit{effect}(a, m, s, b)$.

The axioms of BPA_Λ are those of BPA and the axioms of table 28. Next, we discuss them. The first axiom GS1 states that an atomic action

Table 28. The axioms defining the generalized state operator.

$\Lambda_s^m(a) = \sum_{b \in a(m,s)} b$	GS1
$\Lambda_s^m(ax) = \sum_{b \in a(m,s)} b \cdot \Lambda_{s(m,a,b)}^m(x)$	GS2
$\Lambda_s^m(x+y) = \Lambda_s^m(x) + \Lambda_s^m(y)$	GS3

Table 29. Derivation rules of $T(\mathrm{BPA}_\Lambda)$.

$$\frac{x \xrightarrow{a} \surd}{\Lambda_s^m(x) \xrightarrow{b} \surd}, b \in a(m,s) \qquad \frac{x \xrightarrow{a} x'}{\Lambda_s^m(x) \xrightarrow{b} \Lambda_{s(m,a,b)}^m(x')}, b \in a(m,s)$$

is renamed into a sum of atomic actions. Axiom GS2 shows the side effects of the renaming on the state space. Axiom GS3 expresses that the extended state operator distributes over the alternative composition.

Termination In the previous subsection (2.8) we mentioned that we cannot use the method of the recursive path ordering immediately. The same phenomenon occurs with the extended state operator. Fortunately, the solution of the problems is the same as for the simple state operator. We have to define ranked extended state operators and prove the termination of this system. We omit the details and refer to subsection 2.8 for more information. We only mention the main result.

Theorem 2.9.1. *The term rewriting system that is associated to* BPA_Λ *is strongly normalizing. The rewrite rules are those of tables 2 and 30.*

Semantics We give the semantics for BPA_Λ by means of a term deduction system $T(\mathrm{BPA}_\Lambda)$. Its signature is that of BPA_Λ and its rules are the rules of tables 5 and 29. According to 2.2.32 bisimulation equivalence is a congruence so the quotient of the closed BPA_Λ terms modulo bisimulation equivalence is well-defined. Moreover, it is easily seen that the quotient is a model of BPA_Λ. With the theory of section 2.4.1, we find that BPA_Λ is a conservative extension of BPA. With the termination theorem 2.9.1 and an elimination result, similar to 2.8.4, we find using theorem 2.4.26 that the axioms of tables 1 and 28 constitute a complete axiomatization of BPA_Λ.

Extensions The extensions of BPA_Λ and $\mathrm{BPA}_{\delta\Lambda}$ with recursion and/or projection are obtained in the same way as those with BPA_λ and $\mathrm{BPA}_{\delta\lambda}$. The only difference is that we can now allow $a(m,s) = \emptyset$ if in addition we define

Table 30. The rewrite rules for the extended state operator.

$\Lambda_s^m(a) \to \sum_{b \in a(m,s)} b$	RGS1
$\Lambda_s^m(ax) \to \sum_{b \in a(m,s)} b \cdot \Lambda^m_{s(m,a,b)}(x)$	RGS2
$\Lambda_s^m(x+y) \to \Lambda_s^m(x) + \Lambda_s^m(y)$	RGS3

$$\sum_{b \in \emptyset} b = \delta.$$

As before, we have $\Lambda_s^m(\delta) = \delta$.

Example 2.9.2. In this example we describe a gambling session of a fruit machine player with the aid of the extended state operator. We use the theory BPA_Λ with recursion, again leaving out superscripts.

The player P is specified as follows:

$$P = \mathit{pull} \cdot \mathit{win} \cdot P.$$

Note that P has a serious gambling problem. With the extended state operator we specify what actually will happen during the gambling session. First we give the state space $S = F \times F \times F$ where

$$F = \{\mathit{bar}, \mathit{bell}, \mathit{grape}, \mathit{melon}, \mathit{orange}, \mathit{cherry}\}.$$

The set of atomic actions A is

$$\{\mathit{pull}, \mathit{win}, \mathit{lost}\} \cup \{\mathit{won}(f) : f \in F\} \cup \{\mathit{pulled}(f,g,h) : f,g,h \in F\}.$$

We define the *action* and *effect* functions implicitly by giving the relevant instances of axiom GS2. The first equation expresses that if P pulls the fruit machine it will give one of the possible triples. The second equation describes that *win* is renamed into *lost* if the obtained triple contains different "fruits". If the triple contains only one symbol, say *melon*, we have that *win* is renamed into *won*(*melon*).

$$\begin{aligned}
\Lambda_{(u,v,w)}(\mathit{pull} \cdot x) &= \sum_{(f,g,h) \in S} \mathit{pulled}(f,g,h) \cdot \Lambda_{(f,g,h)}(x) \\
\Lambda_{(f,g,h)}(\mathit{win} \cdot x) &= \sum_{f=g=h} \mathit{won}(f) \cdot \Lambda_{(f,g,h)}(x) + \sum_{f \neq g} \mathit{lost} \cdot \Lambda_{(f,g,h)}(x) \\
&\quad + \sum_{g \neq h} \mathit{lost} \cdot \Lambda_{(f,g,h)}(x) + \sum_{f \neq h} \mathit{lost} \cdot \Lambda_{(f,g,h)}(x).
\end{aligned}$$

For the other actions in A we define both functions *action* and *effect* to be inert. The process $\Lambda_{(f,g,h)}(P)$ with $f, g, h \in F$ describes a gambling session.

2.10 The priority operator

In this subsection we introduce BPA_δ with the priority operator that originates from [Baeten *et al.*, 1986].

The signature of the equational specification BPA_δ with the priority operator, $\mathrm{BPA}_{\delta\theta}$, consists of the signature of BPA_δ plus a unary operator θ and an auxiliary binary operator $\triangleleft$ pronounced "unless". Furthermore, a partial ordering $<$, called the *priority ordering* on the set of atomic actions A is presumed. The axioms of the equational specification $\mathrm{BPA}_{\delta\theta}$ are the usual axioms of BPA_δ (see tables 1 and 10) and the axioms of tables 31 and 32. The axioms that we present in these tables make use of δ. We can imagine a system without δ (BPA_θ) but such a system has a laborious axiomatization; see [Bergstra, 1985] for such an axiomatization.

Next, we will discuss the axioms concerning priority.

The axioms of table 31 define the auxiliary unless operator. It is used to axiomatize the priority operator. The intended behaviour of the unless operator is that the process $x \triangleleft y$ filters out all summands of x with an initial action smaller than some initial action of y. So, one could say that the second argument y is the filter. If, for instance, $a > b > c$ then we want that

$$(ax + by + cz) \triangleleft (bp + cq) = ax + by.$$

To model the filter behaviour we use the constant process δ to rename the unwanted initial actions of x into δ. The axioms U1 and U2 essentially define the mesh of the filter: they say which actions can pass the filter and which cannot. Axiom U3 expresses the fact that the initial actions of y are the same as the initial actions of yz. Axiom U4 says that it is the same to filter the initial actions of x with filter $y + z$ as to filter first the

Table 31. The axioms defining the unless operator.

$a \triangleleft b = a$	if $\neg(a < b)$	U1
$a \triangleleft b = \delta$	if $a < b$	U2
$x \triangleleft yz = x \triangleleft y$		U3
$x \triangleleft (y + z) = (x \triangleleft y) \triangleleft z$		U4
$xy \triangleleft z = (x \triangleleft z)y$		U5
$(x + y) \triangleleft z = x \triangleleft z + y \triangleleft z$		U6

Table 32. The axioms defining the priority operator.

$\theta(a) = a$	TH1
$\theta(xy) = \theta(x) \cdot \theta(y)$	TH2
$\theta(x + y) = \theta(x) \triangleleft y + \theta(y) \triangleleft x$	TH3

initial actions of x with filter y and filter the result with filter z. Axiom U5 expresses that z is a disposable filter: once in xy the process x is filtered through z, the process y can freely pass. Axiom U6 expresses that filtering a sum is the same as adding the filtered summands.

The priority operator uses the unless operator to filter out the summands with low priority. Thus, the priority operator is invariant under atomic actions and sequential composition. This is expressed in the axioms TH1 and TH2. The priority operator does *not* distribute over the alternative composition, since in a prioritized sum $\theta(x + y)$ there is an interaction between the restrictions concerning the priorities imposed on each other by x and y, whereas in $\theta(x) + \theta(y)$ we do not have such an interaction. Axiom TH3 states that the prioritized sum equals the sum of the prioritized summands with the remaining alternatives as filters. So, for instance, we have

$$\theta(a + b + c) = \theta(a) \triangleleft (b + c) + \theta(b) \triangleleft (a + c) + \theta(c) \triangleleft (a + b).$$

Intuition The partial order $<$ is used in order to describe which actions have priority over other actions. If for instance $a < b$ and b and c are not related we want to have that $\theta(a + b) = b$ and $\theta(b + c) = b + c$. The priority operator thus respects the alternative composition for actions without priority but gives the alternative with the highest priority, in the $<$ hierarchy, if the sum contains prioritized actions. A typical example of a low priority action is an atomic action expressing time-out behaviour: as long as there are alternatives with a higher priority no time-out will be performed within the scope of the priority operator. The priority operator has been used to specify and verify time critical protocols in an untimed setting; see, for instance, [Vaandrager, 1990b].

Next, we list some properties of the unless operator and the priority operator that can be derived from $\mathrm{BPA}_{\delta\theta}$. The first identity expresses that the ordering of filtering does not matter. The second equation expresses that when a process is filtered once, a second application of the same filter has no effect. The third one expresses that a prioritized process $\theta(x)$ is automatically filtered with its subprocess x without priority.

Table 33. Rewrite rules for the unless operator.

$\neg(a < b) \Longrightarrow a \lhd b \rightarrow a$	RU1
$a < b \Longrightarrow a \lhd b \rightarrow \delta$	RU2
$x \lhd yz \rightarrow x \lhd y$	RU3
$x \lhd (y + z) \rightarrow (x \lhd y) \lhd z$	RU4
$xy \lhd z \rightarrow (x \lhd z)y$	RU5
$(x + y) \lhd z \rightarrow x \lhd z + y \lhd z$	RU6
$(x \lhd y) \lhd y \rightarrow x \lhd y$	RU7

Lemma 2.10.1. *The following identities are derivable from* $\mathrm{BPA}_{\delta\theta}$*:*

- $(x \lhd y) \lhd z = (x \lhd z) \lhd y$,
- $(x \lhd y) \lhd y = x \lhd y$,
- $\theta(x) \lhd x = \theta(x)$.

Proof. The proofs of these identities are easy. To illustrate the usage of the axioms we provide full proofs. Here is the first one:

$$(x \lhd y) \lhd z = x \lhd (y + z) = x \lhd (z + y) = (x \lhd z) \lhd y.$$

For the second one, take $z = y$ in the above deduction and use the fact that $y + y = y$. The third identity is derived as follows:

$$\theta(x) \lhd x = \theta(x) \lhd x + \theta(x) \lhd x = \theta(x + x) = \theta(x).$$

Note the double use of the idempotency of the alternative composition in this inference. ∎

Next, we formulate a term rewriting result for basic process algebra with priorities. It states that the term rewriting system associated to $\mathrm{BPA}_{\delta\theta}$ is strongly normalizing. To prove this we use the method of the recursive path ordering that we introduced in subsection 2.2.2. We need the lexicographical variant of this method. Note that the rewrite rules concerning the unless operator (table 33) form a *conditional* term rewriting system. We can, however, see the rewrite rules RU1 and RU2 as a scheme of rules; for all a and b there is a rule. So, in fact, this term rewriting system is unconditional. Thus, we may use the method of the recursive path ordering.

Theorem 2.10.2. *The term rewriting system that is associated to* $\mathrm{BPA}_{\delta\theta}$ *is strongly normalizing. This term rewriting system consists of the rewrite rules listed in table 2, table 33, and table 34.*

Table 34. The rewrite rules for the priority operator.

$\theta(a) \to a$	RTH1
$\theta(xy) \to \theta(x) \cdot \theta(y)$	RTH2
$\theta(x+y) \to \theta(x) \triangleleft y + \theta(y) \triangleleft x$	RTH3
$\theta(x) \triangleleft x \to \theta(x)$	RTH4

Proof. We use the lexicographical variant of the recursive path ordering that we treated in subsection 2.2.2. Take as precedence for the elements of the signature of $\mathrm{BPA}_{\delta\theta}$ the following partial ordering:

$$\theta > \triangleleft > \cdot > +, \quad \forall a \in A : a > \delta.$$

Furthermore, we give the sequential composition the lexicographical status for the first argument and we give the unless operator the lexicographical status for the second argument. We will treat a typical case: we treat the case RU4 where we use the lexicographical status of the unless operator.

$$\begin{aligned} x \triangleleft (y+z) &>_{lpo} x \triangleleft^* (y+z) \\ &>_{lpo} \big(x \triangleleft^* (y+z)\big) \triangleleft (y +^* z) \\ &>_{lpo} \big(x \triangleleft (y +^* z)\big) \triangleleft z \\ &>_{lpo} (x \triangleleft y) \triangleleft z. \end{aligned}$$

The other cases are dealt with in a similar way. This means that we find with theorem 2.2.18 that the term rewriting system is strongly normalizing, which ends the proof of the theorem. ∎

Next, we formulate the elimination theorem for basic process algebra with priorities.

Theorem 2.10.3. *The equational specification* $\mathrm{BPA}_{\delta\theta}$ *has the elimination property for* BPA_{δ}. *That is, for every closed* $\mathrm{BPA}_{\delta\theta}$ *term* t *there is a basic* BPA_{δ} *term* s *such that* $\mathrm{BPA}_{\delta\theta} \vdash t = s$.

Proof. Easy. ∎

2.10.1 Semantics of basic process algebra with priorities

In this subsection we discuss the operational semantics of $\mathrm{BPA}_{\delta\theta}$.

The operational semantics of the priority operator can be found in [Baeten and Bergstra, 1988b]. A more accessible reference is, for instance, [Groote, 1990b] or [Baeten and Weijland, 1990]. In table 35 we give the characterization presented in [Baeten and Weijland, 1990]. In [Bol and Groote, 1991] we find rules that operationally define the unless operator,

Table 35. Derivation rules for the priority operator.

$$\frac{x \xrightarrow{a} x', \ \{x \not\xrightarrow{b}, x \xrightarrow{b}\!\!\!\!\!/\ \surd \mid b > a\}}{\theta(x) \xrightarrow{a} \theta(x')} \qquad \frac{x \xrightarrow{a} \surd, \ \{x \not\xrightarrow{b}, x \xrightarrow{b}\!\!\!\!\!/\ \surd \mid b > a\}}{\theta(x) \xrightarrow{a} \surd}$$

Table 36. Derivation rules for the unless operator.

$$\frac{x \xrightarrow{a} x', \ \{y \not\xrightarrow{b}, y \xrightarrow{b}\!\!\!\!\!/\ \surd \mid b > a\}}{x \triangleleft y \xrightarrow{a} x'} \qquad \frac{x \xrightarrow{a} \surd, \ \{y \not\xrightarrow{b}, y \xrightarrow{b}\!\!\!\!\!/\ \surd \mid b > a\}}{x \triangleleft y \xrightarrow{a} \surd}$$

essentially as in table 36 (but we follow the approach of [Baeten and Weijland, 1990]).

We note that it is possible to operationally characterize the priority operator without the use of the unless operator. The latter one is used for the axiomatization of the priority operator. However, [Bergstra, 1985] gives a not so well-known finite axiomatization of the priority operator without the unless operator. Moreover, in this approach the special constant δ is not necessary. For more information on this axiomatization we refer to [Bergstra, 1985].

We also note that on the basis of an operational semantics for the priority operator it is possible to find the unless operator in a systematical way. This can be done with the paper [Aceto *et al.*, 1994] where an algorithm is given to generate a sound and complete axiomatization from a set of operational rules that satisfy a certain SOS format (the so-called GSOS format, see further on).

An interesting point concerning the operational rules of the priority operator and the unless operator is the appearance of negative premises in them. Clearly, such rules do not satisfy the *path* format. Therefore, in this subsection we will make a third journey to the area of general theory on operational semantics. Next, we will generalize the theory that we already treated in subsection 2.2.3. As a running example we take the operational semantics of basic process algebra with priorities. This subsection is based on [Verhoef, 1994a].

In the following definition we generalize the notion of a term deduction system (cf. definition 2.2.19) in the sense that deduction rules may also contain negative premises. [Bloom *et al.*, 1988] formulated the first format with negative premises; it is called the GSOS format. [Groote, 1990b] generalized this substantially and he proposed the so-called *ntyft/ntyxt* format.

Definition 2.10.4. A term deduction system is a structure (Σ, D) with Σ a signature and D a set of deduction rules. The set $D = D(T_p, T_r)$ is parameterized with two sets, which are called respectively the set of predicate symbols and the set of relation symbols. Let s, t, and $u \in O(\Sigma)$, $P \in T_p$, and $R \in T_r$. We call expressions $Ps, \neg Ps$, tRu, and $t\neg R$ formulas. We call the formulas Ps and tRu positive and $\neg Ps$ and $t\neg R$ negative. If S is a set of formulas we write $PF(S)$ for the subset of positive formulas of S and $NF(S)$ for the subset of negative formulas of S.

A deduction rule $d \in D$ has the form

$$\frac{H}{C}$$

with H a set of formulas and C a positive formula; to save space we will also use the notation H/C. We call the elements of H the hypotheses of d and we call the formula C the conclusion of d. If the set of hypotheses of a deduction rule is empty we call such a rule an axiom. We denote an axiom simply by its conclusion provided that no confusion can arise. The notions "substitution", "*var*", and "closed" extend to formulas and deduction rules as expected.

Example 2.10.5. A typical example of a term deduction system with negative premises is the operational semantics of $\mathrm{BPA}_{\delta\theta}$. The term deduction system $T(\mathrm{BPA}_{\delta\theta})$ has as signature that of the equational specification $\mathrm{BPA}_{\delta\theta}$ and its rules are the rules of tables 5, 35, and 36.

Next, we formalize the notion when a formula holds in a term deduction system with negative premises.

Definition 2.10.6. Let T be a term deduction system. Let $F(T)$ be the set of all closed formulas over T. We denote the set of all positive formulas over T by $PF(T)$ and the negative formulas by $NF(T)$. Let $X \subseteq PF(T)$. We define when a formula $\varphi \in F(T)$ holds in X; notation $X \vdash \varphi$.

$$\begin{array}{lll} X \vdash sRt & \text{if} & sRt \in X, \\ X \vdash Ps & \text{if} & Ps \in X, \\ X \vdash s\neg R & \text{if} & \forall t \in C(\Sigma) : sRt \notin X, \\ X \vdash \neg Ps & \text{if} & Ps \notin X. \end{array}$$

The purpose of a term deduction system is to define a set of positive formulas that can be deduced using the deduction rules. For instance, if the term deduction system contains only positive formulas then the set of deducible formulas comprises all the formulas that can be proved by a well-founded proof tree. If we allow negative formulas in the premises of a deduction rule it is no longer obvious which set of positive formulas can be deduced using the deduction rules. [Bloom *et al.*, 1988] formulate that a transition relation must agree with a transition system specification. We will use their notion; it is only adapted in order to incorporate predicates.

Definition 2.10.7. Let $T = (\Sigma, D)$ be a term deduction system and let $X \subseteq PF(T)$ be a set of positive closed formulas. We say that X agrees with T if a formula φ is in X if and only if there is a deduction rule instantiated with a closed substitution such that the instantiated conclusion equals φ and all the instantiated hypotheses hold in X. More formally: X agrees with T if

$$\varphi \in X \iff \exists\, H/C \in D, \sigma : V \longrightarrow C(\Sigma) : \sigma(C) = \varphi, \forall h \in H : X \vdash \sigma(h).$$

[Groote, 1990b] showed that if for each rule the conclusions are in some sense more difficult than the premises, there is always a set of formulas that agrees with the given rules. [Verhoef, 1994a] generalized this to the case where predicates come into play. Next, we will formalize this notion that is termed a stratification.

Definition 2.10.8. Let $T = (\Sigma, D)$ be a term deduction system. A mapping $S : PF(T) \longrightarrow \alpha$ for an ordinal α is called a stratification for T if for all deduction rules $H/C \in D$ and closed substitutions σ the following conditions hold. For all $h \in PF(H)$ we have $S\big(\sigma(h)\big) \leq S\big(\sigma(C)\big)$; for all $s\neg R \in NF(H)$ we have for all $t \in C(\Sigma) : S\big(\sigma(sRt)\big) < S\big(\sigma(C)\big)$; for all $\neg Ps \in NF(H)$ we have $S\big(\sigma(Ps)\big) < S\big(\sigma(C)\big)$. We call a term deduction system stratifiable if there exists a stratification for it.

Remark 2.10.9. Next, we will give a recipe for finding a stratification. In most cases we can find a stratification (for which the two conditions hold) by measuring the complexity of a positive formula in terms of counting a particular symbol occurring in the conclusion of a rule with negative premises.

Example 2.10.10. As an example of the use of the above rule of thumb, we give a stratification for the term deduction system $T(\mathrm{BPA}_{\delta\theta})$. The rules containing negative premises have in their conclusion a θ or an $\triangleleft$. We define a map that counts the number of θ's and the number of $\triangleleft$'s as follows: let t be a closed term with n_0 occurrences of θ's; and n_1 occurrences of $\triangleleft$'s then $S(t \xrightarrow{a} s) = S(t \xrightarrow{a} \surd) = n_0 + n_1$. Now we check the two conditions for the first rule of table 35. Replace each x and x' by closed terms t and t'. Since the number of θ's plus the number of $\triangleleft$'s occurring in $\theta(t)$ is one greater than the number of θ's plus the number of $\triangleleft$'s occurring in t we are done. The other rules are equally simple.

Next, it is our aim to define a set of positive formulas that agrees with a given term deduction system. Therefore, we will use the following notion. Just think of it as a uniform upper bound to the number of positive premises in a given term deduction system. In general, it is not the least upper bound.

Definition 2.10.11. Let V be a set. If $0 \leq |V| < \aleph_0$ we define the degree of V, denoted by $d(V)$ to equal ω_0. If $|V| = \aleph_\alpha$ for an ordinal $\alpha \geq 0$ we define $d(V) = \omega_{\alpha+1}$.

Let $T = (\Sigma, D)$ be a term deduction system. The degree $d(H/C)$ of a deduction rule $H/C \in D$ is the degree of its set of positive premises; in a formula: $d(H/C) = d\big(PF(H)\big)$. Let $\omega_\alpha = \sup\{d(H/C) : H/C \in D\}$. The degree $d(T)$ of a term deduction system T is ω_0 if $\alpha = 0$ and $\omega_{\alpha+1}$ otherwise.

Example 2.10.12. It is not hard to see that the degree of our running example is ω_0. In fact, we will only treat term deduction systems with degree ω_0 in this survey. See, for instance, [Klusener, 1993] for rules that contain infinitely many premises.

Next, we will define this set of positive formulas for which it can be shown that it agrees with a given term deduction system. This definition originates from [Groote, 1990b] and is adapted to our situation by [Verhoef, 1994a].

Definition 2.10.13. Let $T = (\Sigma, D)$ be a term deduction system and let $S : PF(T) \longrightarrow \alpha$ be a stratification for an ordinal number α. We define a set $T_S \subseteq PF(T)$ as follows.

$$T_S = \bigcup_{i<\alpha} T_i^S, \quad T_i^S = \bigcup_{j<d(T)} T_{i,j}^S.$$

It will be useful to introduce the following notations for certain unions over T_i^S and $T_{i,j}^S$:

$$U_i^S = \bigcup_{i'<i} T_{i'}^S \ (i \leq \alpha), \quad U_{i,j}^S = \bigcup_{j'<j} T_{i,j'}^S \ \big(j \leq d(T)\big).$$

We drop the sub- and superscripts S and, for instance, render U_i^S as U_i and $T_S \vdash \varphi$ as $T \vdash \varphi$, provided no confusion arises. Now we define for all $i < \alpha$ and for all $j < d(T)$ the set $T_{i,j} = T_{i,j}^S$:

$$\begin{aligned} T_{i,j} \quad = \quad & \Big\{\varphi \mid S(\varphi) = i,\ \exists H/C \in D, \sigma : V \longrightarrow C(\Sigma) : \sigma(C) = \varphi, \\ & \quad \forall\, h \in PF(H) : U_{i,j} \cup U_i \vdash \sigma(h), \forall\, h \in NF(H) : U_i \vdash \sigma(h)\Big\}. \end{aligned}$$

The next theorem is taken from [Verhoef, 1994a] but its proof is essentially the same as a similar theorem of [Groote, 1990b]. It states that for a stratifiable term deduction system the set that we defined above agrees with it. Moreover, this is independent of the choice of the stratification.

Theorem 2.10.14. *Let* $T = (\Sigma, D)$ *be a term deduction system and let* $S : PF(T) \longrightarrow \alpha$ *be a stratification for an ordinal number* α. *Then* T_S *agrees with* T. *If* S' *is also a stratification for* T *then* $T_S = T_{S'}$.

Example 2.10.15. Since our running example is stratifiable it follows from the above theorem that the term deduction system $T(\text{BPA}_{\delta\theta})$ determines a transition relation (with predicates) on closed terms.

So, now we only know that when a term deduction system has a stratification there exists some set of positive formulas that agrees with it. Next, we are interested in the conditions under which strong bisimulation equivalence is a congruence relation. Just as in subsection 2.2.3 we define a syntactical restriction on a term deduction system. We will generalize the *path* format to the so-called *panth* format, which stands for "predicates and *ntyft/ntyxt* hybrid format". The *ntyft/ntyxt* format stems from [Groote, 1990b].

Definition 2.10.16. Let $T = (\Sigma, D)$ be a term deduction system with $D = D(T_p, T_r)$. Let in the following K, L, M, and N be index sets of arbitrary cardinality, let $s_k, t_l, u_m, v_n, t \in O(\Sigma)$ for all $k \in K$, $l \in L$, $m \in M$, and $n \in N$, let $P_k, P_m, P \in T_p$ be predicate symbols for all $k \in K$ and $m \in M$, and let $R_l, R_n, R \in T_r$ be relation symbols for all $l \in L$ and $n \in N$.

A deduction rule $d \in D$ is in *panth* format if it has one of the following four forms:

$$\frac{\{P_k s_k : k \in K\} \cup \{t_l R_l y_l : l \in L\} \cup \{\neg P_m u_m : m \in M\} \cup \{v_n \neg R_n : n \in N\}}{C}$$

- with $C = f(x_1, \ldots, x_n)Rt$, $f \in \Sigma$ an n-ary function symbol, $X = \{x_1, \ldots, x_n\}$, $Y = \{y_l : l \in L\}$, and $X \cup Y \subseteq V$ a set of distinct variables;
- with $C = xRt$, $X = \{x\}$, $Y = \{y_l : l \in L\}$, and $X \cup Y \subseteq V$ a set of distinct variables;
- with $C = Pf(x_1, \ldots, x_n)$, $X = \{x_1, \ldots, x_n\}$, $Y = \{y_l : l \in L\}$, and $X \cup Y \subseteq V$ a set of distinct variables; or
- with $C = Px$, $X = \{x\}$, $Y = \{y_l : l \in L\}$, and $X \cup Y \subseteq V$ a set of distinct variables.

A term deduction system is in *panth* format if all its rules are.

Example 2.10.17. It is not hard to verify that the deduction rules of our running example satisfy the *panth* format.

Next, we define the notion of strong bisimulation for term deduction systems with negative premises. In definition 2.2.28 we gave the positive case. This definition is based on [Park, 1981] and its formulation is taken from [Verhoef, 1994a].

Definition 2.10.18. Let $T = (\Sigma, D)$ be a term deduction system with stratification S and let $D = D(T_p, T_r)$. A binary relation $B \subseteq C(\Sigma) \times C(\Sigma)$ is called a (strong) bisimulation if for all $s, t \in C(\Sigma)$ with sBt the following conditions hold. For all $R \in T_r$

$$\forall s' \in C(\Sigma)\ (T_S \vdash sRs' \Rightarrow \exists t' \in C(\Sigma) : T_S \vdash tRt' \wedge s'Bt'),$$
$$\forall t' \in C(\Sigma)\ (T_S \vdash tRt' \Rightarrow \exists s' \in C(\Sigma) : T_S \vdash sRs' \wedge s'Bt'),$$

and for all $P \in S_p$

$$T_S \vdash Ps \Leftrightarrow T_S \vdash Pt.$$

The first two conditions are known as the transfer property. Two states s and $t \in C(\Sigma)$ are bisimilar if there exists a bisimulation relation containing the pair (s, t). If s and t are bisimilar we write $s \sim t$. Note that bisimilarity is an equivalence relation, called bisimulation equivalence.

At this point we have all the ingredients that we need to formulate the theorem that is interesting for our purpose: the congruence theorem for the *panth* format. It states that in many situations strong bisimulation equivalence is a congruence. The congruence theorem is taken from [Verhoef, 1994a] albeit that there the well-founded subcase is proved. [Fokkink, 1994] showed that this condition is not necessary. Thus, we dropped the extra assumption.

Theorem 2.10.19. *Let $T = (\Sigma, D)$ be a stratifiable term deduction system in panth format. Then strong bisimulation equivalence is a congruence for all function symbols.*

Example 2.10.20. Since the deduction rules of our running example are in *panth* format and since the term deduction system has a stratification, we find with the congruence theorem that strong bisimulation equivalence is a congruence.

According to the above example we find that the quotient of the closed $\text{BPA}_{\delta\theta}$ terms modulo bisimulation equivalence is well-defined; this means that the operators of $\text{BPA}_{\delta\theta}$ can be defined on this quotient. By a straightforward proof we can show that it is a model of $\text{BPA}_{\delta\theta}$.

We postpone the proof of the completeness of $\text{BPA}_{\delta\theta}$ until we have shown that it is a conservative extension of BPA_{δ}.

2.10.2 Conservativity

In this subsection we take care of the conservativity of $\text{BPA}_{\delta\theta}$ over BPA. We are used to proving this via the conservativity theorem for the *path* format but since the operational rules of $\text{BPA}_{\delta\theta}$ do not fit this format, we cannot simply apply this theorem. Just as with the conservativity of $\text{BPA}_{\delta\theta}$ (see subsection 2.10.1) over BPA we will generalize below the theory that we already treated on conservativity—yet another trip into the general theory on operational semantics. This time we will mainly extend the theory of section 2.4.1 so that we can also deal with negative premises. This subsection is based on [Verhoef, 1994b].

Since we treated some theory on negative premises and some theory on conservative extensions, their combination will be not too much work. We have to update the notions of pure, well-founded, and operationally conservative extension. Then only the operationally conservative extension theorem for the *path* format needs a little modification.

Below, we give the update of the notion pure. It was defined in the positive case in definition 2.2.31.

Definition 2.10.21. A deduction rule containing negative premises is pure if this rule is already pure when the negative premises are discarded. A term deduction system with negative premises is pure if all its deduction rules are pure.

Example 2.10.22. It is not hard to see that the term deduction system $T(\mathrm{BPA}_{\delta\theta})$ is pure.

Now, we update the definition of well-founded. This notion is defined in definition 2.4.13 for the positive case. The update is in the same vein as the one for the purity.

Definition 2.10.23. Let $T = (\Sigma, D)$ be a term deduction system and let F be a set of formulas. The variable dependency graph of F is a directed graph with variables occurring in F as its nodes. The edge $x \longrightarrow y$ is an edge of the variable dependency graph if and only if there is a positive relation $tRs \in F$ with $x \in var(t)$ and $y \in var(s)$.

The set F is called well-founded if any backward chain of edges in its variable dependency graph is finite. A deduction rule is called well-founded if its set of hypotheses is so. A term deduction system is called well-founded if all its deduction rules are well-founded.

Example 2.10.24. It is not hard to see that the term deduction system $T(\mathrm{BPA}_{\delta\theta})$ is well-founded.

Next, we update the notion of an operationally conservative extension. Also this definition does not look very different from its positive counterpart. Note that in the positive case proofs are well-founded trees, whereas in the negative case we use the notion of agreeing with. More information on this can be found in subsection 2.10.1. We also refer to this subsection for the definition of stratifiability.

Definition 2.10.25. Let $T^i = (\Sigma_i, D_i)$ be term deduction systems with $T = (\Sigma, D) := T^0 \oplus T^1$ defined. Let $D = D(T_p, T_r)$. The term deduction system T is called an operationally conservative extension of T^0 if it is stratifiable and for all $s, u \in C(\Sigma_0)$, for all relation symbols $R \in T_r$ and predicate symbols $P \in T_p$, and for all $t \in C(\Sigma)$ we have

$$T_S \vdash sRt \iff T^0_{S^0} \vdash sRt$$

and

$$T_S \vdash Pu \iff T^0_{S^0} \vdash Pu,$$

where S is a stratification for T and S^0 is a stratification for T^0 (take for instance S^0 to be the restriction of S to positive formulas of T^0).

Now we have all the updates of the definitions that we need in order to state the operationally conservative extension theorem for the *panth* format. The following theorem is taken from [Verhoef, 1994b].

Theorem 2.10.26. *Let $T^0 = (\Sigma_0, D_0)$ be a pure well-founded term deduction system in panth format. Let $T^1 = (\Sigma_1, D_1)$ be a term deduction system in panth format. If there is a conclusion sRt or Ps of a rule $d_1 \in D_1$ with $s = x$ or $s = f(x_1, \ldots, x_n)$ for an $f \in \Sigma_0$, we additionally require that d_1 is pure, well-founded, $t \in O(\Sigma_0)$ for premises tRy of d_1, and that there is a positive premise containing only Σ_0 terms and a new relation or predicate symbol. Now if $T = T^0 \oplus T^1$ is defined and stratifiable then T is an operationally conservative extension of T_0.*

Example 2.10.27. In subsection 2.10.1 we already showed that the term deduction system that belongs to $\mathrm{BPA}_{\delta\theta}$ is stratifiable. It is easy to verify the other conditions of the above theorem so we may conclude that $\mathrm{BPA}_{\delta\theta}$ is an operationally conservative extension of BPA.

In the above example we have shown the operational conservativity of $\mathrm{BPA}_{\delta\theta}$ over BPA. We are in fact interested in the equational conservativity. The other theorems, in particular the equationally conservative extension theorem, that we treated in subsection 2.4.1, do not need any updates, since in those theorems we only refer to term deduction systems and we do not specify which ones. [Verhoef, 1994b] showed that these theorems hold for term deduction systems with negative premises.

So, we can formulate and prove the following theorem.

Theorem 2.10.28. *The equational specification $\mathrm{BPA}_{\delta\theta}$ is an equationally conservative extension of BPA.*

Proof. Straightforward: check the conditions of theorem 2.4.24 and use example 2.10.27. ∎

Now that we have the conservativity result, the completeness of $\mathrm{BPA}_{\delta\theta}$ follows more or less from the completeness of BPA_{δ}. We will see this in the following theorem.

Theorem 2.10.29. *The equational specification $\mathrm{BPA}_{\delta\theta}$ is a complete axiomatization of the set of closed $\mathrm{BPA}_{\delta\theta}$ terms modulo bisimulation equivalence.*

Proof. Easy: use theorem 2.4.26. Note that the priority operator can be eliminated; see theorem 2.10.3. ∎

2.10.3 Extensions of $\mathrm{BPA}_{\delta\theta}$

In this subsection we discuss extensions of $\mathrm{BPA}_{\delta\theta}$ with the notions of recursion, projections, renaming, and/or the encapsulation operator, and the state operator. In fact, all extensions but the one with recursion can be obtained just as for the BPA or BPA_δ case.

Recursion The problem with the extension of $\mathrm{BPA}_{\delta\theta}$ with recursion is purely technical. Since there are negative premises in the operational characterization of the priority and unless operators, we introduced the notion of a stratification to ensure that the semantical rules indeed define a transition relation. We recall that in example 2.10.10 we give a stratification for the operational semantics of $\mathrm{BPA}_{\delta\theta}$. The map defined there counts the total of occurrences of θ and $\triangleleft$. This approach no longer works in the presence of the operational rules for recursion that we presented in table 6. We illustrate this with a simple example. Suppose that we have the following recursive specification:

$$E = \{X = a \cdot X + \theta(a)\}.$$

In this case, the operational rule takes the form

$$\frac{a \cdot \langle X|E\rangle + \theta(a) \xrightarrow{a} \langle X|E\rangle}{\langle X|E\rangle \xrightarrow{a} \langle X|E\rangle}.$$

So with the above stratification we have that the stratification of the premise is *not* less than or equal to the stratification of the conclusion. To solve this problem we use infinite ordinals. We adapt the stratification as follows:

$$S(t \xrightarrow{a} t') = \omega \cdot n + m,$$

where n is the number of *unguarded* occurrences of $\langle$ and m is the total number of occurrences of θ and occurrences of $\triangleleft$ (so the m part is the original stratification). With the modified stratification, the problem is solved. We leave it as an exercise to the reader to check the details.

Projection The extension of $\mathrm{BPA}_{\delta\theta}$ with projection is obtained in the same way as this extension for BPA; see subsection 2.4.

Renaming and encapsulation It is straightforward to extend the equational specification $\mathrm{BPA}_{\delta\theta}$ with renaming operators or the encapsulation operator; cf. subsection 2.7.3.

State operator The extension of the theory $\mathrm{BPA}_{\delta\theta}$ with either the simple or extended state operator is obtained in the same way as for the theory BPA; see subsections 2.8 and 2.9.

Inconsistent combinations Remarkably, if we combine recursion with the equational specification $\mathrm{BPA}_{\delta\theta}$ plus renaming operators we will find an inconsistency. [Groote, 1990b] gives the following example. Take a renaming function f such that

- $f(b) = a$,
- $f(a) = c$,
- $f(d) = d$ for all $d \in A \setminus \{a, b\}$.

Consider the recursive equation

$$X = \theta\big(\rho_f(X) + b\big).$$

Now it can be shown that

$$X \xrightarrow{b} \surd \iff X \not\xrightarrow{b} \surd$$

if we take $a > b$ as the partial ordering on the atomic actions.

Observe also that the combination of recursion with $\mathrm{BPA}_{\delta\theta}$ plus state operators is inconsistent since state operators are a generalization of renamings.

2.11 Basic process algebra with iteration

In this subsection we extend basic process algebra with an iterative construct. This construct is, in fact, Kleene's star operator, a binary infix operator denoted *. We will call this operator *Kleene's binary star operator*, since there are two versions of Kleene's star operator: one unary and one binary. The binary construct originates from [Kleene, 1956] and its more commonly known unary version is due to [Copi *et al.*, 1958]. This subsection is based on the papers [Bergstra *et al.*, 1994a] and [Fokkink and Zantema, 1994].

We want to note that using iteration we can also define infinite processes. We already discussed recursion, the standard way to define infinite processes, in subsection 2.3. The advantage of the approach that we explain in this subsection is that there is no need for proof rules like the recursive definition principle or the recursive specification principle, to guarantee that a recursive specification has a possibly unique solution. In this setting, the recursive construct is just some binary operator that we may add to a process language.

The theory The equational specification BPA^* consists of the signature of BPA and a binary infix operator *, called Kleene's binary star operator. Its equations are the ones of BPA plus the axioms in table 37.

We will comment on these axioms. The first one BKS1 is the defining equation for the star operator that [Kleene, 1956] gives in the context of

Table 37. The axioms defining Kleene's binary star operator.

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

finite automata. Only the notation is adapted to the present situation. The second equation originates from [Bergstra *et al.*, 1994a]; it is a simple equation needed for the completeness. The third axiom BKS3 is more sophisticated; it stems from [Troeger, 1993]. Troeger used this equation for a slightly different process specification formalism.

Next, we will show some properties that can be derived from the equational specification BPA*. For instance, if we apply Kleene's axiom to the first term in the display below we find a term to which we can apply Troeger's axiom with $x+y$ substituted for y. Thus, this yields the following identity:

$$\begin{aligned} x^*\big((x+y)^*z\big) &= x^*\Big((x+y)\big((x+y)^*z\big) + z\Big) \\ &= (x+y)^*z. \end{aligned}$$

The next identity expresses that applying the star operator in a nested way for the same process reduces to applying it once. First, we apply Kleene's axiom, then we use the idempotence of the alternative composition, then we use Troeger's identity, and then one application of idempotence finishes the calculation. We display this below.

$$\begin{aligned} x^*(x^*y) &= x^*\big(x(x^*y) + y\big) \\ &= x^*\Big(x\big((x+x)^*y\big) + y\Big) \\ &= (x+x)^*y \\ &= x^*y. \end{aligned}$$

Semantics We give the semantics of the equational specification BPA* by means of a term deduction system $T(\text{BPA}^*)$. Its signature is the signature of BPA*. Its deduction rules are the rules for BPA that we met many times before (see table 5) plus the rules that characterize Kleene's binary star operator. We list them in table 38.

Theorem 2.11.1. *The set of closed* BPA* *terms modulo strong bisimulation equivalence is a model of* BPA*.

Proof. Since bisimulation equivalence is a congruence, we only need to check the soundness of the axioms of BPA*. The first five axioms are already treated in the soundness theorem for BPA (see 2.2.35). So it suffices to prove the soundness of the three remaining equations. The case BKS1 is proved analogously to the case A1: take as relation the pair $\big(x(x^*y)+y, x^*y\big)$ and the diagonal. Now it is not hard to show that this is a bisimulation relation. For the equation BKS2 we have the following relation: relate all terms of the form $x^*(yz)$ with $(x^*y)z$; relate each term of the form $x' \cdot \big(x^*(y \cdot z)\big)$ with $\big(x' \cdot (x^*y)\big) \cdot z$; and relate each term with itself. We leave it to the reader to verify that this relation is a bisimulation relation. The verification of the soundness of Troeger's axiom is obtained analogously to the verification of equation BKS2. ∎

It is easy to see that BPA* is a conservative extension of BPA; see subsection 2.4.1. However, we cannot eliminate Kleene's binary star operator. See subsection 2.14 where we discuss expressivity results. This can be easily seen as follows. Call a term deduction system T operationally terminating if there are no infinite reductions

$$T \vdash s_0 R_0 s_1, T \vdash s_1 R_1 s_2, \ldots$$

possible. It is easy to see by inspection of the operational rules for BPA that its term deduction system is operationally terminating (cf. lemma 2.2.36 where a "weight" function is defined). It is also easy to see that the term deduction system belonging to BPA* is not operationally terminating. We have, for instance, the infinite reduction

$$a^*b \xrightarrow{a} a^*b \xrightarrow{a} a^*b \xrightarrow{a} \ldots.$$

Now suppose that Kleene's binary star operator can be eliminated in favour of the operators of BPA. Then a^*b must be bisimilar to a BPA term, say t. Because of the bisimilarity with a^*b we must have that t can mimic the above steps that a^*b is able to perform. So t must have an infinite reduction. This contradicts the fact that the semantics of BPA is operationally terminating.

Table 38. Operational rules for Kleene's binary star operator.

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

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

Table 39. The axioms defining the discrete time unit delay.

$\sigma_d(x) + \sigma_d(y) = \sigma_d(x+y)$	DT1
$\sigma_d(x) \cdot y = \sigma_d(x \cdot y)$	DT2

As a corollary, we cannot use the completeness theorem 2.4.26 to prove the completeness of BPA^*. The proof that our axiomatization is nevertheless complete is due to [Fokkink and Zantema, 1994] and is beyond the scope of this chapter. The reason for this is that the proof makes use of a sophisticated term rewriting analysis. Below, we will list their main result.

Theorem 2.11.2. *The equational specification* BPA^* *is a complete axiomatization with respect to strong bisimulation equivalence.*

Proof. See the paper [Fokkink and Zantema, 1994]. ■

Extensions of BPA^* The extension of BPA^* with deadlock (BPA^*_δ) is as usual. This system is obtained by taking the syntax of BPA_δ plus Kleene's binary operator *. The axioms of BPA^*_δ are the ones of BPA^* plus those for deadlock.

Since BPA^*_δ is more expressive (see subsection 2.14) than BPA^* we cannot use the usual machinery to prove basic properties such as completeness. There is no completeness result for the system BPA^*_δ so we will not discuss the extensions of BPA^* (or BPA^*_δ) with the notions that we usually extend our systems with. Moreover, at the time of writing this survey the only studied extensions of BPA^* are those with abstraction, fairness principles, deadlock, and parallel constructs. We will discuss some of these extensions after we have introduced such parallel constructs.

2.12 Basic process algebra with discrete relative time

Now, we treat an extension of BPA with a form of discrete relative time; we abbreviate this as $\mathrm{BPA}_{\mathrm{dt}}$. We speak of discrete time since the system works with so-called time slices. It is called relative since the system refers to the current time slice, the next time slice, and so on. $\mathrm{BPA}_{\mathrm{dt}}$ stems from [Baeten and Bergstra, 1992a]. For other approaches to discrete time process algebra we refer to [Moller and Tofts, 1990] and [Nicollin and Sifakis, 1994].

Theory The equational specification $\mathrm{BPA}_{\mathrm{dt}}$ has as its signature the one of BPA and a unary function called discrete time unit delay, which is denoted σ_d. The σ is some fixed symbol, which is a measure for the delay. The axioms of $\mathrm{BPA}_{\mathrm{dt}}$ are the ones of BPA that we listed in table 1 plus the equations defining the discrete time unit delay; see table 39.

We denote the atomic action a in the current time slice by $\underline{\underline{a}}$. We distinguish $\underline{\underline{a}}$ from a because also other embeddings of BPA into $\mathrm{BPA}_{\mathrm{dt}}$

are possible, where a is interpreted as a occurs at some time, that is, we have $a = \underline{\underline{a}} + \sigma_d(a)$. The intended interpretation of the unary operator $\sigma_d(x)$ is that it pushes a process x to the next time slice. The length of a time slice is measured with the positive real σ. This is operationally expressed by the rule $\sigma_d(x) \xrightarrow{\sigma} x$, where $\xrightarrow{\sigma}$ is a special relation that describes the pushing behaviour. Note that the label σ is *not* part of the signature of $\mathrm{BPA_{dt}}$.

Axiom DT1 is called "time factorizing axiom". It expresses that the passage of time by itself cannot determine a choice. We note that the form of choice here is called "strong choice" (the other two approaches mentioned above have weak choice), so in $\underline{\underline{a}} + \sigma_d(\underline{\underline{b}})$ both a in the current time slice and b in the next time slice are possible. We do have in the closed term above that by moving to the next time slice, we disable a.

Next, we will show that the term rewriting system associated to $\mathrm{BPA_{dt}}$ is terminating. Although this result has importance of its own, we cannot use it to prove an elimination result. For the discrete time unit delay cannot be eliminated.

Theorem 2.12.1. *The term rewriting system that is associated to* $\mathrm{BPA_{dt}}$ *is strongly normalizing. This system consists of the rules in tables 2 and 40.*

Proof. We use the method of the recursive path ordering that we treated in subsection 2.2.2. Take as precedence for the operations in the signature $\cdot > + > \sigma_d$ and give the sequential composition the lexicographical status for the first argument. As an example, we treat RDT2.

$$\begin{aligned} \sigma_d(x) \cdot y \quad &>_{lpo} \quad \sigma_d(x) \cdot^* y \\ &>_{lpo} \quad \sigma_d\big(\sigma_d(x) \cdot^* y\big) \\ &>_{lpo} \quad \sigma_d\big(\sigma_d^*(x) \cdot y\big) \\ &>_{lpo} \quad \sigma_d(x \cdot y). \end{aligned}$$

The other rule is dealt with just as simply. ∎

Now that we know that the term rewriting system associated to $\mathrm{BPA_{dt}}$ is terminating, we discuss what form the normal forms can take. Following [Baeten and Bergstra, 1992a], we define these normal forms, called basic terms.

Definition 2.12.2. In order to define inductively the set of basic terms,

Table 40. The rewrite rules for the discrete time unit delay.

$\sigma_d(x) + \sigma_d(y) \to \sigma_d(x + y)$	RDT1
$\sigma_d(x) \cdot y \to \sigma_d(x \cdot y)$	RDT2

we need the auxiliary notion of an A-basic term: a $\mathrm{BPA_{dt}}$ term with no leading σ_d. We define both notions simultaneously.

- every A-basic term is a basic term;
- for each $a \in A$, $\underline{\underline{a}}$ is an A-basic term;
- if $a \in A$ and t is a basic term, then $\underline{\underline{a}} \cdot t$ is an A-basic term;
- if t and s are A-basic terms, then $t + s$ is an A-basic term;
- if t is a basic term, then $\sigma_d(t)$ is a basic term;
- if t is an A-basic term and s is a basic term, then $t + \sigma_d(s)$ is a basic term.

Next, we formulate some facts from [Baeten and Bergstra, 1992a]. They can be easily proved.

Theorem 2.12.3.

- *Let t be a closed* $\mathrm{BPA_{dt}}$ *term. Then there exists a basic term s such that* $\mathrm{BPA_{dt}} \vdash t = s$.
- *An A-basic term takes the form:* $\sum_{i<n} \underline{\underline{a}}_i \cdot t_i + \sum_{j<m} \underline{\underline{b}}_j$ *with* $n+m > 0$, $a_i, b_j \in A$, *and t_i basic.*
- *A basic term is either an A-basic term or of the form $t + \sigma_d(s)$ with t and s A-basic terms.*

Semantics Next, we formally define the semantics by way of a term deduction system for $\mathrm{BPA_{dt}}$. The signature of this system consists of the one for the equational specification $\mathrm{BPA_{dt}}$. The deduction rules are those for $+$ and $\cdot$ of BPA in table 1, and the rules for constants and the discrete time unit delay in table 41.

Note the appearance of negative premises in the operational rules. By means of the theory that we discussed in subsection 2.10.1, we can find that this system indeed defines a set of positive formulas. We recall that we, therefore, have to find a stratification. Let n be the number of $+$ signs that occurs in a closed $\mathrm{BPA_{dt}}$ term t. Then we define a stratification S by assigning to $t \xrightarrow{\sigma} s$ and to $t \xrightarrow{\sigma} \surd$ the number n. For the other formulas φ we simply define $S(\varphi) = 0$. It is not hard to see that this function is a stratification. So the term deduction system is well-defined.

Table 41. The operational semantics for the discrete time unit delay.

$\underline{\underline{a}} \xrightarrow{a} \surd$	$\sigma_d(x) \xrightarrow{\sigma} x$	$\dfrac{x \xrightarrow{\sigma} x'}{x \cdot y \xrightarrow{\sigma} x' \cdot y}$
$\dfrac{x \xrightarrow{\sigma} x',\ y \xrightarrow{\sigma} y'}{x + y \xrightarrow{\sigma} x' + y'}$	$\dfrac{x \xrightarrow{\sigma} x',\ y \not\xrightarrow{\sigma}}{x + y \xrightarrow{\sigma} x'}$	$\dfrac{x \not\xrightarrow{\sigma},\ y \xrightarrow{\sigma} y'}{x + y \xrightarrow{\sigma} y'}$

It is easy to see that the deduction rules are in *panth* format (see subsection 2.10.1), so we find with theorem 2.10.19 that strong bisimulation equivalence is a congruence.

Theorem 2.12.4. *The equational specification* $\mathrm{BPA_{dt}}$ *is a sound axiomatization of the set of closed* $\mathrm{BPA_{dt}}$ *terms modulo strong bisimulation equivalence.*

Proof. We can give the soundness proof along the usual lines: the soundness of the BPA axioms is already done and the soundness of the equations DT1–2 can be obtained just as the case A1. ∎

Theorem 2.12.5. *The equational specification* $\mathrm{BPA_{dt}}$ *is a complete axiomatization of the set of closed* $\mathrm{BPA_{dt}}$ *terms modulo strong bisimulation equivalence.*

Proof. Usually, we prove the completeness with theorem 2.4.26. However, in this case we cannot apply our routine approach. This is due to the fact that the discrete time unit delay cannot be eliminated; for instance, the term $\sigma_d(\underline{a})$ cannot be reduced any further. So we cannot apply our completeness theorem 2.4.26, since there we assume that extra operators can be eliminated.

To prove the completeness we follow [Baeten and Bergstra, 1992a]. Their idea is to define a bijective mapping between $\mathrm{BPA_{dt}}$ terms and BPA terms; the completeness of $\mathrm{BPA_{dt}}$ now follows from the completeness of BPA. Next, we will work out their idea.

The equational specification $\mathrm{BPA_{dt}}$ is parameterized with a set of atomic actions A; we write $\mathrm{BPA_{dt}}(A)$. Similarly, the theory BPA is parameterized in this way. Since there is mostly no confusion with which set our systems are equipped, we omit them often—but not in this case, since we parameterize BPA with $A_\sigma = A \cup \{\sigma\}$, where $\sigma \notin A$ is an atomic action (with a suggestive name). So let φ from $\mathrm{BPA_{dt}}(A)$ to $\mathrm{BPA}(A_\sigma)$ be inductively defined as follows:

- $\varphi(\underline{a}) = a$,
- $\varphi(x + y) = \varphi(x) + \varphi(y)$,
- $\varphi(x \cdot y) = \varphi(x) \cdot \varphi(y)$,
- $\varphi\big(\sigma_d(x)\big) = \sigma \cdot \varphi(x)$.

Now, suppose that we have two bisimilar $\mathrm{BPA_{dt}}$ terms s and t. With the aid of theorem 2.12.3 we may assume that s and t are basic terms. So we find that $\varphi(s)$ and $\varphi(t)$ are also bisimilar. With the completeness theorem 2.2.37 for BPA we find that $\mathrm{BPA}(A_\sigma) \vdash \varphi(s) = \varphi(t)$. Using the inverse mapping of φ, we can mimic each step of this proof by a step in $\mathrm{BPA_{dt}}$. So we find that $\mathrm{BPA_{dt}} \vdash s = t$. ∎

Next, we formulate a conservativity result for $\mathrm{BPA_{dt}}$.

Theorem 2.12.6. *The equational specification* $\mathrm{BPA_{dt}}$ *is a conservative extension of* BPA *(using* $\underline{\underline{a}}$ *instead of* a*).*

Proof. This can be easily shown using the theory that we discussed in subsection 2.10.2.

We note that this result is due to [Verhoef, 1994b]. ■

2.12.1 Extensions of $\mathrm{BPA_{dt}}$

We will only discuss the extension of $\mathrm{BPA_{dt}}$ with deadlock and recursion. We will not treat the other extensions that we usually have. The reason for this is that at the time of writing this survey these have not been formulated.

First, we will discuss how to extend $\mathrm{BPA_{dt}}$ with deadlock and then we discuss the extension with recursion.

Deadlock We can extend $\mathrm{BPA_{dt}}$ with deadlock in the usual way. We abbreviate this equational specification as $\mathrm{BPA_{\delta dt}}$. The axioms for $\underline{\underline{\delta}}$ are the usual ones for deadlock; see table 10. The termination proof is a combination of these proofs for $\mathrm{BPA_{dt}}$ and $\mathrm{BPA_{\delta}}$. The notion of a basic term needs a little modification: $\underline{\underline{\delta}}$ is an A-basic term. The operational semantics is the same as the one for $\mathrm{BPA_{dt}}$. The soundness and completeness are proved along the same lines as the case $\mathrm{BPA_{dt}}$. The conservativity of $\mathrm{BPA_{\delta dt}}$ over $\mathrm{BPA_{(\delta)}}$ is obtained as usual.

Recursion The extension of $\mathrm{BPA_{dt}}$ with recursion has the same technical problem as the extension of $\mathrm{BPA_{\delta\theta}}$ with recursion. We recall that the problem is that we need to define a new stratification on the operational rules of $\mathrm{BPA_{dt}}$ with recursion in order to guarantee that the transition relation is well-defined. For a solution we refer to subsection 2.10.3 where extensions of $\mathrm{BPA_{\delta\theta}}$ are discussed.

2.13 Basic process algebra with other features

When we want to describe parallel or distributed systems, the most important extensions are the ones with some form of parallel composition. We devote section 3 to such extensions. Below, we list a number of other extensions that we will not cover in this survey. We remark that this list is incomplete and in random order.

Abstraction In this survey we only treat concrete process algebra, hence any extension of the systems that we discuss with some notion of abstraction will not be covered by this survey. For more information on process algebras that incorporate abstraction we mention [Bergstra and Klop, 1985] that treats an extension of BPA with abstraction. Other systems that feature abstraction are CCS [Milner, 1980; Milner, 1989], Hennessy's system [Hennessy, 1988], and CSP [Hoare, 1985]. We note that the latter two systems are not extensions of BPA but treat basic notions in a different way.

But also CCS is not an extension of BPA because there is no sequential composition in CCS.

There are many other process algebras (with abstraction) such as CIRCAL [Milne, 1983], MEIJE [Austry and Boudol, 1984], SCCS [Milner, 1983], and the π-calculus [Milner *et al.*, 1992] to mention some.

Backtracking A well-known notion in logic programming is backtracking. [Bergstra *et al.*, 1994c] extended process algebra with this notion. They discuss an algebraic description of backtracking by means of a binary operator. For more details on this extension we refer to [Bergstra *et al.*, 1994c].

Combinatory logic In [Bergstra *et al.*, 1994b], process algebra is extended with combinatory logic. An interesting point of this combination is the possibility to verify the well-known alternating bit protocol without any conditional axiom, that is, the verification is purely equational. For more information on this combination and the equational verification we refer to [Bergstra *et al.*, 1994b].

Real-time In recent years, much effort has been spent on the extension of several process algebras with a notion of time. We discuss in this survey just one such extension: process algebra with relative discrete time. However, there are many more (concrete) extensions present in the literature. We mention the distinction between relative and absolute time, and the choice of the time domain: discrete or dense. We refer to [Klusener, 1993] for more information on real time process algebra in many and diverse forms.

Real-space In [Baeten and Bergstra, 1993] a form of real time process algebra is extended with real space. The paper surveys material from former reports on this topic. We refer the interested reader to [Baeten and Bergstra, 1993] for more information.

Nesting In this survey, we discuss the extension of process algebra with iteration, or Kleene's binary star. An extension that we do not discuss is one with an operator called the nesting operator. Like Kleene's binary star operator, the nesting operator also is a recursive operator (though it defines irregular recursion). We refer to [Bergstra *et al.*, 1994a] for more information on this topic.

Signals In [Baeten and Bergstra, 1992b] process algebra is extended with stable signals. These are attributes of states of a process. They introduce a signal insertion and a signal termination operator to be able to describe signals with a certain duration. A typical example that can be described with this theory is a traffic light system. For more information on the extension with signals we refer to [Baeten and Bergstra, 1992b].

Conditionals An extension with conditionals or guards can be found in the just mentioned paper [Baeten and Bergstra, 1992b]. They introduce an if-then-else operator in the notation of [Hoare *et al.*, 1987]. [Baeten and

Bergstra, 1992b] also introduce a variant of this conditional operator, called the guarded command that originates from [Baeten *et al.*, 1991]. [Groote and Ponse, 1994] developed a substantial amount of theory for a similar conditional construct called a guard.

For more information on the extension of BPA with conditional constructs we refer to the above papers.

Invariants and assertions Often, it is useful to have a connection between algebraic expressions and expressions in a logical language. Logical formulas can be used to express invariants (see [Bezem and Groote, 1994]) or as assertions (see [Ponse, 1991]).

Probabilities Often, systems exhibit behaviour that is probabilistic or statistical in nature. For example, one may observe that a faulty communication link drops a message 2% of the time. Algebraic formulations of probabilistic behaviour can be found in [Baeten *et al.*, 1992], [Giacalone *et al.*, 1990], [Larsen and Skou, 1992], and [Tofts, 1990], to mention some.

2.14 Decidability and expressiveness results in BPA

In this subsection we briefly mention decidability and expressiveness issues for the family of process algebras that we have introduced thus far.

2.14.1 Decidability

In our case, the decidability problems concern the question whether or not two finitely specified processes in, for instance BPArec, are bisimilar; see [Baeten *et al.*, 1993], [Caucal, 1990], and [Christensen *et al.*, 1992]. Informally, we refer to this as the question whether or not BPArec is decidable. It turns out that BPArec is decidable for all guarded processes; see [Christensen *et al.*, 1992]. For almost all extensions of BPA the decidability problem is open. Only for some extensions of BPArec with the state operator we have some information at the time of writing this survey. We refer the interested reader to [Baeten and Bergstra, 1991] and [Blanco, 1995] for more details on the systems $\lambda(\mathrm{BPA}_\delta\mathrm{rec})$ and $\mathrm{BPA}_{\delta\lambda}\mathrm{rec}$ and their decidability problems.

The following theorem is taken from [Christensen *et al.*, 1992]. The proof of this theorem is beyond the scope of this survey.

Theorem 2.14.1. *Bisimulation equivalence is decidable for all* BPArec *processes that can be specified with a finite guarded recursive specification.*

2.14.2 Expressiveness

For the family of systems that we introduced it is natural to address the question of expressivity. The result that is known states that BPArec can express non-regular processes. So, we first need to know what exactly are regular processes. This well-known definition is formulated below and is taken from [Baeten and Weijland, 1990]. Roughly, a process is regular if it has a finite graph.

First, we define when a process is regular in some model.

Definition 2.14.2. Let x be a process in some model M of BPArec. Define the relations $\cdot\xrightarrow{a}\cdot$ on this model as follows:

- $x\xrightarrow{a}y \iff M \models x = x + ay$,
- $x\xrightarrow{a}\surd \iff M \models x = x + a$.

A process y is called a subprocess of x if y is reachable from x; reachability means that there is a path of the following form that begins in x and ends in y:

$$x\xrightarrow{a_1}x_1\xrightarrow{a_2}\ldots\xrightarrow{a_n}y.$$

See also the definition of reachability in a term deduction system 2.2.23.

We say that x is a regular process (for the model M) if x has only finitely many subprocesses.

Next, we define when a guarded recursive specification is linear. It will turn out that a regular process can always be specified by a finite linear specification.

Definition 2.14.3. Let E be a recursive specification with variables from the set V. The specification E is called linear if every recursion equation in E is of the form:

$$X = \sum_{i<n} a_i \cdot X_i + \sum_{j<m} b_j,$$

for certain atomic actions a_i and b_j and variables $X, X_i \in V$ $(n + m > 0$ and $n, m \in \mathbb{N})$.

We call a recursion equation linear if it takes the above form. Note that every linear specification is guarded.

Lemma 2.14.4. *Let M be a model of* BPArec. *A process x is regular for M if and only if there exists a finite linear specification with x as solution.*

Proof. Sketch. We can turn each model into a graph model with definition 2.14.2. Now given a regular process x, we turn it into a finite graph. This graph determines a finite linear specification of which x is a solution.

Vice versa, let E be a finite linear specification. We can easily associate a finite graph to E, which in turn represents a regular process. (For instance, in the next example we turn a recursive specification into a graph using the above method.) ∎

Next, we show that there is a non-regular process that is finitely expressible in the theory BPArec, namely a counter.

Example 2.14.5. Consider the following guarded recursive specification. We call the process C a counter.

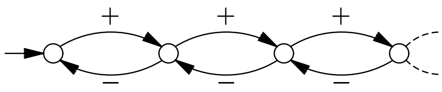

Fig. 6. The deduction graph of the counter C.

$$\begin{array}{rcl} C & = & T \cdot C \\ T & = & \mathit{plus} \cdot T' \\ T' & = & \mathit{minus} + T \cdot T'. \end{array}$$

We give the deduction graph (see definition 2.2.24) of C in figure 6. Note that we use $+$ for *plus* and $-$ for *minus*.

It is well known that the counter is a non-regular process. It has infinitely many distinct states, since for each n there is a state where n consecutive *minus* steps can be executed but not $n+1$.

Recursion versus iteration In subsection 2.3 we discussed the extension of BPA with recursion. In subsection 2.11 we discussed a similar construct: iteration. We can compare both approaches in the following sense: BPA* is less expressive than BPAlin. BPAlin is BPArec where only finite linear specifications are allowed[1]. In other words, BPA* does not contain non-regular processes. In [Bergstra *et al.*, 1994a] a simple example is given that shows the strictness of the inclusion. Consider the following regular process:

$$\begin{array}{rcl} X & = & a \cdot Y + b, \\ Y & = & c \cdot X + d. \end{array}$$

In figure 7 we give the graph that belongs to this process. This process is not definable in BPA*. In the next theorem we summarize the results. For the proof we refer to [Bergstra *et al.*, 1994a].

Theorem 2.14.6. BPA* *is strictly less expressive than* BPAlin. *There is a regular process that cannot be defined in* BPA*.

Remark 2.14.7. We refer to theorem 3.7.8 for more expressivity results concerning BPA* and BPA$^*_\delta$ and systems that we have not seen yet. For now we state that, in general, BPA* is less expressive than BPA$^*_\delta$.

3 Concrete concurrent processes

Up to now, we have discussed the language BPA with many of its extensions. Next, we want to discuss an extension of such significance that we devote a

[1]Note that each recursively specifiable process over ACP can also be specified with a possibly infinite number of linear equations. Hence the finiteness constraint.

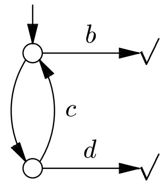

Fig. 7. The deduction graph of a regular process.

new section to it. It is the notion of parallelism or concurrency that we add to our family of basic systems that we treated in the previous section. We will restrict ourselves to concrete concurrency; that is, we do not consider abstraction.

We follow the ACP approach of [Bergstra and Klop, 1984b]. For other approaches to concurrency we refer to Milner's CCS [Milner, 1980; Milner, 1989], [Hennessy, 1988], and Hoare's CSP [Hoare, 1985].

3.1 Introduction

In this section, we first extend the BPA family with a parallel construct without interaction; that is, processes can be put in parallel by means of this operator but they cannot communicate with each other. This system is called PA. Then we will extend this theory with extensions that we discussed in the case of BPA (and new extensions). It will turn out that in most cases the extensions can be obtained in the same way as in the BPA case.

Secondly, we extend the parallel construct itself such that communication between parallel processes is also possible, that is, we discuss the system ACP. Then we discuss extensions of ACP, which is in most cases an easy job since they can be obtained in the same way as the extensions for BPA.

Finally, we discuss decidability and expressiveness issues for various systems.

3.2 Syntax and semantics of parallel processes

In this subsection we will describe the syntax and semantics of concrete concurrent processes.

3.2.1 The theory PA

We will discuss the equational theory $\mathrm{PA} = (\Sigma_{\mathrm{PA}}, E_{\mathrm{PA}})$. This section is based on [Bergstra and Klop, 1982].

The signature Σ_{PA} consists of the signature of BPA plus two binary operators $\|$ and $\mathbin{\|\!\_}$. The operator $\|$ is called *(free) merge* or *parallel composition* and the operator $\mathbin{\|\!\_}$ is called *left merge*. The left merge was introduced in [Bergstra and Klop, 1982] in order to give a finite axiomatization for the free merge. [Moller, 1989] proved that it is impossible to give a finite axiomatization of the merge without an auxiliary operator.

The set of equations E_{PA} consists of the equations of BPA in table 1 and the axioms concerning the merge in table 42. We assume in this table that a ranges over the set of atomic actions. So axioms M2 and M3 are in fact axiom schemes: for each atomic action there are axioms M2 and M3.

We assume that sequential composition binds stronger than both merges and they in turn bind stronger than the alternative composition. So, for instance, the left-hand side of M3 stands for $(a \cdot x) \mathbin{\|\!\_} y$ and the brackets in the left-hand side of M4 are necessary.

Intuition Before we provide the semantics of PA, we give an intuitive meaning to the non-BPA part of PA: the part concerning both merges. We already discussed the BPA part informally in 2.2.1. We recall that we consider the execution of an atomic action to occur at (or to be observed at) a point in time. We start with the signature and then we treat the axioms.

We think of the merge of two processes x and y as the process that executes both x and y in parallel. We think of the left merge of x and y as precisely the same, with the restriction that the first step of the process $x \mathbin{\|\!\_} y$ comes from its left-hand side x. We disregard the simultaneous execution of atomic actions here (but see subsection 3.5 where communication comes into play). This leads to the so-called interleaving view, which clarifies the behaviour of the left merge.

This intuition clarifies that axiom M1 is defined in terms of the left merge: the merge of two processes starts either with the left-hand side or with its right-hand side.

The remaining axioms M2–4 define the left merge following the structure of basic terms.

The parallelism in axiom M2 collapses into sequential composition since the first step at the left-hand side is also the last one. After the first step in M3, we obtain full parallelism for the remainders. Axiom M4 simply says that the left merge distributes over the alternative composition. Note that, in general, $(x + y) \parallel z \neq x \parallel z + y \parallel z$. So, here we describe an interleaving parallel composition. Also, other forms of parallel composition can be formulated. We already mentioned interleaving extended with simultaneous execution, to be discussed from subsection 3.5 on, but also want to mention so-called *synchronous* parallel composition, by which we can describe clocked systems, where all components proceed in lock-step. A well known process algebra with synchronous parallel composition is SCCS [Milner, 1983], two references using the present framework are [Bergstra and Klop, 1984b] and [Weijland, 1989].

Structural induction We can use structural induction for PA as before for BPA, since basic PA terms are just basic BPA terms. This follows

Table 42. Axioms for the free merge.

$x \parallel y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x$	M1
$a \mathbin{\|\!\_} x = ax$	M2
$ax \mathbin{\|\!\_} y = a(x \parallel y)$	M3
$(x + y) \mathbin{\|\!\_} z = x \mathbin{\|\!\_} z + y \mathbin{\|\!\_} z$	M4

immediately from the theorem to follow (theorem 3.2.4). It states that parallel composition can be eliminated from closed PA terms.

Termination Next, it is our aim to prove that the term rewriting system associated to the equational specification PA is strongly normalizing. In subsection 2.2.2 we already discussed the powerful method of the recursive path ordering. Indeed, we will use this method to prove the desired result but we cannot apply it immediately. We recall that the termination problem more or less reduces to finding the appropriate strict partial ordering on some operators in the signature. The problem that we have with this particular system is that we cannot define a consistent partial ordering on the elements of the signature. First, we will explain this problem and then we will see that a possible solution can be obtained in the same way as for the termination of BPA_λ; see section 2.8.1. The problematical pair consists of the rules (RM1, RM3). Analysing this pair we find that if we take the rule RM1 on the one hand, the ordering that does the job is $\parallel \; > \mathbin{\|\!\_}$. On the other hand, if we look at RM3, the right choice is the other way around: $\mathbin{\|\!\_} > \parallel$. This particular problem is tackled by [Bergstra and Klop, 1985]. More detailed information on this problem can be found in a survey on term rewriting that appeared in this series [Klop, 1992, remark 4.11(*ii*)]. The idea of [Bergstra and Klop, 1985] was to equip the operators $\parallel$ and $\mathbin{\|\!\_}$ with a rank. Thus yielding a ranked signature for which it is possible to define the desired strict partial ordering. To formalize the ranked signature we first need a notion termed "weight". Its definition stems from [Bergstra

Table 43. A term rewriting system for PA.

$(x + y)z \to xz + yz$	RA4
$(xy)z \to x(yz)$	RA5
$x \parallel y \to x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x$	RM1
$a \mathbin{\|\!\_} x \to ax$	RM2
$ax \mathbin{\|\!\_} y \to a(x \parallel y)$	RM3
$(x + y) \mathbin{\|\!\_} z \to x \mathbin{\|\!\_} z + y \mathbin{\|\!\_} z$	RM4

and Klop, 1985]. Note that we already defined this notion in the case of BPA with the state operator; see definition 2.8.1.

Definition 3.2.1. Let x and y be terms and let a be an atomic action. The weight of a term x, notation $|x|$, is defined inductively as follows:

- $|a| = 1$
- $|x + y| = \max\{|x|, |y|\}$
- $|x \cdot y| = |x| + |y|$
- $|x \parallel y| = |x| + |y|$
- $|x \mathbin{\|\!\_} y| = |x| + |y|$.

Below we give the definition of a ranked operator as defined by [Bergstra and Klop, 1985]. And we list the new signature.

Definition 3.2.2. The rank of an operator $\parallel$ or $\mathbin{\|\!\_}$ is the weight of the subterm of which it is the leading operator. The signature for the term rewriting system associated with PA is the following:

$$A \cup \{+, \cdot\} \cup \{\parallel_n, \mathbin{\|\!\_}_n : n \geq 2\},$$

where the subscripted n stands for the sum of the weights of the arguments.

Now that we are equipped with the right tools we formulate the termination theorem for the system PA.

Theorem 3.2.3. *The term rewriting system associated to* PA *(see table 43) is strongly normalizing.*

Proof. We will give the partial ordering so that we can use the method of the recursive path ordering. We use the following ordering on the signature; this ordering is taken from [Bergstra and Klop, 1985].

$$+ < \cdot < \mathbin{\|\!\_}_2 < \parallel_2 < \mathbin{\|\!\_}_3 < \parallel_3 < \ldots.$$

Moreover, we give $\cdot$ the lexicographical status for the first argument. We will treat RM1 and RM3 to show the use of the ranked operators. First, we display the calculations that lead to the desired inequality concerning RM1. Let $|x| + |y| = n$. Notice that we are to show that

$$x \parallel_n y >_{lpo} x \mathbin{\|\!\_}_n y + y \mathbin{\|\!\_}_n x.$$

We will make use of the fact that $\parallel_n \; > +$ and that $\parallel_n \; > \mathbin{\|\!\_}_n$.

$$\begin{aligned} x \parallel_n y &>_{lpo} x \parallel_n^* y \\ &>_{lpo} x \parallel_n^* y + x \parallel_n^* y \\ &>_{lpo} (x \parallel_n^* y) \mathbin{\|\!\_}_n (x \parallel_n^* y) + (x \parallel_n^* y) \mathbin{\|\!\_}_n (x \parallel_n^* y) \end{aligned}$$

$$>_{lpo} \quad x \mathbin{⌊\!⌊}_n y + y \mathbin{⌊\!⌊}_n x.$$

Now we handle the case RM3. Let $|x| + |y| = n$.

$$\begin{aligned}
(a \cdot x) \mathbin{⌊\!⌊}_{n+1} y \quad & >_{lpo} \quad (a \cdot x) \mathbin{⌊\!⌊}^*_{n+1} y \\
& >_{lpo} \quad \big((a \cdot x) \mathbin{⌊\!⌊}^*_{n+1} y\big) \cdot \big((a \cdot x) \mathbin{⌊\!⌊}^*_{n+1} y\big) \\
& >_{lpo} \quad (a \cdot x) \cdot \Big(\big((a \cdot x) \mathbin{⌊\!⌊}^*_{n+1} y\big) \parallel_n \big((a \cdot x) \mathbin{⌊\!⌊}^*_{n+1} y\big)\Big) \\
& >_{lpo} \quad (a \cdot^* x) \cdot \big((a \cdot x) \parallel_n y\big) \\
& >_{lpo} \quad a \cdot \big((a \cdot^* x) \parallel_n y\big) \\
& >_{lpo} \quad a \cdot (x \parallel_n y).
\end{aligned}$$

The other cases are verified along the same lines. ∎

By means of the termination of PA we can now formulate the following elimination theorem, which states that the merge and the left merge can be eliminated for closed terms.

Theorem 3.2.4. *For every closed* PA *term t there is a basic* BPA *term s such that* $\mathrm{PA} \vdash t = s$.

Proof. According to theorem 3.2.3 we find that the term rewriting system of table 43 is strongly normalizing. Let t be a closed PA term and let s be its normal form with respect to the term rewriting system of table 43. With proposition 2.2.6 it suffices to show that s is a closed BPA term. Suppose that s contains a merge; then we can use RM1, which contradicts the normality of s. Now suppose that s contains a left merge and consider the smallest subterm containing it. Due to this minimality, it is of the form $u \mathbin{⌊\!⌊} v$ with u and v closed BPA terms. Rewrite u into its BPA normal form. Then RM2, RM3, or RM4 can be applied, which again contradicts the normality of s. So s must be a closed BPA term. ∎

Standard concurrency Some properties concerning both merges cannot be derived from PA, but can only be proved for closed PA terms, for instance the associativity of the merge. In many applications these properties are useful and thus assumed to hold. Hence, following [Bergstra and Tucker, 1984], they are often referred to as *axioms for standard concurrency.* In the next theorem we will treat two such equalities.

Theorem 3.2.5. *Let x, y, and z be closed* PA *terms. Then the following two statements hold:*

(i) $\quad (x \mathbin{⌊\!⌊} y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} (y \parallel z)$,

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

Proof. We prove both equalities with induction on the sum s of the number of symbols occurring in x, y, and z. The case $s = 3$ is trivial so we only treat the case $s+1$. In accordance with theorem 3.2.4, we may assume that x is a basic BPA term. This gives three trivial cases for the first equality.

To prove the second equality use the fact that the first equality holds for $s+1$ and use the (derivable) fact that the merge is commutative. ∎

Expansion An important result in PA with standard concurrency is the so-called expansion theorem, which is a generalization of axiom M1 (see [Bergstra and Tucker, 1984]). It tells us how the merge of more than two processes can be evaluated. For instance, the merge of three processes x, y, and z yields

$$x \mathbin{⌊\!⌊}(y \parallel z) + y \mathbin{⌊\!⌊}(x \parallel z) + z \mathbin{⌊\!⌊}(x \parallel y).$$

Theorem 3.2.6. *In* PA *with standard concurrency we have the following for all open* PA *terms* $x_1, x_2, \ldots, x_n$ *and* $n \geq 2$.

$$x_1 \parallel x_2 \parallel \cdots \parallel x_n = \sum_{i=1}^{n} x_i \mathbin{⌊\!⌊} (x_1 \parallel \cdots \parallel x_{i-1} \parallel x_{i+1} \parallel \cdots \parallel x_n).$$

Proof. Straightforward induction on n. ∎

3.2.2 Semantics of PA

We will give the semantics of PA by means of a term deduction system $T(\mathrm{PA})$. Take for its signature the one of PA and for its rules the ones of BPA in table 5 and the rules concerning the merge in table 44. Bisimulation equivalence is a congruence; see 2.2.32. So the operators of PA can be defined on the quotient of closed PA terms with respect to bisimulation equivalence. The following theorem says that this quotient is a model of PA.

Theorem 3.2.7. *The set of closed* PA *terms modulo bisimulation equivalence is a model of* PA.

Table 44. Derivation rules of $T(\mathrm{PA})$.

$\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$	$\dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$
$\dfrac{x \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} y}$	$\dfrac{y \xrightarrow{a} \surd}{x \parallel y \xrightarrow{a} x}$
$\dfrac{x \xrightarrow{a} x'}{x \mathbin{⌊\!⌊} y \xrightarrow{a} x' \parallel y}$	$\dfrac{x \xrightarrow{a} \surd}{x \mathbin{⌊\!⌊} y \xrightarrow{a} y}$

Proof. A1–A5 are treated as in 2.2.35. M2–M4 are proved as A1. Take for M1 the relation between the left- and right-hand sides of M1, that relates all pairs $(x' \parallel y', y' \parallel x')$, and that contains the diagonal. ■

Next, we take care of the conservativity of PA over BPA.

Theorem 3.2.8. *The equational specification* PA *is a conservative extension of the equational specification* BPA.

Proof. With the aid of theorem 2.4.15 it is very easy to see that the term deduction system $T(\mathrm{PA})$ is an operationally conservative extension of the term deduction system $T(\mathrm{BPA})$ (listed in table 5). With theorem 2.4.19 we also find that this holds up to strong bisimulation equivalence. With the above theorem we know that PA is sound with respect to the model induced by $T(\mathrm{PA})$, so according to theorem 2.4.24 we find the conservativity. ■

Below we give the completeness theorem for PA.

Theorem 3.2.9. *The axiom system* PA *is a complete axiomatization of the set of closed* PA *terms modulo bisimulation equivalence.*

Proof. With the aid of theorem 2.4.26 and the conservativity of PA over BPA we find the completeness of PA (use also theorems 2.2.37 and 3.2.4). ■

3.3 Extensions of PA

In this subsection we will discuss extensions of PA with various features. We already met these extensions when we discussed BPA. We treat the extension of PA with recursion, projections, renaming, the state operator, and iteration. We postpone the extensions of PA with the priority operator and discrete time until we extended the theory PA with deadlock. We deal with PA_δ in subsection 3.3.2. In subsection 3.3.3, we present an application of PA_δ with the state operator, namely in the description of asynchronous communication. We explain in 3.3.4 why we do not treat PA with the empty process.

An extension that is new here is the extension of PA with a process creation mechanism. The reason we did not discuss this extension before, is that this extension makes essential use of the parallel operator. We discuss this extension in subsection 3.3.1.

Recursion Here we will add recursion to PA. This is done in the same way as we did for BPA.

The equational specification PArec has as its signature the signature of BPArec plus the two binary operators $\parallel$ and $⌊\!⌊$ present in the signature of PA. The axioms of PArec are the ones of BPArec plus the axioms of table 42.

The definition of a guard and a guarded recursive specification are the same as in subsection 2.3. Note that there are more guarded terms and recursion equations (thus guarded recursive specifications) in PA than in BPA. For example, $a(X \parallel Y)$ is a guarded term and the recursion equation $X = a \mathbin{\|\!\_} X$ is guarded because of axiom M2.

The semantics of PArec can be given with a term deduction system $T(\mathrm{PArec})$: take for its signature the one of PArec and for its rules the rules of PA plus the rules concerning recursion; see table 6. Since bisimulation equivalence is a congruence (2.2.32), we can define the operators of PArec on the quotient algebra of the set of closed PArec terms with respect to bisimulation equivalence. This quotient is a model of PArec and it satisfies RDP, AIP^{-}, and RSP.

Projection We can extend the equational specification PA with projections as we did for BPA. The equational specification PA+PR has as its signature the one of BPA + PR plus the two binary operators $\parallel$ and $\mathbin{\|\!\_}$ present in the signature of PA. Its axioms are the ones of PA plus the axioms concerning projections; see table 7.

The results that we obtained in subsection 2.4 translate effortlessly to the present situation.

Recursion and projection Here we will extend PA with both recursion and projection. This extension is obtained analogously to the extension of BPA with recursion and projection. The specification PArec + PR has as its signature the one of PArec plus the unary operators π_n that occur in the signature of PA + PR. Its axioms are the ones of PArec plus the axioms concerning projection; see table 7. The results that we obtained for BPArec + PR in subsection 2.4.2 also hold for PArec + PR. We will not mention them here.

The semantics of PArec + PR can be given by a combination of the term deduction system of PArec and PA + PR.

Renaming It is not difficult to extend the equational specification PA, and its extensions, with renaming operators; see subsections 2.7 and 2.7.1.

State operator We can extend the theory PA with either the simple or extended state operator in the same way as we did for the theory BPA. For details we refer to subsections 2.8 and 2.9.

Iteration We can extend PA with iteration by just adding the defining axioms for * in table 37 to the ones for PA. For this theory there is no completeness result present at the time of writing this survey.

3.3.1 Process creation

In this subsection, we discuss an extension of PA with an operator that models process creation. This extension is not present in BPA since it is defined using the parallel composition $\parallel$. This subsection is based on [Bergstra, 1990]. We refer to [America and Bakker, 1988] and to [Baeten and Vaandrager, 1992] for other approaches to process creation.

We refer to [Bergstra and Klint, 1994] for an application of process

Table 45. Axioms for the process creation operator.

$E_\phi(a) = a,\;$ if $a \neq cr(d)$ for $d \in D$	PCR1
$E_\phi\big(cr(d)\big) = \overline{cr}(d) \cdot E_\phi\big(\phi(d)\big),\;$ for $d \in D$	PCR2
$E_\phi(a \cdot x) = a \cdot E_\phi(x),\;$ if $a \neq cr(d)$ for $d \in D$	PCR3
$E_\phi\big(cr(d) \cdot x\big) = \overline{cr}(d) \cdot E_\phi\big(\phi(d) \parallel x\big),\;$ for $d \in D$	PCR4
$E_\phi(x + y) = E_\phi(x) + E_\phi(y)$	PCR5

creation.

The theory The equational specification PA + PCR (process algebra with process creation) is defined in stages. First we take the theory PA where we assume that the set of atomic actions A contains some special actions: for all d in some data set D we assume that $cr(d) \in A$ and $\overline{cr}(d) \in A$. We moreover assume the existence of a function, the process creation function, ϕ on D that assigns to each $d \in D$ a process term $\phi(d)$ over PA with the above set of atomic actions A. Using the function ϕ we add a unary operator E_ϕ to the signature, thus obtaining the signature of PA + PCR. The operator E_ϕ is called the process creation operator.

The equations of PA+PCR are those of PA plus the ones that define E_ϕ. We list these equations in table 45.

Intuition We provide some intuition for PA+PCR. We will compare process creation with the UNIX[2] system call `fork`; see [Ritchie and Thompson, 1974]. We recall that with `fork` we can only create an exact copy (child process) of its so-called parent process. We note that with the process creation operator we are able to create arbitrary processes but to provide an intuition for process creation the system call `fork` is illustrative.

The atomic action $cr(d)$ can be seen as a trigger for E_ϕ; compare $cr(d)$ to the system call `fork`. The operator E_ϕ initiates the creation of a process when a $cr(d)$ is parsed; think of it as a program that invokes the system call `fork`. The action $\overline{cr}(d)$ indicates that a process creation has occurred; this action can be interpreted as the return value of the system call `fork` to the parent process (which is the unique process `ID` of the newly created process). Maybe this intuition is best illustrated by axiom PCR4. There we see that from $E_\phi\big(cr(d) \cdot x\big)$ a process $\phi(d)$ is created that is put in parallel with the remaining process x, while leaving a trace $\overline{cr}(d)$.

Next, we formulate a simple lemma that states that process creation distributes over the merge.

Lemma 3.3.1. *For all closed* PA *terms x and y we have*

[2]UNIX is a registered trademark of UNIX System Laboratories (at least at the time of writing this survey).

Table 46. Operational rules for the process creation operator.

$\dfrac{x \xrightarrow{a} x'}{E_\phi(x) \xrightarrow{a} E_\phi(x')}, a \neq cr(d)$	$\dfrac{x \xrightarrow{a} \surd}{E_\phi(x) \xrightarrow{a} \surd}, a \neq cr(d)$
$\dfrac{x \xrightarrow{cr(d)} x'}{E_\phi(x) \xrightarrow{\overline{cr}(d)} E_\phi\big(\phi(d) \parallel x'\big)}, d \in D$	$\dfrac{x \xrightarrow{cr(d)} \surd}{E_\phi(x) \xrightarrow{\overline{cr}(d)} E_\phi\big(\phi(d)\big)}, d \in D$

$$E_\phi(x \parallel y) = E_\phi(x) \parallel E_\phi(y).$$

Proof. Use structural induction on both x and y. ∎

Example 3.3.2. Let $D = \{d\}$ and let $\phi(d) = cr(d)$. If $x = E_\phi\big(cr(d)\big)$, then $x = \overline{cr}(d) \cdot x$. So we see that even the simplest examples give rise to recursive equations.

Termination The above example shows that the term rewriting system associated to PA + PCR (by orienting the axioms of PA + PCR from left to right) is, in general, not terminating. For, in case of the above example, we have the following infinite sequence of rewritings:

$$E_\phi\big(cr(d)\big) \to \overline{cr}(d) \cdot E_\phi\big(cr(d)\big) \to \overline{cr}(d) \cdot \overline{cr}(d) \cdot E_\phi\big(cr(d)\big) \to \ldots.$$

Semantics We discuss the operational semantics of PA + PCR. It is obtained by means of a term deduction system. The signature is that of PA + PCR; the operational rules are those of PA plus the rules that operationally define the process creation operator E_ϕ. We list them in table 46. The rules of this table originate from [Baeten and Bergstra, 1988b].

The soundness of PA + PCR is easily established.

Theorem 3.3.3. *The set of closed* PA + PCR *terms modulo bisimulation equivalence is a model of* PA + PCR.

Proof. For the equations of PA we refer to theorem 3.2.7. So we only need to show the soundness of the equations PCR1–5. This is easy. We only give the bisimulation relations and leave the calculations to the reader. For PCR1, relate the left-hand side and the right-hand side of PCR1. For PCR2–5 also take such a pair and join this with the diagonal. ∎

Next, we state that PA + PCR is a conservative extension of PA.

Theorem 3.3.4. *The equational specification* PA + PCR *is a conservative extension of the equational specification* PA.

Proof. As usual. ∎

From example 3.3.2 it follows that the process creation operator introduces recursion. So we do not have a completeness theorem.

3.3.2 Deadlock in PA

It is straightforward to add deadlock to the theory PA. The equational specification PA_δ has as its signature the one of PA plus a constant $\delta \notin A$. Its axioms are the ones of PA plus the axioms concerning deadlock listed in table 10. We assume for axioms M2 and M3 that a ranges over the set $A \cup \{\delta\}$.

Structural induction We can use structural induction for PA_δ as before for BPA_δ, since basic PA_δ terms are just basic BPA_δ terms. This follows immediately from the fact that both merges can be eliminated from closed PA_δ terms. This can be shown by means of a term rewriting analysis just as in theorem 3.2.3 and the following elimination theorem.

Theorem 3.3.5. *For every closed* PA_δ *term* t *there is a basic* BPA_δ *term* s *such that* $\mathrm{PA}_\delta \vdash t = s$.

Proof. This is proved along the same lines as theorem 3.2.4. ∎

Also the conservativity of PA_δ over BPA_δ and the completeness of PA_δ can be proved along the same lines as these results for PA without extensions.

Standard concurrency Standard concurrency in PA_δ is dealt with completely analogously to the situation without deadlock, so we refer to theorem 3.2.5 for standard concurrency. Below, we will mention some properties about the connection of deadlock and parallel composition. The proof of these properties is elementary and therefore omitted.

Theorem 3.3.6.

(i) $\mathrm{PA}_\delta \vdash \delta \mathbin{⌊\!⌊} x = \delta$.

Let x *be a closed* PA_δ *term. Then we have*

(ii) $x \parallel \delta = x \mathbin{⌊\!⌊} \delta = x\delta$.

Let x *and* y *be closed* PA_δ *terms. Then we have*

(iii) $x \parallel y\delta = (x \parallel y)\delta = x\delta \parallel y$.

Remark 3.3.7. We mention that if in addition we have standard concurrency, the proof of (iii) follows easily using (ii):

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

Expansion For PA_δ with standard concurrency we have the same expansion theorem as for the theory without deadlock, so for expansion we refer to theorem 3.2.6.

Semantics The semantics of PA_δ can be given by means of a term deduction system $T(\mathrm{PA}_\delta)$, which is just $T(\mathrm{PA})$ with δ added to its signature. The operators of PA_δ can be easily defined by taking representatives on the quotient of the set of closed PA_δ terms modulo bisimulation equivalence, since this relation is a congruence; see 2.2.32. The quotient is a model for PA_δ, which can be easily checked by combining the soundness proofs for BPA_δ and PA. Moreover, the axiom system PA_δ is a complete axiomatization of this quotient. This follows immediately from the completeness of PA since we did not introduce any new transitions.

3.3.3 Asynchronous communication

It is straightforward to extend PA_δ with any of the features mentioned in the beginning of subsection 3.3. Here, we consider an application of PA_δ with the (simple) state operator. We describe mail through a communication channel. Let D be a finite data set and let c be a communication channel. We assume that for each $d \in D$ we have the following special atomic actions:

- $c \uparrow d$ send d via c; *potential* action
- $c \Uparrow d$ send d via c; *realized* action
- $c \downarrow d$ receive d via c; *potential* action
- $c \Downarrow d$ receive d via c; *realized* action.

The state operator will turn potential, intended actions into realized actions. The state space will keep track of outstanding messages.

We consider the case where the communication channel behaves like a queue, i.e. the order of the messages is preserved. Without much trouble, descriptions for other kinds of channels can be generated (for instance, a bag-like channel). Thus, the state space is D^*, the set of words over D. Let σ, ρ range over D^*, and let ε denote the empty word. We denote the concatenation of words σ and ρ simply by $\sigma\rho$. Note that $D \subseteq D^*$ so σd is the concatenation of the words σ and d (for $d \in D$). Let $last(\sigma)$ be the last element of word σ, if $\sigma \neq \varepsilon$.

We define the *action* and *effect* functions implicitly, by giving the relevant instances of axiom SO2.

$$\begin{array}{rcll}
\lambda^c_\sigma(c \uparrow d \cdot x) & = & c \Uparrow d \cdot \lambda^c_{d\sigma}(x) & \\
\lambda^c_{\sigma d}(c \downarrow d \cdot x) & = & c \Downarrow d \cdot \lambda^c_\sigma(x) & \\
\lambda^c_\sigma(c \downarrow d \cdot x) & = & \delta, & \text{if } \sigma = \varepsilon \text{ or } last(\sigma) \neq d.
\end{array}$$

The *action* and *effect* functions are inert for all other atomic actions. Now suppose $0 \in D$; then we can describe two communication partners:

$$S = c \uparrow 0,$$

$$R = \sum_{d \in D} c \downarrow d \cdot print(d).$$

Some easy calculations show that

$$\begin{aligned} \lambda_\varepsilon^c(S \parallel R) &= c \Uparrow 0 \cdot \lambda_0^c(R) \ = \ c \Uparrow 0 \cdot c \Downarrow 0 \cdot \lambda_\varepsilon^c\big(print(0)\big) \\ &= c \Uparrow 0 \cdot c \Downarrow 0 \cdot print(0). \end{aligned}$$

Asynchronous communication in the setting of PA was introduced in [Bergstra *et al.*, 1985]. The present formulation is taken from [Baeten and Weijland, 1990].

3.3.4 Empty process in PA

We will not discuss the combination of parallel composition and the empty process, since this combination is not (yet) standardized. At this moment there are two possible ways to combine the merge and the empty process. These options originate from the various interpretations of the term $\varepsilon \mathbin{\|\!\!\lfloor} x$. It may seem natural to demand that this equals x, since ε is only capable of terminating successfully, but this perspective leads to a non-associative merge, which is rather unnatural and therefore unwanted [Vrancken, 1986]. The intended interpretation of the left merge is that of the merge with the first action from the left process, so the term $\varepsilon \mathbin{\|\!\!\lfloor} x$ cannot proceed, since ε cannot perform an action. One of the options is that $\varepsilon \mathbin{\|\!\!\lfloor} x$ equals δ except if $x = \varepsilon$: then it equals ε; see [Vrancken, 1986] for more information. The other option drops this exception and uses a unary operator indicating whether or not a process has a termination option to axiomatize the merge [Baeten and Glabbeek, 1987].

3.4 Extensions of PA_δ

In this subsection we will discuss the extensions of PA_δ with recursion, projections, renaming, and/or the encapsulation operator, the state operator, the priority operator, iteration, process creation, and discrete time.

Recursion and/or projection The extensions of PA_δ with recursion, projection, or a combination of both are obtained by simply merging these extensions for BPA_δ and PA; see subsections 2.5 and 3.3.

Renaming and encapsulation It is straightforward to extend the equational specification PA_δ, and its extensions, with renaming operators or the encapsulation operator; cf. subsection 2.7.3.

State operator The extension of the theory PA_δ with either the simple or extended state operator is obtained in the same way as for the theory PA; see subsections 2.8 and 2.9.

Priority operator We can extend the theory PA_δ with the priority operator in the same way as the extension of BPA_δ with that operator. For details

Table 47. The interaction between the left merge and the discrete time unit delay.

$\sigma_d(x) \mathbin{\lfloor\!\lfloor} \underline{\underline{\delta}} = \underline{\underline{\delta}}$	DTM1
$\sigma_d(x) \mathbin{\lfloor\!\lfloor} (\underline{\underline{a}} + y) = \sigma_d(x) \mathbin{\lfloor\!\lfloor} y$	DTM2
$\sigma_d(x) \mathbin{\lfloor\!\lfloor} (\underline{\underline{a}} \cdot y + z) = \sigma_d(x) \mathbin{\lfloor\!\lfloor} z$	DTM3
$\sigma_d(x) \mathbin{\lfloor\!\lfloor} \sigma_d(y) = \sigma_d(x \mathbin{\lfloor\!\lfloor} y)$	DTM4

of that extension we refer to section 2.10.

Iteration We can extend PA_δ with iteration by just adding the defining axioms for * in table 37 to the ones for PA_δ. Only for BPA^* is the completeness proved at the time of writing this survey.

Process creation We can extend PA_δ with the process creation operator E_ϕ in the same way as we did for PA. For details we refer to subsection 3.3.1.

3.4.1 Discrete time

In this subsection, we extend PA_δ with discrete time. With the interaction between the discrete time unit delay σ_d and the left merge $\mathbin{\lfloor\!\lfloor}$ we have to be a bit careful. We recall that $x \mathbin{\lfloor\!\lfloor} y$ is x and y in parallel but the first action stems from x. With discrete time present, the question arises if that is possible at all. For instance, $\sigma_d(\underline{\underline{a}}) \mathbin{\lfloor\!\lfloor} \underline{\underline{b}}$ equals $\underline{\underline{\delta}}$ in this system as we cannot move to the next time slice in order to let a happen, since b must occur in the current time slice. The material of this subsection is based on [Baeten and Bergstra, 1992a].

The theory We discuss the equational specification $\mathrm{PA}_{\delta\mathrm{dt}}$. Its signature is the one of PA_δ (with $\underline{\underline{a}}$ instead of a for $a \in A_\delta$) plus the discrete time unit delay operator σ_d that we first introduced in $\mathrm{BPA}_{\mathrm{dt}}$. The equations of $\mathrm{PA}_{\delta\mathrm{dt}}$ are the ones of $\mathrm{BPA}_{\delta\mathrm{dt}}$ plus the equations for the merge that we listed in table 42 (again with $\underline{\underline{a}}$ instead of a) and the equations that represent the interaction between the left merge and the discrete time unit delay; we list the latter axioms in table 47. Incidentally, this axiomatization is new here.

Termination Next, we discuss the termination of a term rewriting system associated to the equational specification $\mathrm{PA}_{\delta\mathrm{dt}}$. Since we have the left merge in our signature we use the ranked operators that we also used for the termination of PA; cf. subsection 3.2.1.

Theorem 3.4.1. *The term rewriting system associated with* $\mathrm{PA}_{\delta\mathrm{dt}}$ *consisting of the rewrite rules for* PA_δ*, the rules of table 40, and the equations in table 47 oriented from left to right is strongly normalizing.*

Proof. Let us use the theory of subsection 2.2.2. As usual we confine ourselves to giving a partial ordering on the signature. In addition to the ordering that we gave in the termination proof for PA_δ

$$+ < \cdot < \mathbin{\lfloor\!\lfloor}_2 < \|_2 < \mathbin{\lfloor\!\lfloor}_3 < \|_3 < \ldots,$$

we have the following precedence:

$$\sigma_d < +.$$

The rest of the proof consists of straightforward calculations. ∎

Elimination With the above theorem we can obtain an elimination result for closed terms. However, we cannot obtain this result directly. This is due to the fact that we did not consider a term rewriting analysis modulo the axioms without a clear direction such as A1 and A2. We make the problems a bit more concrete with the following term rewriting *modulo* A1 and A2.

$$\begin{aligned}
\sigma_d(\underline{\underline{a}}) \mathbin{\lfloor\!\lfloor} \big((\sigma_d(\underline{\underline{b}}) + \underline{\underline{a}}) + \sigma_d(\underline{\underline{c}})\big) &= \sigma_d(\underline{\underline{a}}) \mathbin{\lfloor\!\lfloor} \big(\underline{\underline{a}} + (\sigma_d(\underline{\underline{b}}) + \sigma_d(\underline{\underline{c}}))\big) \\
&\twoheadrightarrow \sigma_d(\underline{\underline{a}}) \mathbin{\lfloor\!\lfloor} \sigma_d(\underline{\underline{b}} + \underline{\underline{c}}) \\
&\rightarrow \sigma_d\big(\underline{\underline{a}} \mathbin{\lfloor\!\lfloor} (\underline{\underline{b}} + \underline{\underline{c}})\big) \\
&\rightarrow \sigma_d\big(\underline{\underline{a}} \cdot (\underline{\underline{b}} + \underline{\underline{c}})\big).
\end{aligned}$$

So, we see that for the elimination of the left merge we need more than just the termination result above. We will solve this problem in the next theorem.

Theorem 3.4.2. *The equational specification* $\mathrm{PA}_{\delta\mathrm{dt}}$ *has the elimination property for* $\mathrm{BPA}_{\delta\mathrm{dt}}$.

Proof. Let t be a $\mathrm{PA}_{\delta\mathrm{dt}}$ term. Rewrite t with the term rewriting system associated with $\mathrm{PA}_{\delta\mathrm{dt}}$ to a normal form t_0. It is possible that left merges still occur in the resulting term. Take the minimal subterm of t_0 that contains a left merge $s \mathbin{\lfloor\!\lfloor} s_1$. Both s and s_1 are $\mathrm{BPA}_{\delta\mathrm{dt}}$ terms. We may assume that s is of the form $\sigma_d(s_0)$ (otherwise t_0 would not be in normal form). With the aid of theorem 2.12.3 we know that s_1 can be written in one of the following forms:

- $\sum\limits_{i<n} \underline{\underline{a}}_i \cdot t_i + \sum\limits_{j<m} \underline{\underline{b}}_j$,
- $t + \sigma_d(s)$ with t of the above form.

Now replace s_1 with one of the above forms and rewrite the resulting new term to a normal form and repeat this procedure until all left merges have been eliminated. ∎

Semantics Now we discuss the operational semantics of $\mathrm{PA}_{\delta\mathrm{dt}}$. The semantics of the system $\mathrm{PA}_{\delta\mathrm{dt}}$ is quite straightforward. In table 48 we list the

Table 48. The additional rules for the merge and the left merge.

$\dfrac{x \xrightarrow{\sigma} x',\, y \xrightarrow{\sigma} y'}{x \parallel y \xrightarrow{\sigma} x' \parallel y'}$	$\dfrac{x \xrightarrow{\sigma} x',\, y \xrightarrow{\sigma} y'}{x \mathbin{\lfloor\!\lfloor} y \xrightarrow{\sigma} x' \mathbin{\lfloor\!\lfloor} y'}$

additional operational rules for the merge and the left merge. The entire semantics of $\mathrm{PA}_{\delta\mathrm{dt}}$ consists of the one of $\mathrm{BPA}_{\delta\mathrm{dt}}$ plus the rules in tables 44 and 48.

Theorem 3.4.3. *The set of closed* $\mathrm{PA}_{\delta\mathrm{dt}}$ *terms modulo bisimulation equivalence is a model of* $\mathrm{PA}_{\delta\mathrm{dt}}$.

Proof. Since bisimulation equivalence is a congruence, we only need to check the soundness of the axioms of $\mathrm{PA}_{\delta\mathrm{dt}}$. We already treated all the axioms except DTM1–4. For these we give the bisimulation relations. For DTM1 relate the left-hand side and the right-hand side. For DTM2–4 also relate the left- and right-hand sides and add the diagonal. ∎

Theorem 3.4.4. *The equational specification* $\mathrm{PA}_{\delta\mathrm{dt}}$ *is a conservative extension of the equational specification* $\mathrm{BPA}_{\delta\mathrm{dt}}$.

Proof. Easy. ∎

Now we have all the prerequisites to state the completeness theorem for $\mathrm{PA}_{\delta\mathrm{dt}}$. The proof is as usual and therefore omitted.

Theorem 3.4.5. *The axiom system* $\mathrm{PA}_{\delta\mathrm{dt}}$ *is a complete axiomatization of the set of closed* $\mathrm{PA}_{\delta\mathrm{dt}}$ *terms modulo bisimulation equivalence.*

3.5 Syntax and semantics of communicating processes

In this subsection we will extend the meaning of the parallel operator $\parallel$ that we introduced in subsection 3.2. We will call the ensuing operator $\parallel$ the merge or parallel composition. We use the name free merge for the merge without communication, that is the merge of PA.

We use the extended merge to model synchronous communication between processes.

3.5.1 The theory ACP

We define the syntax of the equational specification $\mathrm{ACP} = (\Sigma_{\mathrm{ACP}}, E_{\mathrm{ACP}})$ of [Bergstra and Klop, 1984b].

The signature Σ_{ACP} consists of the one of PA_δ plus a binary operator $|$, called the communication merge and the encapsulation operator ∂_H that we already discussed in subsection 2.7.3 (we recall that $H \subseteq A$). Moreover, we fix a partial function $\gamma : A \times A \longrightarrow A$, where A is the set of atomic actions. We call γ the communication function. The communication function is, like A, a parameter of the theory. It is meant to model the communication

Table 49. Axioms for the merge with communication.

$a \mid b = \gamma(a,b)$, if $\gamma(a,b)$ defined;	CF1
$a \mid b = \delta$ otherwise.	CF2
$x \parallel y = x \mathbin{\|\!\!\lfloor} y + y \mathbin{\|\!\!\lfloor} x + x \mid y$	CM1
$a \mathbin{\|\!\!\lfloor} x = ax$	CM2
$ax \mathbin{\|\!\!\lfloor} y = a(x \parallel y)$	CM3
$(x+y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} z + y \mathbin{\|\!\!\lfloor} z$	CM4
$(a \cdot x) \mid b = (a \mid b) \cdot x$	CM5
$a \mid (b \cdot x) = (a \mid b) \cdot x$	CM6
$(a \cdot x) \mid (b \cdot y) = (a \mid b) \cdot (x \parallel y)$	CM7
$(x+y) \mid z = x \mid z + y \mid z$	CM8
$x \mid (y+z) = x \mid y + x \mid z$	CM9

between processes. In fact, the communication merge $\mid$ is the extension of the communication function to processes. We require that γ is both commutative and associative; that is, if $\gamma(a,b)$ is defined, it equals $\gamma(b,a)$ and if $\gamma\big(a,\gamma(b,c)\big)$ is defined it equals $\gamma\big(\gamma(a,b),c\big)$ and vice versa. So, we can leave out the brackets in such formulas and render the latter expression as $\gamma(a,b,c)$.

Now we give the set of equations Σ_{ACP}. This set consists of the equations for $\mathrm{BPA}_\delta + \partial_H$ that we discussed in subsection 2.7.3 and the equations that we list in table 49. See table 21 for the defining axioms of the encapsulation operator. Observe that the equations CM2–4 are the same as the ones that we discussed when we introduced PA. We recall them for the sake of ease.

Now we discuss the axioms of ACP. The most important one is CM1 where a third possibility for the merge is added. The intended interpretation of this summand $x \mid y$ is that it is the parallel composition of the two processes x and y but that the first step must be a communication. Both processes must be able to perform an action for which γ is defined.

Terminology We say that two atomic actions do not communicate if the communication function is not defined for them. We say that an atomic action a is a communication action if $a = \gamma(b,c)$ for atomic actions b and c. A communication action $\gamma(b,c)$ is called a binary communication; likewise $\gamma(a,b,c)$ is called ternary if defined. However, most of the time just using binary communication is enough in applications. See also later on when we discuss so-called handshaking.

Read/send communication An important case of binary communication is called read/send communication. The idea is that in the set of atomic actions we have read actions $r_i(d)$, send actions $s_i(d)$, and communication actions $c_i(d)$. The intended meaning of $r_i(d)$ is to read datum $d \in D$ at port i, where the set D is some finite data set. For $s_i(d)$ a similar intuition holds. Now $c_i(d)$ is the result of a communication of $r_i(d)$ and $s_i(d)$: it means transmit the datum d by communication at port i. The appropriate communication function is $\gamma\big(r_i(d), s_i(d)\big) = \gamma\big(s_i(d), r_i(d)\big) = c_i(d)$ and γ is not defined otherwise on $r_i(d)$, $s_j(e)$, and $c_k(f)$ (for ports i, j, k and data elements d, e, f). For other atomic actions it is permitted to have some communications defined. The above conventions are due to [Bergstra and Klop, 1986]. An example of the use of read/send communication is given in section 3.6.

Structural induction As before we can use structural induction for ACP, since basic ACP terms are just basic BPA$_\delta$ terms. This follows from theorem 3.5.5 that states that parallel composition and encapsulation can be eliminated from closed ACP terms.

3.5.2 Termination

As before we prove the termination of a term rewriting system associated to ACP (see table 50) with the aid of the theory of subsection 2.2.2. The proof of this fact will more or less be the same as the proof for PA. There we have given some operators a weight to avoid problems with the left merge. Note that in table 50, we rewrite $a \,|\, b$ to an atomic action c or δ in order to eliminate the communication merge.

Next, we give the definition of the weight function. It is an extension of definition 3.2.1.

Definition 3.5.1. Let x and y be terms and let a be an atomic action. The weight of a term x, notation $|x|$ is defined inductively as follows.

- $|a| = 1$
- $|x + y| = \max\{|x|, |y|\}$
- $|x \cdot y| = |x| + |y|$
- $|x \parallel y| = |x| + |y|$
- $|x \mathbin{\|\!\_} y| = |x| + |y|$
- $|x \,|\, y| = |x| + |y|$
- $|\partial_H(x)| = |x|$.

Below we give the definition of a ranked operator as defined by [Bergstra and Klop, 1985]. And we list the new signature. This definition is an extension of definition 3.2.2.

Definition 3.5.2. The rank of an operator $\parallel$, $\mathbin{\|\!\_}$, or $|$ is the sum of the weights of its arguments. The signature for the term rewriting system

Table 50. Term rewriting rules for the merge.

$a \mid b \to c$, if $\gamma(a, b) = c$;	RCF1
$a \mid b \to \delta$ otherwise.	RCF2
$x \parallel y \to x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y$	RCM1
$a \mathbin{⌊\!⌊} x \to ax$	RCM2
$ax \mathbin{⌊\!⌊} y \to a(x \parallel y)$	RCM3
$(x + y) \mathbin{⌊\!⌊} z \to x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z$	RCM4
$(a \cdot x) \mid b \to (a \mid b) \cdot x$	RCM5
$a \mid (b \cdot x) \to (a \mid b) \cdot x$	RCM6
$(a \cdot x) \mid (b \cdot y) \to (a \mid b) \cdot (x \parallel y)$	RCM7
$(x + y) \mid z \to x \mid z + y \mid z$	RCM8
$x \mid (y + z) \to x \mid y + x \mid z$	RCM9

associated with ACP is the following:

$$A \cup \{+, \cdot, \partial_H\} \cup \{\parallel_n, ⌊\!⌊_n, \mid_n : n \geq 2\},$$

where the subscripted n stands for the weight of the subterm.

Definition 3.5.3. Let the term rewriting system associated to ACP consist of the following rules: the term rewriting system associated to $\mathrm{BPA}_\delta + \partial_H$ and the rules in table 50. For completeness sake, we note that the term rewriting system associated to $\mathrm{BPA}_\delta + \partial_H$ consists of the rewrite rules of table 2 and the equations in tables 10 and 21 oriented from left to right.

Theorem 3.5.4. *The term rewriting system associated to* ACP *is strongly normalizing.*

Proof. The proof can be given along the same lines as the proof of the termination of the system PA; cf. theorem 3.2.3. We use the partial ordering of the signature in figure 8 and leave the calculations to the reader. ∎

With the aid of the above termination result for ACP we can easily prove the following elimination theorem.

Theorem 3.5.5. *The equational specification* ACP *has the elimination property for* BPA_δ.

Proof. Easy. ∎

Standard concurrency As for PA we have standard concurrency. That is, there are some properties concerning the merge, left merge, and communication merge that are not derivable for arbitrary open terms, but can be

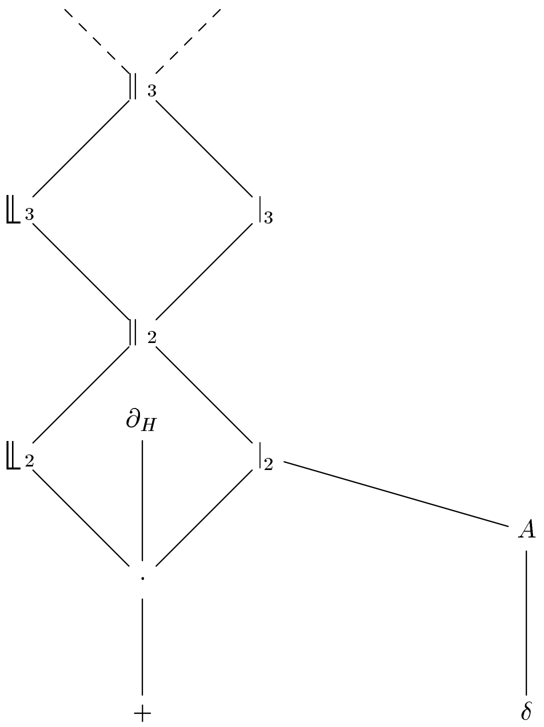

Fig. 8. Partial ordering of the operators in the term rewriting system associated to ACP.

shown to be valid for closed terms (or even for solutions of guarded recursive equations). In many applications these properties are useful and thus these properties are assumed to be valid. This is why these properties are often referred to as axioms for standard concurrency. In the next theorem we list them for ACP. See theorem 3.2.5 for standard concurrency in PA.

Theorem 3.5.6. *Let x, y, and z be closed* ACP *terms. Then the following statements hold. They are called axioms of standard concurrency.*

(i) $x \mid y = y \mid x$,

(ii) $x \parallel y = y \parallel x$,

(iii) $(x \mid y) \mid z = x \mid (y \mid z)$,

(iv) $(x \mathbin{\|\!\!\lfloor} y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} (y \parallel z)$,

(v) $x \mid (y \mathbin{\|\!\!\lfloor} z) = (x \mid y) \mathbin{\|\!\!\lfloor} z$,

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

Table 51. Handshaking axiom.

$x \mid y \mid z = \delta$	HA

Proof. We will not give the details of the proof. They can be found in [Bergstra and Tucker, 1984]. Instead, we explain the proof strategy.

Because of theorem 3.5.5, we only need to prove the properties for basic BPA_δ terms. It is easy to show that the first two properties hold by induction to the sum of symbols that occur in both x and y. Then it remains to show with a simultaneous induction to the number of symbols occurring in x, y, and z that the other properties also hold. ∎

Expansion A useful application of standard concurrency in ACP is the so-called expansion theorem. This theorem states how the merge of more than two processes can be evaluated. For the expansion theorem of PA we refer to theorem 3.2.6. In contrast to the PA expansion theorem, we need an extra proviso for the expansion of the merge in case the communication merge is present. We need a so-called handshaking axiom (HA). We give it in table 51. It states that there is only binary communication present.

Theorem 3.5.7. *Suppose that $\gamma(a,b,c)$ is undefined for all atomic actions a, b, and c. Then for all closed* ACP *terms x, y, and z we have HA.*

Proof. Easy. ∎

Next, we formulate the expansion theorem for ACP. The notation that we use in this theorem can be defined inductively in the obvious way.

Theorem 3.5.8. *In* ACP *with standard concurrency and the handshaking axiom presented in table 51 we have the following for all open* ACP *terms $x_1, x_2, \ldots, x_n$ and $n \geq 2$.*

$$x_1 \parallel x_2 \parallel \cdots \parallel x_n \quad = \quad \sum_{i=1}^{n} x_i \mathbin{\|\!\lfloor} (\mathop{\Big\|}_{\substack{j=1\\ j\neq i}}^{n} x_j) + \sum_{\substack{i,j=1\\ i<j}}^{n} (x_i \mid x_j) \mathbin{\|\!\lfloor} (\mathop{\Big\|}_{\substack{k=1\\ k\neq i,j}}^{n} x_k).$$

Proof. Straightforward induction on n. See [Bergstra and Tucker, 1984] for a detailed proof. ∎

3.5.3 Semantics of ACP

In this subsection we give the semantics of ACP. In fact, this is now an easy job, since almost all the constructs that ACP contains have been discussed before. The only notion that we did not characterize operationally is the communication merge. In table 52 we present the operational rules for this concept.

Table 52. The operational rules for the communication merge.

$$
\begin{array}{ll}
\dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} y'}{x \parallel y \xrightarrow{c} x' \parallel y'}, \gamma(a,b)=c & \dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} y'}{x \mid y \xrightarrow{c} x' \parallel y'}, \gamma(a,b)=c \\
\dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} \surd}{x \parallel y \xrightarrow{c} x'}, \gamma(a,b)=c & \dfrac{x \xrightarrow{a} x',\ y \xrightarrow{b} \surd}{x \mid y \xrightarrow{c} x'}, \gamma(a,b)=c \\
\dfrac{x \xrightarrow{a} \surd,\ y \xrightarrow{b} y'}{x \parallel y \xrightarrow{c} y'}, \gamma(a,b)=c & \dfrac{x \xrightarrow{a} \surd,\ y \xrightarrow{b} y'}{x \mid y \xrightarrow{c} y'}, \gamma(a,b)=c \\
\dfrac{x \xrightarrow{a} \surd,\ y \xrightarrow{b} \surd}{x \parallel y \xrightarrow{c} \surd}, \gamma(a,b)=c & \dfrac{x \xrightarrow{a} \surd,\ y \xrightarrow{b} \surd}{x \mid y \xrightarrow{c} \surd}, \gamma(a,b)=c
\end{array}
$$

The complete operational semantics for ACP is given by a term deduction system $T(\mathrm{ACP})$ that has as its signature the one of ACP and the rules are the ones of table 5 (the BPA part), the rules of table 22 (the encapsulation part), the rules of table 44 (the PA merge part), and the new rules that we list in table 52. As usual, bisimulation equivalence is a congruence; see 2.2.32. So the operators of ACP can be defined on the quotient of closed ACP terms with respect to bisimulation equivalence. The following theorem says that this quotient is a model of ACP.

Theorem 3.5.9. *The set of closed* ACP *terms modulo bisimulation equivalence is a model of* ACP.

Proof. We have already treated the equations that comprise BPA_δ; see theorem 2.5.4. We have also seen the soundness of BPA + RN, which is BPA with renaming operators. Since the encapsulation operator is a special case of a renaming operator, the soundness of the equations D1–4 can be proved in the same way as the soundness proof of BPA + RN; see theorem 2.7.4. We have also seen the soundness of the equations CM2–4, since these axioms are the same in PA; see theorem 3.2.7. So it remains to prove that the other equations are sound. We confine ourselves to only giving the bisimulation relations. We begin with CF1. Take the relation that relates both sides of CF1. Equation CF2 is treated exactly the same. Now we treat CM1. Take for CM1 the relation that relates both sides of CM1, that relates $x' \parallel y'$ and $y' \parallel x'$ for all closed ACP terms x' and y', and that contains the diagonal. We recall that CM2–4 are treated the same as M2–4 of PA. So we continue with CM5–9. These are proved analogously to A1; that is, relate both sides of an equation and add the diagonal. This ends the soundness proof for ACP. ∎

At this point, we are able to prove the conservativity of ACP over BPA_δ.

Theorem 3.5.10. *The equational specification* ACP *is a conservative extension of the equational specification* BPA_δ.

Proof. With the aid of theorem 2.4.15 it is very easy to see that the term deduction system $T(\mathrm{ACP})$ is an operationally conservative extension of the term deduction system $T(\mathrm{BPA}_\delta)$ (for the definition of $T(\mathrm{BPA}_\delta)$ we refer to the soundness theorem 2.5.4 for BPA_δ). With theorem 2.4.19 we also find that this holds up to strong bisimulation equivalence. With theorem 3.5.9 we know that ACP is sound with respect to the model induced by $T(\mathrm{ACP})$, so according to theorem 2.4.24 we immediately find the equational conservativity. ∎

Below we give the completeness theorem for ACP.

Theorem 3.5.11. *The axiom system* ACP *is a complete axiomatization of the set of closed* ACP *terms modulo bisimulation equivalence.*

Proof. With the aid of theorem 2.4.26 and the conservativity of ACP over BPA_δ we find the completeness of ACP (use also theorems 2.5.5 and 3.5.5). ∎

3.6 Extensions of ACP

In this subsection we will discuss extensions of ACP with the features that we already met when we discussed extensions of both BPA and PA. We treat the extension of ACP with recursion, projections, renaming operators, the state operator, the priority operator, iteration, process creation, and discrete time. We will also treat two examples to illustrate the use of two of the extensions. We do not discuss the extension of ACP with the empty process. We explained why in subsection 3.3.4.

Recursion and/or projection The extensions of ACP with recursion, projection, or a combination of both are obtained by simply merging these extensions for BPA_δ and ACP; see subsection 2.5.

Example 3.6.1. In ACPrec, we can describe communicating buffers using the read/send communication function defined in subsection 3.5.1. Let D be a finite data set. A *one-place buffer* over D, with input port 1 and output port 2, is given by the recursive equation

$$B \quad = \quad \sum_{d \in D} r_1(d) \cdot s_2(d) \cdot B.$$

Likewise, a one-place buffer with input port 2 and output port 3 is given by

$$C \quad = \quad \sum_{d \in D} r_2(d) \cdot s_3(d) \cdot C.$$

Now if $H = \{r_2(d), s_2(d) : d \in D\}$, then expression

$$\partial_H(B \parallel C)$$

describes a system of two communicating buffers. Some calculations, and using RSP, show that this system is a solution of the following recursive specification:

$$\begin{aligned} X &= \sum_{d \in D} r_1(d) \cdot c_2(d) \cdot X_d, \\ X_d &= s_3(d) \cdot X + \sum_{e \in D} r_1(e) \cdot s_3(d) \cdot c_2(e) \cdot X_e. \end{aligned}$$

This definition can be seen as defining a *two-place buffer*.

Renaming and encapsulation It is straightforward to extend the equational specification ACP, and its extensions, with renaming operators; cf. subsection 2.7.3.

State operator The extension of the theory ACP with either the simple or extended state operator is obtained in the same way as for the theories BPA or PA; see subsections 2.8 and 2.9.

Priority operator We can extend the theory ACP with the priority operator in the same way as we extended BPA_δ with that operator. For the details we refer to subsection 2.10.

In the system ACP_θ, we can describe forms of *asymmetric communication*. Notice that ACP itself features symmetric communication: both 'halves' of a communication action must be present before either one can proceed.

Example 3.6.2. A *put* mechanism describes a sending action that does not wait for a corresponding receiving action; the message is lost if it cannot be received. If a receiving action is present, then communication should occur.

For a port i and message d, we have actions $put_i(d)$, $r_i(d)$, $c_i(d)$ with communication function given by

$$\gamma\big(put_i(d), r_i(d)\big) = c_i(d) \qquad \gamma \text{ not defined otherwise.}$$

The priority ordering is given by $put_i(d) < c_i(d)$ for all d. Then, if S has actions $put_i(d)$, and R actions $r_i(d)$, put communication is described by the expression

$$\partial_H \circ \theta(S \parallel R),$$

where $H = \{r_i(d) : d \in D\}$.

Table 53. An extra axiom for Kleene's binary star operator.

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

Similarly, we can describe a *get* mechanism, where a process tries to receive a message. When no message is available, an error message $\perp$ will be read. We have actions $get_i(d)$, $s_i(d)$, $c_i(d)$ $(d \in D)$, and an action $get_i(\perp)$. Communication is given by

$$\gamma\big(s_i(d), get_i(d)\big) = c_i(d) \qquad \gamma \text{ not defined otherwise,}$$

and a priority ordering $get_i(\perp) < c_i(d)$ for all d. The system is described as

$$\partial_H \circ \theta(S \parallel R),$$

where $H = \{get_i(d), s_i(d) : d \in D\}$, and R typically has the form

$$R \quad = \quad \sum_{d \in D} get_i(d) \cdot X_d + get_i(\perp) \cdot X_\perp.$$

This material on put and get communication is based on [Bergstra, 1985], more information can also be found in [Baeten and Weijland, 1990].

Iteration We can extend ACP with iteration by adding the defining axioms for * in table 37 to the ones for ACP and one more axiom that gives the relation between Kleene's star and the encapsulation operator. We give this axiom in table 53. We denote this system by ACP*. Only for BPA* the completeness is proved at the time of writing this survey.

Process creation We discuss the extension of ACP with the process creation operator E_ϕ (for the basic definitions we refer to subsection 3.3.1). We recall that the process creation operator is defined in terms of the parallel composition. So it will not be surprising that we need to impose restrictions on the communication behaviour of the special atomic actions $cr(d)$. The restrictions are that $cr(d)$ does not communicate and is not the result of any communication ($cr(d)$ is not a communication action). In a formula: $cr(d) \,|\, a = \delta$ and for all $a, b \in A : a \,|\, b \neq cr(d)$. We note that lemma 3.3.1 only holds when the communication function is trivial; that is, for all $a, b \in A : \gamma(a, b)$ is undefined.

Furthermore, the only difference with the discussion in subsection 3.3.1 is the restrictions on the $cr(d)$-actions with respect to the communication. So for more details we refer to that subsection.

Inconsistent combinations If we combine *unguarded* recursion with ACP$_\theta$ we will find an inconsistency. We show this with an example that originates

Table 54. The interaction between σ_d and communication and encapsulation.

$\underline{\underline{a}} \mid \sigma_d(x) = \underline{\underline{\delta}}$	DTM5
$\sigma_d(x) \mid \underline{\underline{a}} = \underline{\underline{\delta}}$	DTM6
$(\underline{\underline{a}} \cdot x) \mid \sigma_d(y) = \underline{\underline{\delta}}$	DTM7
$\sigma_d(x) \mid (\underline{\underline{a}} \cdot y) = \underline{\underline{\delta}}$	DTM8
$\sigma_d(x) \mid \sigma_d(y) = \sigma_d(x \mid y)$	DTM9
$\partial_H\big(\sigma_d(x)\big) = \sigma_d\big(\partial_H(x)\big)$	DTD

from [Baeten and Bergstra, 1988b]. Suppose that there are three atomic actions r, s, and c and $r \mid s = c$. Let $c > s$ be the partial ordering. Now consider the following recursion equation:

$$X = r \parallel \big(\theta \circ \partial_{\{c\}}(s + X)\big).$$

With the operational rules we can infer that $X \xrightarrow{c} \surd \iff X \not\xrightarrow{c} \surd$.

3.6.1 Discrete time

In this subsection we add relative discrete time to ACP. In subsection 3.4.1, we already extended PA_δ with this feature. So we only need to clarify the interaction between the discrete time unit delay σ_d and the communication merge and the relation between σ_d and the encapsulation operator.

The theory The equational specification $\mathrm{ACP_{dt}}$ consists of the signature that is the union of the signatures of ACP (with $\underline{\underline{a}}$ instead of a for $a \in A_\delta$) and $\mathrm{PA}_{\delta\mathrm{dt}}$. The equations of this specification are the ones of ACP (again with $\underline{\underline{a}}$ instead of a) plus the rules of table 47 (they represent the interaction between σ_d and the left merge) and some new axioms that we present in table 54; they express the interaction between the discrete time unit delay operator and the communication merge and the interaction between σ_d and the encapsulation operator.

Termination The termination of a term rewriting system associated to $\mathrm{ACP_{dt}}$ can be obtained with the aid of the method that we discussed in subsection 2.2.2. We can prove the termination by merging the termination proofs of ACP and $\mathrm{PA}_{\delta\mathrm{dt}}$ and a small addition.

Theorem 3.6.3. *The term rewriting system consisting of the rewrite rules for* ACP *(see definition 3.5.3) plus the axioms listed in tables 47 and 54 oriented from left to right is strongly normalizing (or terminating).*

Proof. The partial ordering on the signature is as follows. Take the one for ACP; see figure 8. Now add the following to this partial ordering:

Table 55. The additional rules for the communication merge and the encapsulation operator.

$\dfrac{x \xrightarrow{\sigma} x',\, y \xrightarrow{\sigma} y'}{x \mid y \xrightarrow{\sigma} x' \mid y'}$	$\dfrac{x \xrightarrow{\sigma} x'}{\partial_H(x) \xrightarrow{\sigma} \partial_H(x')}$

- $+ > \sigma_d$,
- $\sigma_d > \underline{\underline{\delta}}$,
- $\sigma_d > \partial_H$.

With the proofs of theorems 3.5.4 and 3.4.1 we find that the term rewriting system associated to $\mathrm{ACP_{dt}}$ is strongly normalizing. We omit these inferences as they are straightforward. ∎

With the above theorem it is easy to obtain an elimination result for closed terms.

Theorem 3.6.4. *The equational specification* $\mathrm{ACP_{dt}}$ *has the elimination property for* $\mathrm{BPA_{\delta dt}}$.

Proof. Easy. ∎

Semantics The semantics of $\mathrm{ACP_{dt}}$ can be given with a term deduction system $T(\mathrm{ACP_{dt}})$. Its signature is that of $\mathrm{ACP_{dt}}$. The rules are those of ACP (with $\underline{\underline{a}}$ instead of a for $a \in A_\delta$), those in table 48 (they concern the merge and the left merge), and the operational rules that we present in table 55. The two rules in table 55 define what kind of σ transitions the communication merge and the encapsulation operator can perform.

Theorem 3.6.5. *The set of closed* $\mathrm{ACP_{dt}}$ *terms modulo strong bisimulation equivalence is a model of* $\mathrm{ACP_{dt}}$.

Proof. Bisimulation equivalence is a congruence, so we only need to check the soundness of the axioms. The only axioms that we have not checked yet in other soundness proofs are the ones of table 54. They are all very easy. For DTM5–8 we only need to relate the left- and right-hand side. For DTM9 and DTD we additionally include the diagonal. ∎

Theorem 3.6.6. *The equational specification* $\mathrm{ACP_{dt}}$ *is a conservative extension of the equational specification* $\mathrm{BPA_{\delta dt}}$.

Proof. Easy. ∎

Now we have all the prerequisites to state the completeness theorem for $\mathrm{ACP_{dt}}$. The proof is as usual and therefore omitted.

Theorem 3.6.7. *The axiom system* $\mathrm{ACP_{dt}}$ *is a complete axiomatization of the set of closed* $\mathrm{ACP_{dt}}$ *terms modulo bisimulation equivalence.*

3.7 Decidability and expressiveness results in ACP

In subsection 2.14 we discussed the decidability of bisimulation equivalence for BPArec and we showed that BPArec can express non-regular processes. In this subsection we will briefly discuss decidability and expressiveness results for the PArec and ACPrec families.

3.7.1 Decidability

At the time of writing this survey the results are that bisimulation equivalence is undecidable for ACPrec and the problem is open for PArec. However, some results have been obtained in the direction of PArec. For the so-called Basic Parallel Processes (BPP) [Christensen *et al.*, 1993] the problem is solved: BPP is decidable. The equational theory of BPP is close to PA_δrec with prefix sequential composition instead of sequential composition.

We will formulate these results below.

The next theorem is due to [Bergstra and Klop, 1984b]. For the proof of this fact we refer to [Bergstra and Klop, 1984b].

Theorem 3.7.1. *The bisimulation equivalence problem for finitely recursively defined processes over* ACP *is undecidable.*

For the decidability result on basic parallel processes we briefly introduce the syntax and semantics of this system.

Basic parallel processes We will introduce the syntax and semantics of BPP below using the notation that we are used to in this survey. We have a special constant δ, alternative composition $+$, parallel composition $\|$, and a unary prefix operator $a\_$, called prefix sequential composition, for all $a \in A$, where A is some set. Now if we also add recursion we have the syntax of BPP.

The semantics of BPP is given by means of the term deduction system in table 56. For all the operators we have the usual operational rules but only the non-predicate parts. We have not seen the well-known operational characterization of prefix sequential composition before in this chapter. We give the rules for BPP in one table for the sake of ease.

We note that bisimulation equivalence in this case is just the one that we defined in definition 2.2.28 without the predicate part.

The next theorem is taken from [Christensen *et al.*, 1993]. For the proof of this fact we refer to [Christensen *et al.*, 1993].

Theorem 3.7.2. *Bisimulation equivalence is decidable for basic parallel processes* (BPP).

3.7.2 Expressiveness

In this subsection we discuss various expressivity results. It turns out that ACPrec is more expressive than PArec and that the latter is more expressive than BPArec.

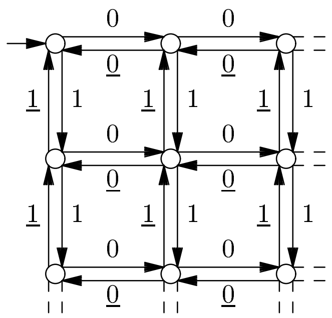

Fig. 9. A deduction graph of a bag over two datum elements.

The bag We consider a so-called bag of unbounded capacity. A bag is a process able to input data elements that reappear in some arbitrary order. Let D be a finite set of such datum elements containing more than one datum. Suppose that we have atomic actions for all $d \in D$:

- $r_1(d)$ means put a d in the bag;
- $s_2(d)$ means remove a d from the bag.

The following recursive equation formally defines the bag.

$$B = \sum_{d \in D} r_1(d) \cdot \big(B \parallel s_2(d)\big).$$

It will be clear that B is definable over PArec. In figure 9 we depict the deduction graph of a bag over two datum elements 0 and 1. Note that we abbreviate $r_1(d)$ to d and $s_2(d)$ to $\underline{d}$ for $d = 0, 1$.

Next, we state that PArec is more expressive than BPArec. This theorem stems from [Bergstra and Klop, 1984a; Bergstra and Klop, 1995]. For its proof we refer to this paper.

Table 56. The operational semantics of BPP.

$ax \xrightarrow{a} x$	$\dfrac{\langle s_X \mid E\rangle \xrightarrow{a} y}{\langle X \mid E\rangle \xrightarrow{a} y}$
$\dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}$	$\dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$
$\dfrac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y}$	$\dfrac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'}$

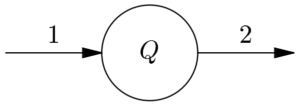

Fig. 10. A FIFO queue.

Theorem 3.7.3. *A bag over more than one datum element cannot be given by means of a finite recursive specification in* BPArec. *So,* PArec *is more expressive than* BPArec.

Remark 3.7.4. Observe that the bag can also be specified in BPP. But since there is a process (the stack) that can be defined in BPArec and *not* in BPP the systems are incomparable as far as expressivity is concerned. See [Christensen, 1993] for more details.

Next, we consider the expressivity of ACPrec over PArec. The following theorem is taken from [Bergstra and Klop, 1984a; Bergstra and Klop, 1995]. For more details on this result and its proof we refer to this paper.

Theorem 3.7.5. *The process* $p = ba(ba^2)^2(ba^3)^2(ba^4)^2 \ldots$ *cannot be defined in* PArec *with a set of atomic actions* $\{a, b\}$ *but* p *can be defined in* ACPrec *with atomic actions* $\{a, b, c, d\}$ *and with communication function* $\gamma(c, c) = a$, $\gamma(d, d) = b$ *(other communications yield* δ*).*

Next, we discuss a result that states that ACPrec + RN is more expressive than ACPrec.

The queue A queue is a process that transmits incoming data while preserving their order. See also figure 10. Such a process is also called a FIFO (First In First Out) queue. First, we describe the queue with input port 1 and output port 2 over a finite data set D by means of an infinite linear specification. As in 3.3.3, D^* is the set of words over D.

It is not hard to see that a queue with input port 1 and output port 2 over the data set D can be specified as follows:

$$\begin{aligned} Q = Q_\varepsilon &= \sum_{d \in D} r_1(d) \cdot Q_d, \\ Q_{\sigma d} &= s_2(d) \cdot Q_\sigma + \sum_{e \in D} r_1(e) \cdot Q_{e\sigma d}. \end{aligned}$$

We have the last equation for all $\sigma \in D^*$ and $d \in D$.

In figure 11 we give a deduction graph of a queue over two datum elements; note that we abbreviate $r_1(d)$ by d and $s_2(d)$ by $\underline{d}$ for $d = 0, 1$.

The next theorem states that there is no finite specification for the queue in ACPrec. This result is taken from [Baeten and Bergstra, 1988a];

the proof uses results from [Bergstra and Tiuryn, 1987]. For a proof we refer to [Baeten and Bergstra, 1988a].

Theorem 3.7.6. *The queue is not finitely definable over* ACPrec *using the usual read/send communication that we discussed in subsection 3.5.1.*

In [Baeten and Bergstra, 1988a] it is shown that in ACPrec + RN there exists a finite specification of the queue. For details we refer to [Baeten and Bergstra, 1988a].

Theorem 3.7.7. *The queue is finitely definable over* ACPrec + RN *using the usual read/send communication that we discussed in subsection 3.5.1.*

Next, we list some expressivity results for extensions of PA and ACP with iteration. These results are taken from [Bergstra *et al.*, 1994a].

Theorem 3.7.8. *If there are at least six atomic actions we have the expressivity results for the systems* BPA^*, BPA^*_δ, PA^*, PA^*_δ, *and* ACP^* *as in figure 12. The systems* BPA^*_δ *and* PA^* *are incomparable and for the other systems we have that a line down to a system indicates that the upper system is more expressive than the lower one.*

Recursion versus iteration In subsection 2.14 we devoted a small paragraph to the comparison of recursion as treated in subsection 2.3 and iteration (see subsection 2.11). We stated that the system BPAlin (BPA with finite linear recursion) is more expressive than BPA^* (see theorem 2.14.6). This result is in fact stronger: the regular system of figure 7 cannot be expressed in ACP^*.

In [Bergstra *et al.*, 1994a] it is shown that the regular process depicted in figure 7 can be expressed in ACP^* with abstraction.

The next theorem is taken from [Bergstra *et al.*, 1994a]. For the proof we refer to their paper.

Theorem 3.7.9. *Not every regular process can be expressed in* ACP^* *(not even using auxiliary actions).*

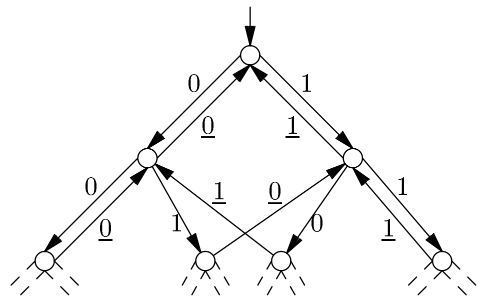

Fig. 11. A deduction graph for the queue.

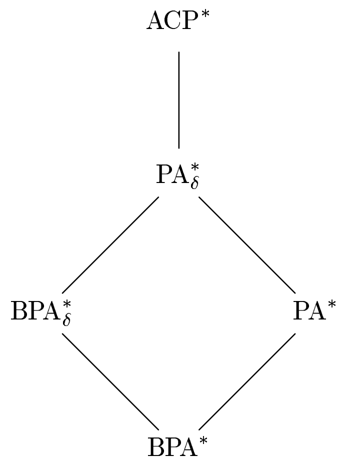

Fig. 12. Expressivity results for systems with iteration.

More information on the subject of expressiveness in ACP can be found in [Baeten *et al.*, 1987] and in [Vaandrager, 1993]. For more information on expressiveness in systems related to ACP we refer to [Ponse, 1992] or [Glabbeek, 1995].

4 Further reading

For those readers who want to know more about process algebra, we give some references. First of all, we want to mention a couple of textbooks in the area. A textbook for CCS-style process algebra is [Milner, 1989], for CSP style we refer to [Hoare, 1985], and for testing theory, there is [Hennessy, 1988]. In ACP style, the standard reference is [Baeten and Weijland, 1990]. The companion volume [Baeten, 1990] discusses applications of this theory. We also want to mention the proceedings of a workshop on ACP style process algebra [Ponse *et al.*, 1995].

When process algebra is applied to larger examples, the need arises to handle data structures also in a formal way. The combination of processes and data is treated in the theories LOTOS [Brinksma, 1987], PSF [Mauw and Veltink, 1993], or μCRL [Groote and Ponse, 1995].

Tool support in the use of process algebra is provided by most systems; a few references are [Boudol *et al.*, 1990], [Cleaveland *et al.*, 1990], [Godskesen *et al.*, 1989], [Lin, 1992], and [Veltink, 1993].

For an impression of the state of the art in concurrency theory we refer to the proceedings of the series of CONCUR conferences on concurrency theory: [Baeten and Klop, 1990], [Baeten and Groote, 1991], [Cleaveland, 1992], [Best, 1993], and [Jonsson and Parrow, 1994].

References

[Aceto and Hennessy, 1992] L. Aceto and M. Hennessy. Termination, deadlock, and divergence. *Journal of the ACM*, 39(1):147–187, January 1992.

[Aceto *et al.*, 1994] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111(1):1–52, 1994.

[America and Bakker, 1988] P. America and J.W. de Bakker. Designing equivalent semantic models for process creation. *Theoretical Computer Science*, 60:109–176, 1988.

[Austry and Boudol, 1984] D. Austry and G. Boudol. Algèbre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

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

[Baeten and Bergstra, 1988b] J.C.M. Baeten and J.A. Bergstra. Processen en procesexpressies. *Informatie*, 30(3):214–222, 1988. In Dutch.

[Baeten and Bergstra, 1991] J.C.M. Baeten and J.A. Bergstra. Recursive process definitions with the state operator. *Theoretical Computer Science*, 82:285–302, 1991.

[Baeten and Bergstra, 1992a] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra (extended abstract). In Cleaveland [1992], pages 401–420. Full version, report P9208b, Programming Research Group, University of Amsterdam, 1992.

[Baeten and Bergstra, 1992b] J.C.M. Baeten and J.A. Bergstra. Process algebra with signals and conditions. In M. Broy, editor, *Programming and Mathematical Methods, Proceedings Summer School Marktoberdorf 1991*, pages 273–323. Springer-Verlag, 1992. NATO ASI Series F88.

[Baeten and Bergstra, 1993] J.C.M. Baeten and J.A. Bergstra. Real space process algebra. *Formal Aspects of Computing*, 5(6):481–529, 1993.

[Baeten and Glabbeek, 1987] J.C.M. Baeten and R.J. van Glabbeek. Merge and termination in process algebra. In K.V. Nori, editor, *Proceedings 7^{th} Conference on Foundations of Software Technology and Theoretical Computer Science,* Pune, India, volume 287 of *Lecture Notes in Computer Science*, pages 153–172. Springer-Verlag, 1987.

[Baeten and Groote, 1991] J.C.M. Baeten and J.F. Groote, editors. *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[Baeten and Klop, 1990] J.C.M. Baeten and J.W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[Baeten and Vaandrager, 1992] J.C.M. Baeten and F.W. Vaandrager. An algebra for process creation. *Acta Informatica*, 29(4):303–334, 1992.

[Baeten and Verhoef, 1993] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Best [1993], pages 477–492.

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

[Baeten *et al.*, 1986] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[Baeten *et al.*, 1987] 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.

[Baeten *et al.*, 1991] J.C.M. Baeten, J.A. Bergstra, S. Mauw, and G.J. Veltink. A process specification formalism based on static COLD. In J.A. Bergstra and L.M.G. Feijs, editors, *Algebraic Methods II: Theory, Tools and Applications*, volume 490 of *Lecture Notes in Computer Science*, pages 303–335. Springer-Verlag, 1991.

[Baeten *et al.*, 1992] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. In Cleaveland [1992], pages 472–485.

[Baeten *et al.*, 1993] 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, July 1993.

[Baeten, 1990] J.C.M. Baeten, editor. *Applications of Process Algebra.* Cambridge Tracts in Theoretical Computer Science 17. Cambridge University Press, 1990.

[Bakker and Zucker, 1982] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1/2):70–120, 1982.

[Bergstra and Klint, 1994] J.A. Bergstra and P. Klint. The ToolBus—a component interconnection architecture. Report P9408, Programming Research Group, University of Amsterdam, 1994.

[Bergstra and Klop, 1982] J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam, 1982.

[Bergstra and Klop, 1984a] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th ICALP, Antwerpen*, volume 172 of *Lecture Notes in Computer Science*, pages 82–95. Springer-Verlag, 1984. Extended abstract, full version appeared in [Ponse *et al.*, 1995].

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

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

[Bergstra and Klop, 1986] 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, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85, Math. Research 31*, pages 9–23, Berlin, 1986. Akademie-Verlag.

[Bergstra and Klop, 1995] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In Ponse et al. [1995], pages 1–25. Full version of [Bergstra and Klop, 1984a].

[Bergstra and Tiuryn, 1987] J.A. Bergstra and J. Tiuryn. Process algebra semantics for queues. *Fundamenta Informaticae*, X:213–224, 1987.

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

[Bergstra *et al.*, 1985] J.A. Bergstra, J.W. Klop, and J.V. Tucker. Process algebra with asynchronous communication mechanisms. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 76–95. Springer-Verlag, 1985.

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

[Bergstra *et al.*, 1994b] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with combinators. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of CSL '93*, volume 832 of *Lecture Notes in Computer Science*, pages 36–65. Springer-Verlag, 1994.

[Bergstra *et al.*, 1994c] J.A. Bergstra, A. Ponse, and J.J. van Wamel. Process algebra with backtracking. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Symposium "A Decade of Concurrency: Reflections and Perspectives"*, volume 803 of *Lecture Notes in Computer Science*, pages 46–91. Springer-Verlag, 1994.

[Bergstra, 1985] J.A. Bergstra. Put and get, primitives for synchronous unreliable message passing. Logic Group Preprint Series Nr. 3, CIF, State University of Utrecht, 1985.

[Bergstra, 1990] J.A. Bergstra. A process creation mechanism in process algebra. In Baeten [1990], pages 81–88.

[Best, 1993] E. Best, editor. *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science.* Springer-Verlag, August 1993.

[Bezem and Groote, 1994] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In Jonsson and Parrow [1994], pages 401–416.

[Blanco, 1995] J.O. Blanco. Definability with the state operator in process algebra. In Ponse et al. [1995], pages 218–241.

[Bloom *et al.*, 1988] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: Preliminary report. In *Conference Record of the* 15^{th} *ACM Symposium on Principles of Programming Languages,* San Diego, California, pages 229–239, 1988. Full version available as Technical Report 90-1150, Department of Computer Science, Cornell University, Ithaca, New York, August 1990. Accepted to appear in *Journal of the ACM.*

[Bol and Groote, 1991] R.N. Bol and J.F. Groote. The meaning of negative premises in transition system specifications (extended abstract). In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 481–494. Springer-Verlag, 1991. Full version appeared as Report CS-R9054, CWI, Amsterdam, 1990.

[Boudol *et al.*, 1990] G. Boudol, V. Roy, R. De Simone, and D. Vergamini. Process calculi, from theory to practice: verification tools. In Sifakis [1990], pages 1–10.

[Brinksma, 1987] E. Brinksma. *Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour* ISO/TC97/SC21/N DIS8807, International Standardisation Organisation, 1987.

[Brookes *et al.*, 1984] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[Caucal, 1990] D. Caucal. On the transition graphs of automata and grammars. Report 1318, INRIA, 1990.

[Christensen *et al.*, 1992] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In Cleaveland [1992], pages 138–147.

[Christensen *et al.*, 1993] S. Christensen, Y. Hirschfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In Best [1993], pages 143–157.

[Christensen, 1993] S. Christensen. *Decidability and decomposition in process algebra.* PhD thesis, University of Edinburgh, 1993.

[Cleaveland *et al.*, 1990] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench. In Sifakis [1990], pages 24–37.

[Cleaveland, 1992] W.R. Cleaveland, editor. *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

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

[De Simone, 1985] R. De Simone. Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Formal Models and Semantics. Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier – MIT Press, Amsterdam, 1990.

[Dershowitz, 1987] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69–116, 1987.

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

[Fokkink, 1994] W.J. Fokkink. The *tyft/tyxt* format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proceedings 2nd International Symposium on Theoretical Aspects of Computer Software (TACS'94),* Sendai, Japan, volume 789 of *Lecture Notes in Computer Science*, pages 440–453. Springer Verlag, 1994.

[Giacalone *et al.*, 1990] A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B. Jones, editors, *Proceedings IFIP Working Conference on Programming Concepts and Methods,* Sea of Gallilee, Israel. North-Holland, 1990.

[Glabbeek and Weijland, 1989] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In G.X. Ritter, editor, *Information Processing 89*, pages 613–618. North-Holland, 1989. Full version available as Report CS-R9120, CWI, Amsterdam, 1991.

[Glabbeek, 1987] R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebra. In F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, editors, *Proceedings STACS 87*, volume 247 of *Lecture Notes in Computer Science*, pages 336–347. Springer-Verlag, 1987.

[Glabbeek, 1990] R.J. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [1990], pages 278–297.

[Glabbeek, 1993] R.J. van Glabbeek. The linear time – branching time spectrum ii (extended abstract). In Best [1993], pages 66–81.

[Glabbeek, 1995] R.J. van Glabbeek. On the expressiveness of ACP (extended abstract). In Ponse et al. [1995], pages 188–217.

[Godskesen *et al.*, 1989] J. Godskesen, K.G. Larsen, and M. Zeeberg.

TAV–tools for autimatic verification. Technical report R89–19, University of Aalborg, 1989.

[Groote and Ponse, 1994] J.F. Groote and A. Ponse. Process algebra with guards: combining Hoare logic and process algebra. *Formal Aspects of Computing*, 6(2):115–164, 1994.

[Groote and Ponse, 1995] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In Ponse et al. [1995], pages 26–62.

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

[Groote, 1990a] J.F. Groote. A new strategy for proving ω–completeness with applications in process algebra. In Baeten and Klop [1990], pages 314–331.

[Groote, 1990b] J.F. Groote. Transition system specifications with negative premises (extended abstract). In Baeten and Klop [1990], pages 332–341. Full version appeared as Technical Report CS-R8950, CWI, Amsterdam, 1989.

[Hennessy, 1988] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.

[Hoare *et al.*, 1987] C.A.R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Surfin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.

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

[Jonsson and Parrow, 1994] B. Jonsson and J. Parrow, editors. *Proceedings CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science.* Springer-Verlag, 1994.

[Jouannaud and Kirchner, 1986] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15:1155–1194, 1986.

[Jouannaud and Muñoz, 1984] J.-P. Jouannaud and M. Muñoz. Termination of a set of rules modulo a set of equations. In R. E. Shostak, editor, *7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 175–193. Springer-Verlag, 1984.

[Kamin and Lévy, 1980] S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering, 1980. Unpublished manuscript.

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

[Klop, 1992] J.W. Klop. Term rewriting systems. In *Handbook of Logic* *in Computer Science, Volume II*, pages 1–116. Oxford University Press, 1992.

[Klusener, 1993] A.S. Klusener. *Models and axioms for a fragment of real time process algebra.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, December 1993.

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

[Larsen and Skou, 1992] K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [1992], pages 456–471.

[Lin, 1992] H. Lin. PAM: a process algebra manipulator. In K.G. Larsen and A. Skou, editors, *Proceedings of the 3rd International Workshop on Computer Aided Verification,* Aalborg, Denmark, July 1991, volume 575 of *Lecture Notes in Computer Science*, pages 176–187. Springer-Verlag, 1992.

[Mauw and Veltink, 1993] S. Mauw and G.J. Veltink, editors. *Algebraic Specification of Communication Protocols.* Cambridge Tracts in Theoretical Computer Science 36. Cambridge University Press, 1993.

[Milne, 1983] G.J. Milne. CIRCAL: a calculus for circuit description. *Integration*, 1:121–160, 1983.

[Milner *et al.*, 1992] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

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

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

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

[Moller and Tofts, 1990] F. Moller and C.M.N. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [1990], pages 401–415.

[Moller, 1989] F. Moller. *Axioms for concurrency.* PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.

[Nicollin and Sifakis, 1994] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: theory and application. *Information and Computation*, 114(1):131–178, 1994.

[Park, 1981] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Peackock, 1830] G. Peackock. *A treatise of algebra.* Cambridge, 1830.

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

[Ponse *et al.*, 1995] A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors. *Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing. Springer-Verlag, 1995.

[Ponse, 1991] A. Ponse. Process expressions and Hoare's logic: showing an irreconcilability of context-free recursion with Scott's induction rule. *Information and Computation*, 95(2):192–217, 1991.

[Ponse, 1992] A. Ponse. Computable processes and bisimulation equivalence. Report CS-R9207, CWI, Amsterdam, January 1992. To appear in Formal Aspects of Computing.

[Ponse, 1993] A. Ponse. Personal communication, June 1993.

[Ritchie and Thompson, 1974] D.M. Ritchie and K. Thompson. The UNIX time-sharing system. *Communications of the ACM*, 17(7):365–375, 1974.

[Sifakis, 1990] J. Sifakis, editor. *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems,* Grenoble, France, June 1989, volume 407 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[Tofts, 1990] C.M.N. Tofts. A synchronous calculus of relative frequency. In Baeten and Klop [1990], pages 467–480.

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

[Vaandrager, 1990a] F.W. Vaandrager. Process algebra semantics of POOL. In Baeten [1990], pages 173–236.

[Vaandrager, 1990b] F.W. Vaandrager. Two simple protocols. In Baeten [1990], pages 23–44.

[Vaandrager, 1993] F.W. Vaandrager. Expressiveness results for process algebras. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Semantics: Foundations and Applications,* Beekbergen, The Netherlands, June 1992, volume 666 of *Lecture Notes in Computer Science*, pages 609–638. Springer-Verlag, 1993.

[Veltink, 1993] G.J. Veltink. The PSF toolkit. *Computer Networks and ISDN Systems*, 25:875–898, 1993.

[Verhoef, 1992] C. Verhoef. *Linear Unary Operators in Process Algebra.* PhD thesis, University of Amsterdam, June 1992.

[Verhoef, 1994a] C. Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. In Jonsson and Parrow [1994], pages 433–448.

[Verhoef, 1994b] C. Verhoef. A general conservative extension theorem in process algebra. In E.-R. Olderog, editor, *Programming Concepts,* *Methods and Calculi (PROCOMET '94)*, volume A-56 of *IFIP Transactions A: Computer Science and Technology*, pages 149–168. North-Holland, 1994.

[Vrancken, 1986] J.L.M. Vrancken. The algebra of communicating processes with empty process. Report FVI 86-01, Dept. of Computer Science, University of Amsterdam, 1986.

[Weijland, 1989] W.P. Weijland. *Synchrony and asynchrony in process algebra.* PhD thesis, University of Amsterdam, 1989.

Author index

Subject index

Notation index