UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Process Algebra with Guards. Combining Hoare Logic and Process Algebra

Groote, J.F.; Ponse, A.

Publication date
1994

Published in
Formal Aspects of Computing

Link to publication

Citation for published version (APA):
Groote, J. F., & Ponse, A. (1994). Process Algebra with Guards. Combining Hoare Logic and Process Algebra. *Formal Aspects of Computing*, *6*, 115-164.

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:13 Oct 2026

Formal Aspects of Computing (1994) 6: 115–164
© 1994 BCS

Formal Aspects of Computing

Process Algebra with Guards: Combining Hoare Logic with Process Algebra

Jan Friso Groote[†1] and Alban Ponse[§2]

[†] University of Utrecht, Department of Philosophy, Utrecht, The Netherlands
[§] University of Amsterdam, Faculty of Mathematics and Computer Science, Amsterdam, The Netherlands

Keywords: Process Algebra; Hoare Logic; Guards; Structural/ed Operational Semantics; Bisimulation; Completeness; Partial Correctness; Conditionals

Abstract. We extend process algebra with guards, comparable to the guards in guarded commands or conditions in common programming constructs such as 'if – then – else – fi' and 'while – do – od'.

The extended language is provided with an operational semantics based on transitions between pairs of a process and a (data-)state. The data-states are given by a data environment that also defines in which data-states guards hold and how atomic actions (non-deterministically) transform these states. The operational semantics is studied modulo strong bisimulation equivalence. For basic process algebra (without operators for parallelism) we present a small axiom system that is complete with respect to a general class of data environments. Given a particular data environment $\mathscr{S}$ we add three axioms to this system, which is then again complete, provided weakest preconditions are expressible and $\mathscr{S}$ is sufficiently deterministic.

Then we study process algebra with parallelism and guards. A two phase-calculus is provided that makes it possible to prove identities between parallel processes. Also this calculus is complete. In the last section we show that partial correctness formulas can easily be expressed in this setting. We use process algebra with guards to prove the soundness of a Hoare logic for linear processes by translating proofs in Hoare logic into proofs in process algebra.

[1] Supported by ESPRIT Basic Research Action no. 3006 (CONCUR) and by RACE project no. 1046 (SPECS).

[2] Supported by RACE project no. 1046 (SPECS).

Correspondence and offprint requests to: Alban Ponse, University of Amsterdam, Faculty of Mathematics and Computer Science, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands. Email: alban@fwi.uva.nl.

1. Introduction

Hoare logic has been introduced in 1969 to prove correctness of programs [Hoa69]. Since then it has been applied to many problems, and it has been thoroughly studied (see [Apt81, Apt84] for an overview). In Hoare logic a program is considered to be a state transformer; the initial state is transformed to a final state. The correctness of a program is expressed by pre- and post-conditions.

More recently processes, where a process is the behaviour of a system, have attracted attention. This has led to several process calculi (CCS [Mil80, Mil89], CSP [Hoa85], ACP [BeK84a, BaW90] and Meije [AuB84]). In these calculi correctness is often expressed by equations saying that a specification and an implementation are equivalent in some sense. These equivalences are mainly based on observations: two processes are equivalent if some observer cannot distinguish between the two. A classification of process equivalences has been described in [Gla90, Gla93].

It seems a natural and useful question how Hoare logic and process algebra can be integrated. In this paper we provide an answer in two steps. First we extend process algebra with *guards*. Depending on the state, a guard can either be transparent such that it can be passed, or it can block and prevent subsequent processes from being executed. Typical for our approach is that a guard *itself* represents a process. With this construct we can easily express the guarded commands of Dijkstra [Dij76] and the guards occurring in several languages such as LOTOS [ISO87] and CRL [SPE90]. A nice property of the guards in our framework is that they constitute a Boolean algebra.

Using guards a partial correctness formula

$$\{\alpha\}\, p\, \{\beta\}$$

with α, β guards and p representing some process can be expressed by the algebraic equation

$$\alpha p = \alpha p \beta$$

saying that if process p starts in a state where the guard α holds, then it follows that the guard β holds when p terminates. As far as we know such equations modelling partial correctness formulas were first given by Manes and Arbib [MaA86].

We provide process terms (with guards) with an operational semantics involving state transformations. This semantics is based on transitions between configurations (p, s) where p is a process term and s is the state. To avoid confusion between 'state' and 'configuration' (also often called state in process algebra) we consequently use the term *data-state* instead of 'state'. We assume that data-states are given by some *data environment* that also prescribes in which data-states guards hold and how atomic actions (non-deterministically) transform data-states.

We study the operational semantics modulo strong bisimulation equivalence [Par81] and we come up with several axiomatisations. In the case of Basic Process Algebra (BPA) with the standard operators $+$ (choice) and $\cdot$ (sequential composition), termination constants (δ, ϵ) and guards we present two axiom systems, BPA^4_G and $\mathrm{BPA}_G(\mathcal{S})$. The system BPA^4_G is complete for finite processes with respect to a general class of data environments. It contains three simple and one somewhat more involved axiom besides the nine that are standard for BPA

with termination constants. The axioms of BPA_G^4 enable us to derive general facts about processes with guards that do not depend on a particular data environment.

The axiom system $BPA_G(\mathcal{S})$ applies when one wants to prove equivalences between processes for a particular data environment $\mathcal{S}$. This axiom system is defined only if *weakest preconditions* are expressible and $\mathcal{S}$ is *sufficiently deterministic*. It contains the axioms of BPA_G^4 together with three new axiom schemes that depend on $\mathcal{S}$. As an example we use $BPA_G(\mathcal{S})$ to prove the correctness of a well-known small program in a completely algebraic manner.

Parallel operators fit easily in the process algebra framework. In Hoare logic, however, parallelism turns out to be rather intricate; proof rules for parallel operators are often substantial [OwG76, Lam80, Sti88]. In our setup we cannot completely avoid the difficulties caused by parallel operators in Hoare logic, but we can deal with them in a simple algebraic way. We introduce a new set of axioms, called ACP_G that enables us to rewrite every process term to a term without parallel operators. Then using BPA_G^4 or $BPA_G(\mathcal{S})$ we can verify the equivalences we are interested in. We apply these techniques to an example.

In the last section of this paper we show that process algebra with guards can indeed be used to verify partial correctness formulas, even in a setting with parallelism. Furthermore we apply $BPA_G(\mathcal{S})$ to show soundness of a Hoare logic for process algebra with linear processes [Pon91]. The proof uses a canonical translation of proofs in Hoare logic into proofs in process algebra.

2. Basic Process Algebra with Guards

In this section we extend the basic theory BPA (Basic Process Algebra, see e.g. [BeK84b, BaW90]) with *guards*. These guards are comparable to those in the guarded commands of Dijkstra [Dij76], or to the conditions in programming constructs as **if - then - else - fi** and **while - do - od**. We call this extension BPA_G (BPA with Guards).

2.1. Signature and Axioms

The theory of BPA_G has two parameters: a set of *atomic actions* and a set of *atomic guards*. Atomic actions represent the basic activities that processes can perform, such as reading input, incrementing counters and so forth. Guards represent constructs that (relative to a structure defining their interpretation) are either transparent such that they can be passed, or block and prevent subsequent processes from being executed.

Let A be a set of atomic actions with typical elements $a, b, \ldots$ For each atomic action a the signature of BPA_G, denoted as $\Sigma(BPA_G)$, contains an identically named constant a. Let G_{at} be a set of atomic guards disjoint with A, and also disjoint with $\{\delta, \epsilon\}$. We extend G_{at} to the set G of *basic* guards with typical elements $\phi, \psi, \ldots$ where basic guards are defined by the following syntax:

$$\phi ::= \delta \mid \epsilon \mid \neg\phi \mid \psi \in G_{at}.$$

In particular the process algebra constants δ and ϵ are considered as basic guards: δ is the guard that always blocks, and ϵ is the guard that can always be passed. Furthermore $\neg$ is the negation operator on basic guards. For each basic guard ϕ the signature $\Sigma(BPA_G)$ contains a constant ϕ. We also have the binary infix

constants:	a	for any atomic action $a \in A$
	ϕ	for any basic guard $\phi \in G$
binary operators :	$+$	alternative composition (sum)
	$\cdot$	sequential composition (product)

Fig. 1. The signature $\Sigma(\mathrm{BPA}_G)$.

operators $+$ (alternative composition) and $\cdot$ (sequential composition) available. We summarise the signature $\Sigma(\mathrm{BPA}_G)$ in Fig. 1.

Example 2.1. The addition of guards to process algebra brings it far closer to existing specification and programming languages. This can be seen by modelling an imperative language in process algebra with guards. The actions of process algebra represent assignments, which have the form

$$[x := t],$$

where t is some expression and x is a variable. In order to describe the semantics of these assignments we must use some kind of store for the value of x. Generally, this is represented by a valuation that maps variables to values. In sequential programming languages this valuation is part of the 'state' of a program. As the word 'state' is also commonly used in process algebra with a close but different meaning, we systematically use the word 'data-state'.

The data-state can influence the course of action of a program or process. Guards are used to describe this. They block for some, and are transparent for other data states. In the setting of this example, guards have the form:

$$\langle t = u \rangle$$

with the interpretation that $\langle t = u \rangle$ holds in some valuation iff t and u represent the same value. Now the conditional programming construct

if $t = u$ **then** $x := t$ **else** `skip` **fi**

can be translated into a process term in the language of BPA_G by

$$\langle t = u \rangle \cdot [x := t] + \neg\langle t = u \rangle \cdot \epsilon$$

where ϵ is the special guard (process) that always holds.

In this paper we introduce several axiom systems for reasoning on an algebraic level about the behaviour of process terms containing guards. A typical example of this type of reasoning is expressed by the law

$$x \cdot \epsilon = x$$

(where x ranges over process terms) saying that the always successful guard can be omitted in sequential composition. This law implies that the process term above equals $\langle t = u \rangle \cdot [x := t] + \neg\langle t = u \rangle$. For another example consider the law

$$\alpha \cdot (x + y) = \alpha \cdot x + \alpha \cdot y$$

where α ranges over guards. This second law expresses that the moment of evaluation of a guard and the moment of choice are interchangeable. □

Remark 2.2. The special constants δ and ϵ are already well-known in process algebra: δ (*inaction* or *deadlock*) represents the process that cannot perform any

activity and prevents subsequent processes from being executed; ϵ denotes a process that can do nothing but terminate and is called the *empty process* (see e.g. Baeten and Weijland [BaW90]). □

Throughout this text let $V = \{x, y, z, \ldots\}$ be a set of variables. Process terms, or shortly terms, over $\Sigma(\mathrm{BPA}_G)$ are constructed from the variables in V and the elements of $\Sigma(\mathrm{BPA}_G)$. In terms the function symbol $\cdot$ is generally left out, and brackets are omitted according to the convention that $\cdot$ binds stronger than $+$. The symbol $\equiv$ is used to denote syntactic equivalence (modulo associativity) between terms. Finally, letters $t, t', \ldots$ range over open terms and $p, q, r, \ldots$ over closed terms.

In the sequel results are often proved by reasoning on the structure of process terms. In order to give some general definitions, let the symbol Σ range over all signatures we consider in this paper. Any such signature Σ always extends the signature $\Sigma(\mathrm{BPA}_G)$ defined above. Terms over Σ are constructed in the usual way and may contain variables from V. We define two elementary notions:

Definition 2.3. Let $\Gamma(\Sigma)$ denote some axiom system defined over a signature Σ and let t, t' be terms over Σ.

1. t and t' are *provably equal* in $\Gamma(\Sigma)$, notation

   $$\Gamma(\Sigma) \vdash t = t'$$

   iff there exists a proof of $t = t'$ using the axioms of $\Gamma(\Sigma)$ and the usual inference rules for equality (stating that '=' is a congruence relation),
2. t is a *(provable) summand* of t' in $\Gamma(\Sigma)$, notation

   $$\Gamma(\Sigma) \vdash t \subseteq t'$$

   iff $\Gamma(\Sigma) \vdash t + t' = t'$. We write $t' \supseteq t$ for $t \subseteq t'$. The relation $\subseteq$ is called *summand inclusion.* □

In proofs we adopt the convention to write $t = t'$ instead of $\Gamma(\Sigma) \vdash t = t'$ and $t \neq t'$ instead of $\Gamma(\Sigma) \nvdash t = t'$ and a similar convention with respect to summand inclusion.

The axioms presented in Fig. 2 constitute the axiom system BPA_G^4. In this figure ϕ ranges over G and a over A. These axioms describe the basic identities between terms over $\Sigma(\mathrm{BPA}_G)$. The operator $+$ is commutative, associative and idempotent (A1 – A3). The operator $\cdot$ right-distributes over $+$ and is associative (A4, A5). Note that left distributivity of $\cdot$ over $+$ is absent. Furthermore δ behaves as the neutral element for $+$, and ϵ as the neutral element for $\cdot$ (A6 – A9). The axioms A1 – A9 form the system $\mathrm{BPA}_{\delta\epsilon}$ as described in e.g. [BaW90].

The axioms G1 – G3 are new in process algebra and describe the expected identities between guards. G1 and G2 express that a basic guard always behaves dually to its negation: ϕ holds in a data-state s iff $\neg\phi$ does not and vice versa. The axiom G3 states that $+$ does not change the interpretation of a basic guard ϕ. It does not matter whether the choice is exercised before or after the evaluation of ϕ. Notice the $\mathrm{BPA}_{\delta\epsilon}$-derivability for the δ and ϵ-instances of G3. The last new axiom G4 can be explained as follows: the process $a(\phi x + \neg\phi y)$, where a is an atomic action, behaves either like ax or ay, depending on the data-state resulting from the execution of a. As a consequence its behaviour is a summand of $ax + ay$. The a in this axiom may not be replaced by a larger process term. If it is for instance replaced by the term $a \cdot b$ then after a has happened, it is in general

(A1)	$x+y=y+x$	(G1)	$\phi \cdot \neg\phi = \delta$
(A2)	$x+(y+z)=(x+y)+z$	(G2)	$\phi + \neg\phi = \epsilon$
(A3)	$x+x=x$	(G3)	$\phi(x+y)=\phi x+\phi y$
(A4)	$(x+y)z=xz+yz$		
(A5)	$(xy)z=x(yz)$		
(A6)	$x+\delta=x$		
(A7)	$\delta x=\delta$	(G4)	$a(\phi x+\neg\phi y)\subseteq ax+ay$
(A8)	$\epsilon x=x$		
(A9)	$x\epsilon=x$		

Fig. 2. The axioms of BPA^4_G where $\phi \in G$ and $a \in A$.

possible to execute b and either arrive in a data-state where ϕ holds, or arrive in a data-state where $\neg\phi$ holds (b can affect the data-state in a nondeterministic manner). Neither abx nor aby covers this behaviour. Hence $ab(\phi x+\neg\phi y)$ need not be a summand of $abx+aby$. The axiom G4 is not derivable from the first three 'guard'-axioms. The superscript 4 in BPA^4_G expresses that there are four axioms referring to guards. We do not always consider all guard axioms. In particular the system BPA^3_G, containing all BPA^4_G-axioms except G4 plays an important role in this paper.

Example 2.4. We illustrate the use of the BPA^3_G-axioms by showing that if for two terms t and t' over $\Sigma(\mathrm{BPA}_G)$ we have $\mathrm{BPA}^3_G \vdash t+t'=\delta$, then also $\mathrm{BPA}^3_G \vdash t=\delta$:

$$
\begin{array}{lllll}
\mathrm{BPA}^3_G \vdash & t & = & t+\delta & \text{(by A6)} \\
 & & = & t+t+t' & \text{(by assumption)} \\
 & & = & t+t' & \text{(by A3)} \\
 & & = & \delta & \text{(by assumption).}
\end{array}
$$

□

We now give a result expressing some useful properties of basic guards, in which the axiom G4 is not used. Note clause (v), which expresses that the sequential composition is commutative for basic guards.

Lemma 2.5. *Let $\phi,\psi \in G$. The following identities are derivable in* BPA^3_G*:*

(i) $\neg\delta=\epsilon$,
(ii) $\neg\epsilon=\delta$,
(iii) $\neg\neg\phi=\phi$,
(iv) $\phi\psi\neg\phi=\delta$,
(v) $\phi\psi=\psi\phi$.

Proof. In the proofs of (i) and (ii) the axiom G3 is also not used.

$$
\begin{array}{llllll}
(i) & \neg\delta & = & \delta+\neg\delta & (ii) \quad \neg\epsilon & = \epsilon\cdot\neg\epsilon \\
 & & = & \epsilon. & & = \delta.
\end{array}
$$

In the proof of (iv) we use $t+t'=\delta \Longrightarrow t=\delta$ (see Example 2.4). In (v) we use $\neg\phi\psi\phi=\delta$, which is a direct consequence of (iii) and (iv).

$$
\begin{array}{lll}
(iii) & \neg\neg\phi & = (\phi + \neg\phi)\neg\neg\phi \\
 & & = \phi\neg\neg\phi + \delta \\
 & & = \phi\neg\neg\phi + \phi\neg\phi \\
 & & = \phi(\neg\neg\phi + \neg\phi) \\
 & & = \phi.
\end{array}
\qquad
\begin{array}{lll}
(iv) & \delta & = \phi\neg\phi \\
 & & = \phi(\psi + \neg\psi)\neg\phi \\
 & & = \phi(\psi\neg\phi + \neg\psi\neg\phi) \\
 & & = \phi\psi\neg\phi + \phi\neg\psi\neg\phi \\
 & & \Longrightarrow \phi\psi\neg\phi = \delta.
\end{array}
$$

$$
\begin{array}{lll}
(v) & \phi\psi & = \phi\psi(\phi + \neg\phi) \\
 & & = \phi\psi\phi + \phi\psi\neg\phi \\
 & & = \phi\psi\phi + \neg\phi\psi\phi \\
 & & = (\phi + \neg\phi)\psi\phi \\
 & & = \psi\phi.
\end{array}
$$

□

Up till now we only defined 'atomic' and 'basic' guards.

Definition 2.6. A *guard* α over $\Sigma(\mathrm{BPA}_G)$ has the following syntax

$$\alpha ::= \alpha + \alpha \mid \alpha \cdot \alpha \mid \phi \in G.$$ □

Let the symbols $\alpha, \beta, \ldots$ range over guards. On guards the operators $+$ and $\cdot$ correspond to the Boolean operators $\vee$ and $\wedge$, respectively. Let $\phi, \psi \in G$, then the guard $\phi + \psi$ holds in a data-state s whenever ϕ or ψ holds in s. The guard $\phi\psi$ holds in s iff both ϕ and ψ hold in s. In order to have the Boolean operator $\neg$ on guards, we introduce the *abbreviations*

$$
\begin{array}{ll}
\neg(\alpha\beta) & \text{for } \neg\alpha + \neg\beta, \\
\neg(\alpha + \beta) & \text{for } \neg\alpha\neg\beta.
\end{array}
$$

It is not hard to prove that all identities on *basic* guards that are derivable in BPA_G^3 (or BPA_G^4), are derivable in BPA_G^3 (BPA_G^4, respectively) for all guards:

Theorem 2.7. *Let α be a guard over $\Sigma(\mathrm{BPA}_G)$, then the following identities are derivable in BPA_G^3 (cf. G1 – G3):*

(*i*) $\alpha \cdot \neg\alpha = \delta$,
(*ii*) $\alpha + \neg\alpha = \epsilon$,
(*iii*) $\alpha(x + y) = \alpha \cdot x + \alpha \cdot y$.

The following identity is derivable in BPA_G^4 (cf. G4):

(*iv*) $a(\alpha \cdot x + \neg\alpha \cdot y) \subseteq ax + ay$

where $a \in A$. □

Moreover, restricting the signature $\Sigma(\mathrm{BPA}_G)$ to terms without atomic actions, the axiom system BPA_G^3 constitutes a Boolean algebra. According to [Sio64], the following five equations form an equational basis for a Boolean algebra $(G_{at}, +, \cdot, \neg)$:

(B1) $\alpha\beta = \beta\alpha$
(B2) $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
(B3) $\alpha + \beta\neg\beta = \alpha$
(B4) $\alpha(\beta + \neg\beta) = \alpha$
(B5) $\alpha + (\beta + \neg\beta) = \beta + \neg\beta$.

The only equation here that does not immediately follow from BPA_G^3 is B5:

$$\begin{aligned}\alpha + (\beta + \neg\beta) &= \alpha + \epsilon \\ &= \alpha + (\alpha + \neg\alpha) \\ &= (\alpha + \alpha) + \neg\alpha \\ &= \alpha + \neg\alpha \\ &= \epsilon \\ &= \beta + \neg\beta.\end{aligned}$$

2.2. Operational Semantics and Soundness

In process algebra closed process terms are often related to *(labelled) transition systems*, modelling their possible behaviour.

Definition 2.8. A *labelled transition system* $\mathscr{A}$ is a tuple $\langle S_{\mathscr{A}}, A_{\mathscr{A}}, \longrightarrow_{\mathscr{A}}, s_{\mathscr{A}}\rangle$ where

- $S_{\mathscr{A}}$ is a set of *states*,
- $A_{\mathscr{A}}$ is a set of *labels*,
- $\longrightarrow_{\mathscr{A}} \subseteq S_{\mathscr{A}} \times A_{\mathscr{A}} \times S_{\mathscr{A}}$ is the *transition relation*, and
- $s_{\mathscr{A}} \in S_{\mathscr{A}}$ is the *initial state*.

Elements $(s, a, t) \in \longrightarrow_{\mathscr{A}}$ are generally written as $s \xrightarrow{a} t$. □

Contrary to the traditional approach in process algebra, we provide an operational semantics that is based on data-state transformations and the interpretation of guards. The operational meaning of a process term is defined by a transition system, where the states of the transition system are *configurations*, i.e. pairs of a process term and a data-state.

We adopt an abstract view and assume that data-states are given by a set S. Atomic actions are considered as non-deterministic data-state *transformers*. This is modelled by a function *effect* that, given some atomic action a and a data-state s, returns the data-states which may result from the execution of a in s (see also [BKT85, BaB88]; in [BaB88] the *state operator* is introduced which provides an alternative way to handle processes operating on data-states). We demand that the function *effect* never returns the empty set, ensuring that an atomic action can always be executed. We use guards to prevent actions from happening in certain data-states. Finally the interpretation of guards is given by a predicate *test* that determines whether an atomic guard holds in some data-state.

Definition 2.9. A *data environment* $\mathscr{S}$ over a set A of atomic actions and a set G_{at} of atomic guards is a triple $\langle S, \mathit{effect}, \mathit{test}\rangle$ where

- S is a non-empty set of data-states,
- $\mathit{effect} : S \times A \to 2^S \setminus \{\emptyset\}$ defines the data-state transformations associated with atomic actions,
- $\mathit{test} \subseteq G_{at} \times S$ defines the interpretation of atomic guards. □

Observe that the function *effect* possibly introduces non-determinism in data-state transformations. Whenever $\mathit{test}(\phi, s)$ holds, this denotes that in data-state s the atomic guard ϕ may be passed. In this case we say that ϕ *holds* in s. In order to interpret basic guards, we extend the predicate *test* in the obvious way.

Definition 2.10. Let $\langle S, \mathit{effect}, \mathit{test}\rangle$ be some data environment. We extend the domain of *test* to $G \times S$ as follows:

- For all $s \in S$: $\mathit{test}(\epsilon, s)$ holds and $\mathit{test}(\delta, s)$ does not hold,

$$a \in A \quad (a,s) \xrightarrow{a} (\epsilon, s') \quad \text{if } s' \in \mathit{effect}(a,s)$$

$$\phi \in G \quad (\phi,s) \xrightarrow{\surd} (\delta,s) \quad \text{if } \mathit{test}(\phi,s)$$

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

$$\cdot \quad \frac{(x,s) \xrightarrow{a} (x',s')}{(xy,s) \xrightarrow{a} (x'y,s')} \; a \neq \surd \qquad \frac{(x,s) \xrightarrow{\surd} (x',s') \quad (y,s) \xrightarrow{a} (y',s'')}{(xy,s) \xrightarrow{a} (y',s'')}$$

Fig. 3. Transition rules for $\Sigma(\mathrm{BPA}_G)$ ($a \in A_{\surd}$, $\phi \in G$).

- For all $s \in S$ and $\phi \in G$: $\mathit{test}(\neg\phi, s)$ holds iff $\mathit{test}(\phi, s)$ does not hold. □

Let $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test} \rangle$ be some data environment over A and G_{at}. We give an operational semantics in the style of Plotkin [Plo81]. The behaviour of a process p with some initial data-state $s \in S$ starts in the *configuration* (p,s):

Definition 2.11. Let Σ be some signature and S a set of data-states. A *configuration* (p,s) over (Σ, S) is a pair containing a closed term p over Σ and a data-state $s \in S$. The set of all configurations over (Σ, S) is denoted by $C(\Sigma, S)$. □

Let $\surd \notin A$ be a special symbol which we use to represent successful termination, and

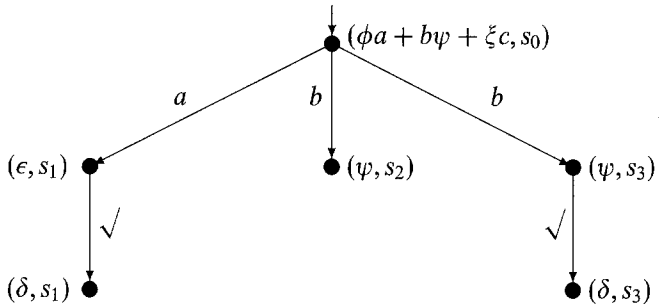
$$A_{\surd} \stackrel{\text{def}}{=} A \cup \{\surd\}.$$

The rules in Fig. 3, where the label a ranges over $A_{\surd}$ and ϕ over G, determine the transition relation $\longrightarrow_{\Sigma(\mathrm{BPA}_G),\mathscr{S}}$ that contains exactly all derivable transitions between the configurations over $(\Sigma(\mathrm{BPA}_G), S)$. The idea is that for $a \in A$, the transition $(p,s) \xrightarrow{a} (p',s')$ expresses that by executing a, the process p in data-state s can evolve into p' in data-state s'. In this case we have $s' \in \mathit{effect}(a,s)$ and the configuration (p',s') represents what remains to be executed. The transition $(p,s) \xrightarrow{\surd} (p',s')$ expresses that the process p in data-state s can terminate successfully. A basic guard ϕ can terminate successfully in data-state s if $\mathit{test}(\phi,s)$ holds, which is denoted by the transition $(\phi,s) \xrightarrow{\surd} (\delta,s)$ in Fig. 3. The configuration (δ,s) has no outgoing transitions, which expresses that no further activity is possible ('inaction' or 'deadlock').

In the case of BPA_G we define

$$\mathscr{A}(p,s) \stackrel{\text{def}}{=} \langle C(\Sigma(\mathrm{BPA}_G), S),\ A_{\surd},\ \longrightarrow_{\Sigma(\mathrm{BPA}_G),\mathscr{S}},\ (p,s) \rangle.$$

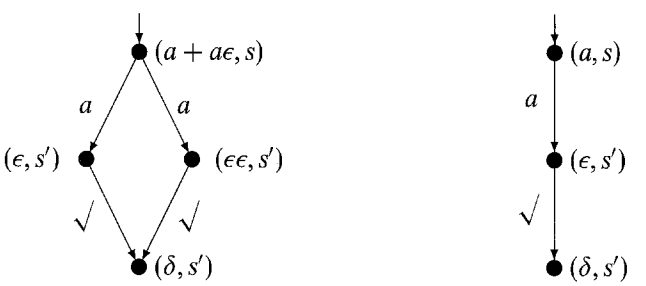
Example 2.12. Consider the data environment $\langle \{s_0, s_1, s_2, s_3\}, \mathit{effect}, \mathit{test} \rangle$ and the following partially depicted transition system $\mathscr{A}(\phi a + b\psi + \xi c, s_0)$ where the initial state is marked with a little arrow:



The (implicit) information about the function *effect* and the predicate *test* present in this transition system tells us that in this data environment apparently

$$effect(a, s_0) = \{s_1\} \text{ and } effect(b, s_0) = \{s_2, s_3\}$$
$$test(\phi, s_0), test(\neg\xi, s_0), test(\neg\psi, s_2) \text{ and } test(\psi, s_3). \qquad \square$$

Consider the following (partially depicted) transition systems $\mathscr{A}(a + a\epsilon, s)$ and $\mathscr{A}(a, s)$ over some data environment satisfying $effect(a, s) = \{s'\}$.



Observe that the transition system $\mathscr{A}(a + a\epsilon, s)$ is shaped as two transition systems for $\mathscr{A}(a, s)$. With respect to *operational* behaviour it does not matter whether the a-summand or the $a\epsilon$-summand is executed. Therefore we would like to consider both transition systems as equivalent. This can be achieved by identifying *bisimilar* configurations (see [Par81]), as bisimilarity is the coarsest equivalence that respects the operational characteristics of a transition system [Vaa89]. Following the traditional approach in semantics based on data-state transformations, processes with different data-states in their configurations are not considered as equivalent (see e.g. [Man74]). Therefore we adapt the standard notion of bisimilarity in the following way:

Definition 2.13. Let Σ be a signature, $\mathscr{S}$ a data environment with data-state space S and $\longrightarrow_{\Sigma,\mathscr{S}}$ a transition relation over $C(\Sigma, S)$.

- A binary relation $R \subseteq C(\Sigma, S) \times C(\Sigma, S)$ is an *$\mathscr{S}$-bisimulation* iff R satisfies the *transfer property*, i.e. for all $(p, s), (q, s) \in C(\Sigma, S)$ with $(p, s)R(q, s)$:

  1. Whenever $(p, s) \xrightarrow{a}_{\Sigma,\mathscr{S}} (p', s')$ for some a and (p', s'), then, for some q', also $(q, s) \xrightarrow{a}_{\Sigma,\mathscr{S}} (q', s')$ and $(p', s')R(q', s')$,

2. Conversely, whenever $(q,s) \xrightarrow{a}_{\Sigma,\mathscr{S}} (q',s')$ for some a and (q',s'), then, for some p', also $(p,s) \xrightarrow{a}_{\Sigma,\mathscr{S}} (p',s')$ and $(p',s')R(q',s')$.

- A configuration $(p,s) \in C(\Sigma,S)$ is *$\mathscr{S}$-bisimilar* with a configuration $(q,s') \in C(\Sigma,S)$, notation

  $$(p,s) \leftrightarrows_{\mathscr{S}} (q,s')$$

  iff $s = s'$ and there is an $\mathscr{S}$-bisimulation containing the pair $((p,s),(q,s'))$ (note the equality of the data-states!).
- A transition system $\mathscr{A}(p,s) = \langle C(\Sigma,S), A_{\surd}, \longrightarrow_{\Sigma,\mathscr{S}}, (p,s)\rangle$ is *$\mathscr{S}$-bisimilar* with a transition system $\mathscr{A}(q,s') = \langle C(\Sigma,S), A_{\surd}, \longrightarrow_{\Sigma,\mathscr{S}}, (q,s')\rangle$, notation

  $$\mathscr{A}(p,s) \leftrightarrows_{\mathscr{S}} \mathscr{A}(q,s')$$

  iff $(p,s) \leftrightarrows_{\mathscr{S}} (q,s')$.
- Two closed terms p,q over Σ are *$\mathscr{S}$-bisimilar*, notation

  $$p \leftrightarrows_{\mathscr{S}} q$$

  iff $\mathscr{A}(p,s) \leftrightarrows_{\mathscr{S}} \mathscr{A}(q,s)$ for all $s \in S$. □

We introduced the symbol $\leftrightarrows_{\mathscr{S}}$ instead of the more consistent symbol $\leftrightarrows_{\Sigma,\mathscr{S}}$ to avoid lengthy notation. We take care that Σ is known from the context when we use $\leftrightarrows_{\mathscr{S}}$. Note that the symbol $\leftrightarrows_{\mathscr{S}}$ is also overloaded in another way. It denotes either a relation between configurations, between transition systems or between closed terms.

Lemma 2.14. *For any data environment $\mathscr{S}$ the relation $\leftrightarrows_{\mathscr{S}}$ between closed terms over $\Sigma(\mathrm{BPA}_G)$ is a congruence with respect to the operators of $\Sigma(\mathrm{BPA}_G)$.*

Proof. Standard. □

Moreover, it is not hard to prove that BPA_G is a *sound* axiom system with respect to $\mathscr{S}$-bisimulation equivalence for any data environment $\mathscr{S}$.

Theorem 2.15. *Let p,q be closed terms over $\Sigma(\mathrm{BPA}_G)$. If $\mathrm{BPA}_G \vdash p = q$ then $p \leftrightarrows_{\mathscr{S}} q$ for any data environment $\mathscr{S}$.*

Proof. The relation $\leftrightarrows_{\mathscr{S}}$ between the closed terms over $\Sigma(\mathrm{BPA}_G)$ is a congruence and hence respects the inference rules for equality. We have to show that all axioms are valid. As an example we prove this for G4.

Assume that $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$, $a \in A$, $\phi \in G$ and p,q are closed process terms over $\Sigma(\mathrm{BPA}_G)$. We have to show $(ap + aq + a(\phi p + \neg\phi q), s) \leftrightarrows_{\mathscr{S}} (ap + aq, s)$ for all $s \in S$. We define the relation R as follows:

$$\begin{aligned} R \stackrel{\text{def}}{=} Id \;&\cup\; \{((ap + aq + a(\phi p + \neg\phi q), s), (ap + aq, s)) \mid s \in S\} \\ &\cup\; \{((\epsilon(\phi p + \neg\phi q), s), (\epsilon p, s)) \mid s \in S \text{ and } \mathit{test}(\phi, s)\} \\ &\cup\; \{((\epsilon(\phi p + \neg\phi q), s), (\epsilon q, s)) \mid s \in S \text{ and } \mathit{test}(\neg\phi, s)\} \end{aligned}$$

where Id is the identity relation on $C(\Sigma(\mathrm{BPA}_G), S)$. In the standard way it follows that R is an $\mathscr{S}$-bisimulation satisfying

$$(ap + aq + a(\phi p + \neg\phi q), s) R (ap + aq, s)$$

for all $s \in S$. □

2.3. Completeness

In this section we show that the axiom system BPA_G^4 is *complete* in the following general sense. Let p, q be closed terms over $\Sigma(\mathrm{BPA}_G)$. If for *all* data environments $\mathscr{S}$ we have $p \leftrightarrows_{\mathscr{S}} q$, then $\mathrm{BPA}_G^4 \vdash p = q$. So completeness says that the axioms of BPA_G^4 are sufficiently strong to prove all identities between closed terms over $\Sigma(\mathrm{BPA}_G)$ that are valid in all data environments, and that $\mathscr{S}$-bisimilarity between terms that cannot be proved in this way depends on the particular ingredients of $\mathscr{S}$. If for example the atomic guard ϕ holds in all the data-states of some data environment $\mathscr{S}$, we have $\phi \leftrightarrows_{\mathscr{S}} \epsilon$. Of course we cannot derive $\mathrm{BPA}_G^4 \vdash \phi = \epsilon$, as ϕ is not interpreted as ϵ in all possible data environments. Proving identities that are dependent on a particular data environment is the topic of the next section. Some of the results proved in this section concern the axiom system BPA_G^3 (the system containing all axioms of BPA_G^4, except G4). These can be reused in the completeness theorems on parallel processes in section 4.

All completeness results in this paper are proved according to the following strategy: define classes of *basic terms* such that

1. Any closed term can be proved equal to a unique basic term, and
2. If two basic terms are not provably equal (i.e., syntactically different), then one can find a data environment in which they are not bisimilar.

We introduce *reference* sets in order to define suitable basic terms.

Definition 2.16. *(Reference)*

1. Let p be a closed term over $\Sigma(\mathrm{BPA}_G)$. By $Ref(p)$ we denote the set of atomic guards to which p makes *reference*:

$$Ref(p) \stackrel{\text{def}}{=} \{\phi \in G_{at} \mid \phi \text{ occurs (possibly negated) in } p\}.$$

2. Any non-empty, finite subset of G_{at} is called a *reference set*. We use symbols $R, R_1, R_2, \ldots$ to denote reference sets. For technical convenience we assume that the elements in reference sets are ordered.
3. Let $R = \{\phi_0, \ldots, \phi_n\}$ be some (ordered) reference set. A 'sequential' expression $\psi_0 \cdots \psi_n$ is called a *complete guard sequence over* R iff for $i = 0, \ldots, n$ we have that either $\psi_i \equiv \phi_i$ or $\psi_i \equiv \neg\phi_i$. Such sequences are abbreviated by symbols $\vec{\phi}, \vec{\psi}, \ldots$ and we write R^{co} for the set of all complete guard sequences over R.

□

We demonstrate two properties of reference sets by a simple observation and a lemma. Let R be some reference set. First observe that if $\vec{\phi}, \vec{\psi} \in R^{co}$, then

$$\mathrm{BPA}_G^3 \vdash \vec{\phi} \cdot \vec{\psi} = \begin{cases} \vec{\phi} & \text{if } \vec{\phi} \equiv \vec{\psi}, \\ \delta & \text{otherwise.} \end{cases}$$

This observation holds because R is ordered: if $\{\phi, \psi\}$ is an *unordered* reference set, we have by Lemma 2.5. for instance $\mathrm{BPA}_G^3 \vdash (\phi\psi)(\psi\phi) = \phi\psi$.

In order to denote terms in a convenient way we further use the Σ-notation: let I be some finite index set, then

$$\sum_{i \in I} t_i \stackrel{\text{def}}{=} \begin{cases} \delta & \text{if } I = \emptyset, \\ t_{i_0} + \cdots + t_{i_n} & \text{if } I = \{i_0, \ldots, i_n\}. \end{cases}$$

Note that due to the axioms A1 and A2 the actual enumeration of the terms t_{i_j} does not matter.

The following lemma establishes a second useful property.

Lemma 2.17. *For any t over $\Sigma(\mathrm{BPA}_G)$ and reference set R we have*

$$\mathrm{BPA}_G^3 \vdash t = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} t.$$

Proof. By induction on the cardinality of R:

$R = \{\phi\}$. In this case $t = \epsilon t = (\phi + \neg\phi)t = \phi t + \neg\phi t = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} t$.

$R = \{\phi_0, \ldots, \phi_{k+1}\}$. Let $R_1 \stackrel{\text{def}}{=} R - \{\phi_0\}$. First applying the induction hypothesis we derive

$$\begin{aligned} t &= \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} t \\ &= (\phi_0 + \neg\phi_0) \cdot \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} t \\ &= \phi_0 \cdot \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} t + \neg\phi_0 \cdot \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} t \\ &= \sum_{\vec{\psi} \in R_1^{co}} \phi_0 \vec{\psi} t + \sum_{\vec{\psi} \in R_1^{co}} \neg\phi_0 \vec{\psi} t \\ &= \sum_{\vec{\phi} \in R^{co}} \vec{\phi} t. \end{aligned}$$

□

Using reference sets, we introduce the following two classes of basic terms over $\Sigma(\mathrm{BPA}_G)$.

Definition 2.18. Let R be some reference set.

1. A closed term p is called *G-basic over R* iff

$$p = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

where for each $\vec{\phi} \in R^{co}$ the term $q_{\vec{\phi}}$ is an A-basic term over R.

2. A closed term q is called *A-basic over R* iff

$$q = \sum_{i \in I} a_i p_i [+\epsilon]$$

where for each $i \in I$ it holds that $a_i \in A$ and the term p_i is a G-basic term over R. The notation $[+\epsilon]$ means that the occurrence of the summand ϵ is optional. □

We show that any closed term over $\Sigma(\mathrm{BPA}_G)$ is provably equal to a G-basic term over some reference set. The proof is split up in two parts. First, any closed term

can be proved equal to one that is in 'prefix normal form' (defined below), then we show that any term in prefix normal form is provably equal to a G-basic term.

Definition 2.19. A closed term p over $\Sigma(\mathrm{BPA}_G)$ is in *prefix normal form over* $\Sigma(\mathrm{BPA}_G)$ iff

$$p ::= \delta \mid \epsilon \mid \phi p \mid \neg\phi p \mid ap \mid p + p$$

with $\phi \in G_{at}$ and $a \in A$. □

The following lemma states that it is sufficient to consider terms that are in prefix normal form over $\Sigma(\mathrm{BPA}_G)$.

Lemma 2.20. *If p is a closed term over $\Sigma(\mathrm{BPA}_G)$, then there is a term p' in prefix normal form over $\Sigma(\mathrm{BPA}_G)$ such that $\mathrm{BPA}_G^3 \vdash p = p'$.*

Proof. By induction on the structure of closed terms. □

Lemma 2.21. *If p is a closed term over $\Sigma(\mathrm{BPA}_G)$ and R some reference set satisfying $R \supseteq Ref(p)$, then there is a G-basic term p' over R such that $\mathrm{BPA}_G^3 \vdash p = p'$.*

Proof. By Lemma 2.20. we may assume that p is in prefix normal form over $\Sigma(\mathrm{BPA}_G)$. We apply induction on the structure of such normal forms:

$p \equiv \delta$ **or** $p \equiv \epsilon$. By Lemma 2.17. we have

$$\delta = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}\delta \quad \text{and} \quad \epsilon = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}\epsilon, \text{ respectively,}$$

for any reference set R.

$p \equiv \phi q$. Let $R \supseteq Ref(p)$, then $R \supseteq Ref(q)$. By the induction hypothesis we have

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

with all the terms $q_{\vec{\phi}}$ A-basic over R. Let for each $\vec{\phi} \in R^{co}$

$$q'_{\vec{\phi}} \equiv \begin{cases} q_{\vec{\phi}} & \text{if } \phi \text{ occurs in } \vec{\phi}, \\ \delta & \text{otherwise,} \end{cases}$$

then

$$\sum_{\vec{\phi} \in R^{co}} \vec{\phi} q'_{\vec{\phi}}$$

is a G-basic term over R that is provably equal to p.

$p \equiv \neg\phi q$. Likewise.

$p \equiv aq$. Let $R \supseteq Ref(p)$, then $R \supseteq Ref(q)$. By the induction hypothesis we have

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

with all the terms $q_{\vec{\phi}}$ A-basic over R. By Lemma 2.17 we have $a = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} a$ and we can take

$$\sum_{\vec{\phi} \in R^{co}} \vec{\phi} a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

which clearly is a G-basic term over R provably equal to p.

$p \equiv q + r$. Let $R \supseteq Ref(p)$, then $R \supseteq Ref(q)$ and $R \supseteq Ref(r)$. By the induction hypothesis we have

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}} \text{ and } r = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q'_{\vec{\phi}}$$

with all the terms $q_{\vec{\phi}}, q'_{\vec{\phi}}$ A-basic over R. Hence

$$q + r = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}(q_{\vec{\phi}} + q'_{\vec{\phi}}).$$

Observe that the sum of two A-basic terms over R is provably equal to an A-basic term over R: change to one index-set or remove a δ-summand and replace double occurrences of ϵ-summands. So for each $\vec{\phi} \in R^{co}$ there is an A-basic term $q''_{\vec{\phi}}$ over R such that $q_{\vec{\phi}} + q'_{\vec{\phi}} = q''_{\vec{\phi}}$. Hence

$$\sum_{\vec{\phi} \in R^{co}} \vec{\phi} q''_{\vec{\phi}}$$

is a G-basic term over R provably equal to p. □

The syntax of an A-basic term is sufficiently strict to derive information about its (syntactic) structure from its operational behaviour. This information is formulated with help of the following syntactic relation on terms:

Definition 2.22. Let t_1, t_2 be terms over Σ. We call t_1 a *syntactic summand* of t_2, notation $t_1 \sqsubseteq t_2$, iff

1. $t_1 \not\equiv t + t'$ for any t, t' over Σ, and
2. $t_1 \equiv t_2$, or there are t, t' over Σ such that $t_2 \equiv t + t'$ and $t_1 \sqsubseteq t$ or $t_1 \sqsubseteq t'$.

□

So e.g. $x(y + y) + z + z$ has $x(y + y)$ and z as its only syntactic summands and $(x + y)z$ has no other syntactic summand than itself.

Lemma 2.23. *Let $\mathcal{S} = \langle S, effect, test \rangle$, and R be some reference set. For any A-basic term q over R the following properties hold:*

1. *If $\exists s \in S$ such that $(q, s) \stackrel{\surd}{\rightarrow} (r, s')$, then $\epsilon \sqsubseteq q$,*
2. *If $\exists s \in S$ such that $(q, s) \stackrel{a}{\rightarrow} (r, s')$ $(a \in A)$, then there is a G-basic term p over R such that $ap \sqsubseteq q$ and $\epsilon p \equiv r$.*

Proof. By using representations of the form

$$\sum_{i \in I} a_i p_i [+\epsilon]$$

and applying induction on the cardinality of I. □

We also need the following result, which is in fact a generalisation of the axiom G4.

Lemma 2.24. *(Saturation) Let R be some reference set. For any $a \in A$, terms $t_0, \ldots, t_n$ over $\Sigma(\mathrm{BPA}_G)$ and function $f : R^{co} \rightarrow \{t_0, \ldots, t_n\}$ we have*

$$\mathrm{BPA}_G^4 \vdash \sum_{i=0}^{n} at_i \trianglerighteq a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} \cdot f(\vec{\phi}).$$

Proof. By induction on the cardinality of R.

$R = \{\phi\}$. Then

$$a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} \cdot f(\vec{\phi}) = a(\phi t_j + \neg\phi t_k)$$

for some $j,k \in \{0,\ldots,n\}$. By the axiom G4 ($at_j + at_k \trianglerighteq a(\phi t_j + \neg\phi t_k)$) we derive

$$\sum_{i=0}^{n} at_i \trianglerighteq a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} \cdot f(\vec{\phi}).$$

$R = \{\phi_0,\ldots,\phi_{k+1}\}$. Let $f : R^{co} \to \{t_0,\ldots,t_n\}$ be given, and $R_1 \stackrel{\text{def}}{=} R - \{\phi_0\}$. Take $g^i : R_1^{co} \to \{t_0,\ldots,t_n\}$ $(i = 1,2)$ such that

$$g^1(\vec{\psi}) \stackrel{\text{def}}{=} f(\phi_0\vec{\psi}) \text{ and } g^2(\vec{\psi}) \stackrel{\text{def}}{=} f(\neg\phi_0\vec{\psi}).$$

First applying the induction hypothesis two times and then the axiom G4 we derive

$$\begin{aligned}
\sum_{i=0}^{n} at_i &\trianglerighteq a \cdot \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot g^1(\vec{\psi}) + a \cdot \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot g^2(\vec{\psi}) \\
&\trianglerighteq a\Big(\phi_0 \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot g^1(\vec{\psi}) + \neg\phi_0 \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot g^2(\vec{\psi})\Big) \\
&= a\Big(\sum_{\vec{\psi} \in R_1^{co}} \phi_0\vec{\psi} \cdot g^1(\vec{\psi}) + \sum_{\vec{\psi} \in R_1^{co}} \neg\phi_0\vec{\psi} \cdot g^2(\vec{\psi})\Big) \\
&= a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} \cdot f(\vec{\phi}).
\end{aligned}$$

□

The two previous results give us the means to prove a key lemma stating that whenever two G-basic terms over some reference set R do *not* obey certain provable characteristics, then we can find a data environment $\mathcal{S}$ such that $p \not\leftrightarrows_{\mathcal{S}} q$. Such a data environment is then defined in terms of R.

Definition 2.25. Let R be some reference set. We define the data environment $\mathcal{S}(R) = \langle R^{co}, \mathit{effect}, \mathit{test}\rangle$ by

$$\begin{aligned}
a \in A &\implies \mathit{effect}(a,\vec{\phi}) \stackrel{\text{def}}{=} R^{co}, \\
\phi \in G_{at} &\implies \mathit{test}(\phi,\vec{\phi}) \text{ iff } \phi \text{ occurs in } \vec{\phi}, \text{ or if } \phi \notin R.
\end{aligned}$$

□

The idea is that $\mathcal{S}(R)$ is sufficiently discriminating to distinguish any two G-basic terms over R that are not provably equal. We define the *depth* of a closed term over $\Sigma(\mathrm{BPA}_G)$ as the maximal number of consecutive atomic actions that can be performed. It plays a role as a criterion for induction in proofs.

Definition 2.26. The *depth* of a closed term p over $\Sigma(\mathrm{BPA}_G)$, written as $|p|$, is some element of $\mathbb{N}$, defined inductively as follows ($\phi \in G$ and $a \in A$):

$$|\phi| \stackrel{\text{def}}{=} 0,$$
$$|a| \stackrel{\text{def}}{=} 1,$$
$$|pq| \stackrel{\text{def}}{=} |p| + |q|,$$
$$|p + q| \stackrel{\text{def}}{=} max(|p|, |q|).$$

□

Lemma 2.27. *Let p_1, p_2 be G-basic terms over some reference set R. If there is a syntactic summand $\vec{\phi} q_1$ of p_1 such that for any A-basic term q' over R we have*

$$\mathrm{BPA}_G^4 \vdash q_1 = q' \implies \vec{\phi} q' \not\sqsubseteq p_2,$$

then $(p_1, \vec{\phi}) \not\leftrightarrows_{\mathscr{S}(R)} (p_2, \vec{\phi})$.

Proof. Apply induction on $|p_1| + |p_2|$. The case $|p_1| + |p_2| = 0$ is trivial, so let $|p_1| + |p_2| > 0$. By definition p_2 has a syntactic summand $\vec{\phi} q_2$ and by assumption $q_1 \neq q_2$. At least one of the following should hold:

1. $\epsilon \sqsubseteq q_1$ and $\epsilon \not\sqsubseteq q_2$,
2. $\epsilon \sqsubseteq q_2$ and $\epsilon \not\sqsubseteq q_1$,
3. $ar \sqsubseteq q_1$ and $ar \not\sqsubseteq q_2$ for some $a \in A$ and G-basic term r over R,
4. $ar \sqsubseteq q_2$ and $ar \not\sqsubseteq q_1$ for some $a \in A$ and G-basic term r over R.

If not, then $q_1 \subseteq q_2$ by 1 and 3, and $q_2 \subseteq q_1$ by 2 and 4, so $q_1 = q_2$, contradicting the assumption.

In cases 1 and 2 we have that for one of $(p_1, \vec{\phi})$, $(p_2, \vec{\phi})$ there is a derivable $\surd$-transition, whereas by Lemma 2.23. this is not the case for the other (for $\epsilon \not\sqsubseteq q_2 \implies \epsilon \not\sqsubseteq q_2$). Hence $(p_1, \vec{\phi}) \not\leftrightarrows_{\mathscr{S}(R)} (p_2, \vec{\phi})$. We only prove case 3 (the last case can be dealt with in a similar fashion):

either q_2 has no syntactic summand of the form ar'. Now $(p_1, \vec{\phi}) \not\leftrightarrows_{\mathscr{S}(R)} (p_2, \vec{\phi})$, for $(p_1, \vec{\phi})$ has an a-transition, whereas $(p_2, \vec{\phi})$ has no such transition by Lemma 2.23;

or q_2 has $n+1$ syntactic summands starting with a, say $ar_0, \ldots, ar_n$ with $r_0, \ldots, r_n$ G-basic terms over R. Now there is $\vec{\psi} t_{\vec{\psi}} \sqsubseteq r$ such that for all A-basic terms t' over R we have

$$t_{\vec{\psi}} = t' \implies \forall i \in \{0, \ldots, n\} \ \vec{\psi} t' \not\sqsubseteq r_i$$

If this were not the case, then there would be a function $f : R^{co} \to \{r_0, \ldots, r_n\}$ such that for any syntactic summand $\vec{\phi} t_{\vec{\phi}}$ of r there is a $t'_{\vec{\phi}}$ satisfying $t_{\vec{\phi}} = t'_{\vec{\phi}}$ and $\vec{\phi} t'_{\vec{\phi}} \sqsubseteq f(\vec{\phi})$. Using 'saturation' (see Lemma 2.24.) we derive

$$\begin{aligned}
\sum_{i=0}^{n} ar_i &\supseteq a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} \cdot f(\vec{\phi}) \\
&= a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} t'_{\vec{\phi}} \quad (\vec{\psi} \neq \vec{\phi} \implies \vec{\psi} \cdot \vec{\phi} = \delta) \\
&= a \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} t_{\vec{\phi}} \\
&= ar.
\end{aligned}$$

We conclude $ar \subseteq q_2$, which is a contradiction in this case. By the induction hypothesis we have for $i = 0,\ldots,n$ that $(r,\vec{\psi}) \not\leftrightarrows_{\mathcal{S}(R)} (r_i,\vec{\psi})$. Now $(p_1,\vec{\phi}) \xrightarrow{a} (\epsilon r,\vec{\psi})$ is a derivable transition that can only be mimicked from $(p_2,\vec{\phi})$ by a transition $(p_2,\vec{\phi}) \xrightarrow{a} (\epsilon r_i,\vec{\psi})$ for some i. As $(\epsilon r,\vec{\psi}) \leftrightarrows_{\mathcal{S}(R)} (r,\vec{\psi})$ and $(\epsilon r_i,\vec{\psi}) \leftrightarrows_{\mathcal{S}(R)} (r_i,\vec{\psi})$ it follows that $(p_1,\vec{\phi}) \not\leftrightarrows_{\mathcal{S}(R)} (p_2,\vec{\phi})$. □

With this key lemma on the specific data environment $\mathcal{S}(R)$, the main result of this section follows easily.

Theorem 2.28. *(Completeness) Let r_1, r_2 be closed terms over $\Sigma(\mathrm{BPA}_G)$. If $r_1 \leftrightarrows_{\mathcal{S}} r_2$ for all data environments $\mathcal{S}$, then $\mathrm{BPA}_G^4 \vdash r_1 = r_2$.*

Proof. We prove the theorem by contraposition. Suppose $r_1 \neq r_2$. We have to find a data environment $\mathcal{S}$ such that $r_1 \not\leftrightarrows_{\mathcal{S}} r_2$.

According to Lemma 2.21 there are G-basic terms p_1, p_2 over some reference set $R \supseteq Ref(r_1) \cup Ref(r_2)$ such that $\mathrm{BPA}_G^4 \vdash r_i = p_i$ $(i = 1,2)$. By soundness (see Theorem 2.15) we have $r_i \leftrightarrows_{\mathcal{S}} p_i$ for all $\mathcal{S}$. Because $\mathrm{BPA}_G^4 \not\vdash p_1 = p_2$, either p_1 has a syntactic summand $\vec{\phi}q$ such that for any A-basic term q' over R we have $\mathrm{BPA}_G^4 \vdash q = q' \Longrightarrow \vec{\phi}q' \not\sqsubseteq p_2$, or vice versa: if this were not the case, then $p_1 = \sum_{\vec{\phi} \in R^{\omega}} \vec{\phi} q_{\vec{\phi}} = \sum_{\vec{\phi} \in R^{\omega}} \vec{\phi} q'_{\vec{\phi}} = p_2$. This means that the previous Lemma 2.27 can be applied, and hence $(p_1,\vec{\phi}) \not\leftrightarrows_{\mathcal{S}(R)} (p_2,\vec{\phi})$. As $\leftrightarrows_{\mathcal{S}}$ is an equivalence relation we conclude $(r_1,\vec{\phi}) \not\leftrightarrows_{\mathcal{S}(R)} (r_2,\vec{\phi})$ and therefore $r_1 \not\leftrightarrows_{\mathcal{S}(R)} r_2$, which finishes our proof. □

2.4. Specifying Processes Recursively

We extend our process language with a mechanism that enables us to specify infinite processes by recursive equations.

Definition 2.29. A *recursive specification* $E = \{x = t_x \mid x \in V_E\}$ *over* a signature Σ is a set of equations where V_E is a (possibly infinite) set of (indexed) variables and t_x a term over Σ such that its variables (if any) are in V_E. □

A *solution* of a recursive specification $E = \{x = t_x \mid x \in V_E\}$ is an interpretation of the variables in V_E as processes, such that the equations of E are satisfied. For instance the recursive specification $\{x = x\}$ has any process as a solution for x and $\{x = ax\}$ has the infinite process "a^{ω}" as a solution for x. We introduce the following syntactical restriction on recursive specifications.

Definition 2.30. Let t be a term over a signature Σ. An occurrence of a variable x in t is *guarded* iff t has a subterm of the form $a \cdot M$ with $a \in A \cup \{\delta\}$, and this x occurs in M. Let $E = \{x = t_x \mid x \in V_E\}$ be a recursive specification over Σ. We say that E is a *guarded* specification iff all occurrences of variables in the terms t_x are guarded. □

The property "guarded" of a recursive specification has nothing to do with the "guards" that form the main subject of this paper. It is however established terminology, and therefore we respect it. Now the signature Σ_{REC}, in which we are interested, is defined by:

Definition 2.31. The signature Σ_{REC} is obtained by extending Σ in the following way: for each guarded specification $E = \{x = t_x \mid x \in V_E\}$ over Σ a set of

(REC)	$<x \mid E> = <t_x \mid E>$	if $x = t_x \in E$ and E guarded
(RSP)	$\dfrac{E(\vec{p_x})}{p_x = <x \mid E>}$	if $x \in V_E$ and E guarded

Fig. 4. Axioms for guarded recursive specifications.

constants $\{<x \mid E> \mid x \in V_E\}$ is added, where the construct $<x \mid E>$ denotes the x-component of a solution of E. □

We introduce some more notations: let $E = \{x = t_x \mid x \in V_E\}$ be a guarded specification over Σ, and t some term over Σ_{REC}. Then $<t \mid E>$ denotes the term in which each occurrence of a variable $x \in V_E$ in t is replaced by $<x \mid E>$, e.g. $<aax \mid \{x = ax\}>$ denotes the term $aa<x \mid \{x = ax\}>$. If we assume that the variables in recursive specifications are chosen uniquely, there is no need to repeat E in each occurrence of $<x \mid E>$. Variables reserved in this way are called *formal variables* and denoted by capital letters. We adopt the convention that $<x \mid E>$ can be abbreviated by X once E is declared. As an example consider the guarded recursive specification $\{x = ax\}$: the closed term aaX abbreviates $aa<x \mid \{x = ax\}>$.

For the new Σ-constants of the form $<x \mid E>$ there are two axioms in Fig. 4. In these axioms the letter E ranges over guarded specifications. The axiom REC states that the constant $<x \mid E>$ $(x \in V_E)$ is a solution for the x-component of E, so expresses that each guarded recursive specification has *at least* one solution for each of its (bounded) variables. The conditional axiom RSP (Recursive Specification Principle) expresses that E has *at most* one solution for each of its variables: whenever we can find processes p_x $(x \in V_E)$ satisfying the equations of E, notation $E(\vec{p_x})$, then $p_x = <x \mid E>$. This axiom was first formulated in [BeK86] and the format adopted here stems from [vGV89]. Finally, a convention is to denote a particular recursive specification right away by all its REC instances (see the following example).

Example 2.32. Consider the guarded specifications $E = \{x = ax\}$ and $E' = \{y = ayb\}$ over $\Sigma(\mathrm{BPA}_G)$. So by the convention just introduced, E can be represented by $X = aX$ and E' by $Y = aYb$. With REC and RSP (and the congruence properties of $=$) we prove $\mathrm{BPA}_G^4 + \mathrm{REC} + \mathrm{RSP} \vdash X = Y$ in the following way:

$$Xb \overset{\mathrm{REC}}{=} aXb \quad \overset{\mathrm{RSP}}{\Longrightarrow} \quad Xb = X, \tag{1}$$

and secondly

$$Xb \overset{\mathrm{REC}}{=} aXb \overset{(1)}{=} aXbb \quad \overset{\mathrm{RSP}}{\Longrightarrow} \quad Xb = Y.$$

Hence $\mathrm{BPA}_G^4 + \mathrm{REC} + \mathrm{RSP} \vdash X = Y$. □

In order to associate transition systems with closed terms over Σ_{REC} by guarded specifications, we define in the case of $E = \{x = t_x \mid x \in V_E\}$ being a guarded recursive specification over some signature Σ the general transition rule in Fig. 5. Observe that this rule immediately implies the soundness of REC.

In the case of $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ we define:

$$\mathscr{A}(p, s) \overset{\mathrm{def}}{=} \langle C(\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, S)\,,\, A_{\surd}\,,\, \longrightarrow_{\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, \mathscr{S}}\,,\, (p, s)\rangle.$$

$$\textit{recursion} \quad \frac{(<t_x \mid E>, s) \xrightarrow{a} (y, s')}{(<x \mid E>, s) \xrightarrow{a} (y, s')} \quad \text{if } x = t_x \in E$$

Fig. 5. Transition rule for guarded recursive specifications ($a \in A_{\surd}$).

We state without proof that $\mathrm{BPA}_G^4 + \mathrm{REC} + \mathrm{RSP}$ is sound (the interested reader is referred to [BaW90]).

Theorem 2.33. *Let p, q be closed terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$. If $\mathrm{BPA}_G^4 + \mathrm{REC} + \mathrm{RSP} \vdash p = q$, then $p \leftrightarrows_{\mathscr{S}} q$ for any data environment $\mathscr{S}$.* □

Note that RSP is not valid in the case of *un*guarded recursion: the unguarded recursive specification $\{x = x\}$ would otherwise lead to provable equality between all terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$.

Example 2.34. We conclude this section by an example in the style of the introductory one on the **if - then - else - fi** construct with which we started out: given an atomic action $[x := x + t]$ and an atomic guard $\langle x = t \rangle$ (where t ranges over integer expressions possibly containing program variable x), consider the program

while $x \neq t$ **do** $[x := x + t]$ **od**.

This program can be recursively specified over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ by

$$X \text{ where } X = \neg\langle x = t \rangle \cdot [x := x + t] \cdot X + \langle x = t \rangle$$

or equivalently by

$$Y \cdot \langle x = t \rangle \text{ where } Y = \neg\langle x = t \rangle \cdot [x := x + t] \cdot Y + \epsilon$$

(as $\mathrm{BPA}_G + \mathrm{REC} + \mathrm{RSP} \vdash X = Y \cdot \langle x = t \rangle$). The idea is that data-states are integer valuations in this case, and indeed X terminates in a data-state where $\langle x = t \rangle$ holds, and performs $[x := x + t]$ otherwise. □

3. BPA with Guards in a Specific Data Environment

Up until now we have studied basic process algebra with guards with respect to the general class of data environments. But often one wants to consider a data environment that is already determined, for instance in the case where atomic actions are assignments and guards are Boolean expressions. Therefore we now investigate bisimulation semantics for basic process algebra with guards in a *specific* data environment. For any data environment satisfying some expressibility constraints we present a complete axiomatisation by adding some new axioms to the system BPA_G^4. Finally, we show by an example how we can prove the (partial) correctness of a small imperative program in process algebra.

3.1. Axioms and Weakest Preconditions

Let A be a set of atomic actions and G_{at} a set of atomic guards. In this section we fix a data environment $\mathscr{S} = \langle S, \textit{effect}, \textit{test} \rangle$ over A and G_{at}. Now the axiom

(A1)	$x + y = y + x$	(G1)	$\phi \cdot \neg\phi = \delta$
(A2)	$x + (y + z) = (x + y) + z$	(G2)	$\phi + \neg\phi = \epsilon$
(A3)	$x + x = x$	(G3)	$\phi(x + y) = \phi x + \phi y$
(A4)	$(x + y)z = xz + yz$	(G4)	$a(\phi x + \neg\phi y) \subseteq ax + ay$
(A5)	$(xy)z = x(yz)$		
(A6)	$x + \delta = x$	(SI)	$\phi_0 \cdots \phi_n = \delta$
(A7)	$\delta x = \delta$		if $\forall s \in S\ \exists i \leq n \,.\, test(\neg\phi_i, s)$
(A8)	$\epsilon x = x$	(WPC1)	$wp(a, \phi)a\phi = wp(a, \phi)a$
(A9)	$x\epsilon = x$	(WPC2)	$\neg wp(a, \phi)a\neg\phi = \neg wp(a, \phi)a$

Fig. 6. The axioms of $\mathrm{BPA}_G(\mathscr{S})$ where $\phi, \phi_i \in G$ and $a \in A$.

system BPA_G^4 need not be complete. Assume for instance that two basic guards ϕ and ψ both satisfy $test(\phi, s) \iff test(\psi, s)$ for all $s \in S$, i.e. ϕ and ψ behave the same in all data-states. Obviously we have that $\phi \leftrightarrows_{\mathscr{S}} \psi$, but this cannot be shown using BPA_G^4 because in general $\phi \not\leftrightarrows_{\mathscr{S}} \psi$. For another example, assume that the process a, starting in a data-state where ϕ holds, always ends in a data-state where ψ holds. In this case $\phi a \leftrightarrows_{\mathscr{S}} \phi a \psi$. Again this cannot be proved in BPA_G^4.

In Fig. 6 we present the axiom system $\mathrm{BPA}_G(\mathscr{S})$ by which we can prove these identities. It contains the axioms of BPA_G^4 and three new axioms depending on $\mathscr{S}$ (this explains the $\mathscr{S}$ in $\mathrm{BPA}_G(\mathscr{S})$).

The axiom SI (Sequence is Inaction) expresses that if a sequence of basic guards fails in each data-state, then it equals δ. Note that SI implies G1. The equivalence $\phi \leftrightarrows_{\mathscr{S}} \psi$ mentioned above implies that $\phi\neg\psi = \delta$ and $\neg\phi\psi = \delta$ are in this case instances of SI. We can prove $\mathrm{BPA}_G(\mathscr{S}) \vdash \phi = \psi$ as follows:

$$\begin{aligned}
\phi &= \phi(\psi + \neg\psi) \\
&= \phi\psi + \phi\neg\psi \\
&= \phi\psi \\
&= \phi\psi + \neg\phi\psi \\
&= (\phi + \neg\phi)\psi \\
&= \psi.
\end{aligned}$$

In the axioms WPC1 and WPC2 (Weakest Preconditions under some Constraints) the expression $wp(a, \phi)$ represents the basic guard that is the *weakest precondition* of an atomic action a and an atomic guard ϕ. Weakest preconditions are semantically defined as follows:

Definition 3.1. Let A be a set of atomic actions, G_{at} a set of atomic guards and $\mathscr{S} = \langle S, effect, test \rangle$ be a data environment over A and G_{at}. A *weakest precondition* of an atomic action $a \in A$ and an atomic guard $\phi \in G_{at}$ is a basic guard $\psi \in G$ satisfying for all $s \in S$:

$$test(\psi, s) \text{ iff } \forall s' \in S\ (s' \in effect(a, s) \Longrightarrow test(\phi, s')).$$

If ψ is a weakest precondition of a and ϕ, it is denoted by $wp(a, \phi)$. Weakest preconditions are *expressible* with respect to A, G_{at} and $\mathscr{S}$ iff there is a weakest precondition in G of any $a \in A$ and $\phi \in G_{at}$. □

In the remainder of this section we assume that weakest preconditions are expressible with respect to $\mathscr{S}$. The axioms WPC1 and SI can be used to prove that $\phi a = \phi a \psi$ (see above). In this case, in all data-states where $wp(a, \psi)$ holds, ϕ holds

as well. So we have the axioms $\phi \cdot \neg wp(a,\psi) = \delta$ (SI) and $wp(a,\psi)a = wp(a,\psi)a\psi$ (WPC1). We derive:

$$\begin{aligned} \phi a &= \phi(wp(a,\psi) + \neg wp(a,\psi))a \\ &= \phi wp(a,\psi)a \\ &= \phi wp(a,\psi)a\psi \\ &= \phi wp(a,\psi)a\psi + \phi\neg wp(a,\psi)a\psi \\ &= \phi a\psi. \end{aligned}$$

The expressibility of weakest preconditions is not yet sufficient to give an axiomatic characterisation of their properties. For this we also need a constraint on the non-determinism possibly caused by the function *effect* that we call *sufficient determinism.*

Definition 3.2. Let A be a set of atomic actions and G_{at} a set of atomic guards and let $\mathcal{S} = \langle S, effect, test\rangle$ be a data environment over A and G_{at}. We say that $\mathcal{S}$ is *sufficiently deterministic* iff for all $a \in A$ and $\phi \in G_{at}$:

$$\forall s, s', s'' \in S\ (s', s'' \in effect(a,s) \implies (test(\phi, s') \iff test(\phi, s''))).$$ □

Remark that a data environment with a deterministic function *effect* is sufficiently deterministic. Now if $\mathcal{S}$ is also sufficiently deterministic, then the axioms WPC1 and WPC2 characterise (the properties of) weakest conditions in an algebraic way: WPC1 expresses that $wp(a,\phi)$ is a *precondition* of a and ϕ, and WPC2 states that $wp(a,\phi)$ is the *weakest* precondition of a and ϕ. The following lemma states that the soundness of $\mathrm{BPA}_G(\mathcal{S})$ implies sufficient determinism.

Lemma 3.3. *Let $\mathcal{S}$ be some data environment over a set A of atomic actions and a set G_{at} of atomic guards. If weakest preconditions are expressible and $\mathrm{BPA}_G(\mathcal{S})$ is sound, then $\mathcal{S}$ is sufficiently deterministic.*

Proof. Suppose $\mathcal{S}$ is not sufficiently deterministic. So there are $a \in A$, $\phi \in G_{at}$ and $s \in S$ such that we can find $s', s'' \in S$ with

1. $\{s', s''\} \subseteq effect(a,s)$, and
2. $test(\phi, s')$ holds and $test(\phi, s'')$ does *not* hold.

We derive

$$\begin{aligned} a &= wp(a,\phi)a + \neg wp(a,\phi)a \\ &= wp(a,\phi)a\phi + \neg wp(a,\phi)a\neg\phi \end{aligned}$$

but obviously $(a,s) \not\leftrightarrows_{\mathcal{S}} (wp(a,\phi)a\phi + \neg wp(a,\phi)a\neg\phi, s)$, which contradicts the supposition. □

Remark 3.4. Weakest preconditions can be extended to guards as follows (adopting the use of $\neg$ on guards as defined in 3.1):

$wp(a,\neg\alpha)$	abbreviates	$\neg wp(a,\alpha)$
$wp(a,\alpha+\beta)$	abbreviates	$wp(a,\alpha) + wp(a,\beta)$
$wp(a,\alpha\beta)$	abbreviates	$wp(a,\alpha) \cdot wp(a,\beta)$.

Weakest preconditions of guards behave as expected: they satisfy the axiom schemes WPC1 and WPC2 of $\mathrm{BPA}_G(\mathcal{S})$, i.e. we have:

$$\mathrm{BPA}_G(\mathcal{S}) \vdash wp(a,\alpha)a\alpha = wp(a,\alpha)a$$

for any $a \in A$ and guard α over G. We show this in case $\alpha \equiv \neg\beta$:

$$\begin{array}{lrcll} \text{WPC1}: & \neg wp(a,\beta)a\neg\beta & = & \neg wp(a,\beta)a & \text{(from WPC2)} \\ \text{WPC2}: & \neg\neg wp(a,\beta)a\neg\neg\beta & = & \neg\neg wp(a,\beta)a & \text{(from WPC1).} \end{array}$$ □

We conclude the introduction of $\mathrm{BPA}_G(\mathscr{S})$ with some small observations. First observe that $\mathrm{BPA}_G(\mathscr{S})$ is not meaningful if weakest preconditions cannot be expressed in $\mathscr{S}$ (we cannot even read its axioms). Furthermore note that the axiom SI cannot be replaced by the simpler axiom

$$\phi = \psi \quad \text{if} \quad \forall s \in S\ (test(\phi, s) \Longleftrightarrow test(\psi, s)).$$

If e.g. ϕ holds in data-states s_0, s_1 and ψ only holds in s_0, then $\phi\psi \leftrightarrows_{\mathscr{S}} \psi$, but $\phi\psi = \psi$ cannot be derived with the scheme above. Finally, note that the axiom G4 (i.e., $a(\phi x + \neg\phi y) \subseteq ax + ay$) is derivable:

$$\begin{array}{rcl} \mathrm{BPA}_G(\mathscr{S}) \vdash ax + ay & = & (wp(a,\phi) + \neg wp(a,\phi))(ax + ay) \\ & \supseteq & wp(a,\phi)ax + \neg wp(a,\phi)ay \\ & = & wp(a,\phi)a\phi x + \neg wp(a,\phi)a\neg\phi y \\ & = & wp(a,\phi)a\phi(\phi x + \neg\phi y) + \neg wp(a,\phi)a\neg\phi(\phi x + \neg\phi y) \\ & = & wp(a,\phi)a(\phi x + \neg\phi y) + wp(a,\neg\phi)a(\phi x + \neg\phi y) \\ & = & (wp(a,\phi) + \neg wp(a,\phi))a(\phi x + \neg\phi y) \\ & = & a(\phi x + \neg\phi y). \end{array}$$

3.2. Soundness and Completeness

In the following let $\mathscr{S}$ be a data environment over A and G_{at} such that weakest preconditions are expressible and $\mathscr{S}$ is sufficiently deterministic. As stated in Lemma 2.14., the relation $\leftrightarrows_{\mathscr{S}}$ is a congruence. We state without proof that $\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP}$ is sound with respect to $\mathscr{S}$ (see Theorem 2.33., and it is easy to check that the 'new' axioms are sound).

Theorem 3.5. *(Soundness) Let $\mathscr{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. Let p, q be closed terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$. If $\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash p = q$, then $p \leftrightarrows_{\mathscr{S}} q$.* □

We show that the axiom system $\mathrm{BPA}_G(\mathscr{S})$ completely axiomatises bisimulation equivalence in $\mathscr{S}$, i.e. the relation $\leftrightarrows_{\mathscr{S}}$, between the closed terms over $\Sigma(\mathrm{BPA}_G)$. In order to do so we use some results of section 2, though we do not need the concepts of A-basic and G-basic terms over $\Sigma(\mathrm{BPA}_G)$. The reason for this is that weakest preconditions allow us to manipulate closed terms over $\Sigma(\mathrm{BPA}_G)$ in such a way that any basic guard different from δ, ϵ can occur only at 'head level'. This makes it possible to use a much simpler type of basic terms in proving completeness. We first illustrate what kind of manipulation we mean. As an example consider the term $a\neg\phi c(b + \epsilon)$. We derive

$$\begin{array}{rcl} a\neg\phi c(b+\epsilon) & = & wp(a,\phi)a\neg\phi c(b+\epsilon) + \neg wp(a,\phi)a\neg\phi c(b+\epsilon) \\ & = & wp(a,\phi)a\phi\neg\phi c(b+\epsilon) + \neg wp(a,\phi)ac(b+\epsilon) \\ & = & wp(a,\phi)a\delta + \neg wp(a,\phi)ac(b+\epsilon) \end{array}$$

with all basic guards different from δ, ϵ at head level. Using the possibility to push basic guards to head level as illustrated above, it suffices to define the following simpler syntactic class of basic terms.

Definition 3.6. A term p over $\Sigma(\mathrm{BPA}_G)$ is called *basic over* some reference set R iff the following conditions hold:

1. $\mathrm{A1,A2} \vdash p = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$,
2. For all $\vec{\phi} \in R^{co}$ the term $q_{\vec{\phi}}$ is a term in *atomic prefix normal form* over $\Sigma(\mathrm{BPA}_G)$:

   $$p ::= \delta \mid \epsilon \mid ap \mid p + p$$

   where $a \in A$. □

In the following two lemmas we show that for any closed term p over $\Sigma(\mathrm{BPA}_G)$ there exists a basic term p' (over some reference set) satisfying

$$\mathrm{BPA}_G(\mathscr{S}) \vdash p = p'.$$

Hence we may restrict our attention to basic terms in proving completeness, and exploit their syntactic structure. Particularly, if two basic terms p, q are *not* provably equal, then there is a data-state s such that $(p, s) \not\leftrightarrow_{\mathscr{S}} (q, s)$.

Lemma 3.7. *Let $a \in A$ and R be some reference set. For any term t over $\Sigma(\mathrm{BPA}_G)$ it holds that*

$$\mathrm{BPA}_G(\mathscr{S}) \vdash t = \sum_{\vec{\phi} \in R^{co}} wp(a, \vec{\phi}) \cdot t.$$

Proof. By induction on the cardinality of R. □

Lemma 3.8. *(Basic form) If p is a closed term over $\Sigma(\mathrm{BPA}_G)$, then there is a basic term p' over some reference set R such that $\mathrm{BPA}_G(\mathscr{S}) \vdash p = p'$.*

Proof. By Lemma 2.20. we may assume that p is a term in prefix normal form over $\Sigma(\mathrm{BPA}_G)$ and we apply induction on the structure of p:

$p \equiv \delta$ **or** $p \equiv \epsilon$. By Lemma 2.17. we have

$$\delta = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}\delta \quad \text{and} \quad \epsilon = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}\epsilon, \text{ respectively,}$$

for any reference set R.

$p \equiv \phi q$. By the induction hypothesis there is a reference set R such that

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

with all the terms $q_{\vec{\phi}}$ in atomic prefix normal form over $\Sigma(\mathrm{BPA}_G)$. Let $R_1 \overset{\text{def}}{=} \{\phi\} \cup R$. By Lemma 2.17. we have

$$\begin{aligned} \phi q &= \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi}\phi \cdot \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}} \\ &= \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot q'_{\vec{\psi}} \end{aligned}$$

where for all $\vec{\psi} \in R_1^{co}$

$$q'_{\vec{\psi}} \equiv \begin{cases} q_{\vec{\phi}} & \text{if } \phi \text{ occurs in } \vec{\psi} \text{ and } \vec{\phi} \text{ occurs in } \vec{\psi}, \\ \delta & \text{otherwise.} \end{cases}$$

Furthermore

$$\sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} q'_{\vec{\psi}}$$

is clearly a basic term over R_1.

$p \equiv \neg\phi q$. Likewise.

$p \equiv aq$. By the induction hypothesis there is a reference set R such that

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q_{\vec{\phi}}$$

with all the terms $q_{\vec{\phi}}$ in atomic prefix normal form over $\Sigma(\mathrm{BPA}_G)$. We derive

$$\begin{aligned} aq &= a \cdot \sum_{\vec{\psi} \in R^{co}} \vec{\psi} q_{\vec{\psi}} \\ &= \sum_{\vec{\phi} \in R^{co}} wp(a, \vec{\phi}) \cdot a \cdot \sum_{\vec{\psi} \in R^{co}} \vec{\psi} q_{\vec{\psi}} \qquad \text{(by Lemma 3.7.)} \\ &= \sum_{\vec{\phi} \in R^{co}} wp(a, \vec{\phi}) \cdot a \cdot \vec{\phi} q_{\vec{\phi}} \qquad (\vec{\psi} \not\equiv \vec{\phi} \implies \vec{\phi} \cdot \vec{\psi} = \delta) \\ &= \sum_{\vec{\phi} \in R^{co}} wp(a, \vec{\phi}) \cdot a \cdot q_{\vec{\phi}} \end{aligned}$$

Let $wp(a, R) \stackrel{\text{def}}{=} \{Ref(wp(a, \phi)) \mid \phi \in R\}$. Note that $wp(a, R)$ may be empty (for instance in case $R = \{\phi\}$ and $wp(a, \phi) = \epsilon$). Let

$$R_1 \stackrel{\text{def}}{=} \{\phi\} \cup wp(a, R)$$

for some arbitrary $\phi \in G_{at}$. Obviously R_1 is a reference set, and we derive

$$\begin{aligned} aq &= \sum_{\vec{\phi} \in R^{co}} wp(a, \vec{\phi}) \cdot a \cdot q_{\vec{\phi}} \\ &= \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} \cdot \sum_{\{\vec{\phi} \in R^{co} \mid \phi \cdot wp(a,\vec{\phi}) = \vec{\psi} \vee \neg\phi \cdot wp(a,\vec{\phi}) = \vec{\psi}\}} a \cdot q_{\vec{\phi}} \end{aligned}$$

with the latter term basic over R_1.

$p \equiv q + r$. By the induction hypothesis there are reference sets R_1, R_2 such that

$$q = \sum_{\vec{\psi} \in R_1^{co}} \vec{\psi} q_{\vec{\psi}} \text{ and } r = \sum_{\vec{\theta} \in R_2^{co}} \vec{\theta} r_{\vec{\theta}}$$

with all the terms $q_{\vec{\psi}}, r_{\vec{\theta}}$ in atomic prefix normal form over $\Sigma(\mathrm{BPA}_G)$. Let $R \stackrel{\text{def}}{=} R_1 \cup R_2$. By Lemma 2.17. we have

$$q = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} q'_{\vec{\phi}} \text{ and } r = \sum_{\vec{\phi} \in R^{co}} \vec{\phi} r'_{\vec{\phi}}.$$

where for all $\vec{\phi} \in R^{co}$ the terms $q'_{\vec{\phi}}, r'_{\vec{\phi}}$ are defined as follows:

$$q'_{\vec{\phi}} \equiv q_{\vec{\psi}}, \quad \text{provided } \vec{\psi} \text{ occurs in } \vec{\phi},$$
$$r'_{\vec{\phi}} \equiv r_{\vec{\theta}}, \quad \text{provided } \vec{\theta} \text{ occurs in } \vec{\phi}.$$

We derive

$$q + r = \sum_{\vec{\phi} \in R^{co}} \vec{\phi}(q'_{\vec{\phi}} + r'_{\vec{\phi}})$$

and clearly the right hand side term is basic over R. □

The syntax of a basic term is sufficiently strict to derive information about its (syntactic) structure from its operational behaviour. As announced before, we show that if two basic terms over some reference set R do *not* obey certain provable characteristics, then we can find a data-state $s \in S$ such that the associated transition systems with initial data-state s are *not* $\mathcal{S}$-bisimilar. The proof of this fact is quite easy compared to the proof of the related Lemma 2.27.

Lemma 3.9. *Let p_1, p_2 be basic terms over some reference set R. If there is $\vec{\phi} \in R^{co}$ such that*

1. $\mathrm{BPA}_G(\mathcal{S}) \not\vdash \vec{\phi} = \delta$,
2. $\vec{\phi} q^i_{\vec{\phi}} \sqsubseteq p_i$ $(i = 1, 2)$,
3. $\mathrm{BPA}^3_G \not\vdash q^1_{\vec{\phi}} = q^2_{\vec{\phi}}$,

then $\exists s \in S\ ((p_1, s) \not\leftrightarrows_{\mathcal{S}} (p_2, s))$.

Proof. Assume that $\vec{\phi}$ satisfies the conditions of the lemma. So by 1 we can find some $s \in S$ such that $(\vec{\phi}, s) \not\rightarrow$ (δ, s).

Now suppose $(p_1, s) \leftrightarrows_{\mathcal{S}} (p_2, s)$ by some $\mathcal{S}$-bisimulation B. Adding the tuple $((q^1_{\vec{\phi}}, s), (q^2_{\vec{\phi}}, s))$ to B would by condition 2 result in an $\mathcal{S}$-bisimulation establishing $(q^1_{\vec{\phi}}, s) \leftrightarrows_{\mathcal{S}} (q^2_{\vec{\phi}}, s)$. We show that for all terms q_1, q_2 in *atomic prefix normal form* that

$$\exists s \in S\ ((q_1, s) \leftrightarrows_{\mathcal{S}} (q_2, s)) \implies \mathrm{BPA}^3_G \vdash q_1 = q_2$$

contradicting condition 3 of the lemma, and therefore the supposition.

Assume $(q_1, s) \leftrightarrows_{\mathcal{S}} (q_2, s)$, we show that $\mathrm{BPA}^3_G \vdash q_1 = q_2$ by proving that any syntactic summand (see Definition 2.22.) of q_1 is provably equal to a syntactic summand of q_2 and vice versa. We apply induction on $|q_1| + |q_2|$ (see Definition 2.26.). The case $|q_1| + |q_2| = 0$ is trivial, so assume $|q_1| + |q_2| > 0$. By symmetry it suffices to show that if $t \sqsubseteq q_1$ for some term t, then we can find a term t' such that $\mathrm{BPA}^3_G \vdash t = t'$ and $t' \sqsubseteq q_2$.

Suppose $ar \sqsubseteq q_1$. For any $s' \in \mathit{effect}(a, s)$ we have $(q_1, s) \xrightarrow{a} (\epsilon r, s')$. By assumption $(q_2, s) \xrightarrow{a} (r', s')$ for some term r', satisfying $(\epsilon r, s') \leftrightarrows_{\mathcal{S}} (r', s')$. By a simple argument (cf. Lemma 2.23.) there exists a term r'' in atomic prefix normal form such that $\epsilon r'' \equiv r'$ and $ar'' \sqsubseteq p'$. So $(\epsilon r, s') \leftrightarrows_{\mathcal{S}} (\epsilon r'', s')$, and thus $(r, s') \leftrightarrows_{\mathcal{S}} (r'', s')$. By the induction hypothesis $r = r''$, and hence $ar = ar''$.

In case $\epsilon \sqsubseteq q_1$, we can show in the same way that $\epsilon \sqsubseteq q_2$. □

Connecting all the results proved so far, we can prove the completeness of $\mathrm{BPA}_G(\mathcal{S})$ in a simple way.

Theorem 3.10. *(Completeness) Let $\mathscr{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. Let r_1, r_2 be closed terms over $\Sigma(\mathrm{BPA}_G)$. If $r_1 \leftrightarrows_{\mathscr{S}} r_2$, then $\mathrm{BPA}_G(\mathscr{S}) \vdash r_1 = r_2$.*

Proof. We prove the theorem by contraposition. Suppose $r_1 \neq r_2$. We have to show $r_1 \not\leftrightarrows_{\mathscr{S}} r_2$. According to Lemma 3.8 there are basic terms p_1', p_2' over reference sets R_1, R_2, respectively, such that $r_i = p_i'$ $(i = 1, 2)$. By Lemma 2.17. we can find basic terms p_1, p_2 over $R = R_1 \cup R_2$ such that $p_i = p_i'$, and hence $r_i = p_i$ $(i = 1, 2)$. By soundness (see Theorem 3.5.) we have that $r_i \leftrightarrows_{\mathscr{S}} p_i$. Because $p_1 \neq p_2$, there must be $\vec{\phi} \in R^{co}$ satisfying the conditions of the previous Lemma 3.9., i.e., there is some $s \in S$ such that $(p_1, s) \not\leftrightarrows_{\mathscr{S}} (p_2, s)$. As $\leftrightarrows_{\mathscr{S}}$ is an equivalence relation, we conclude $(r_1, s) \not\leftrightarrows_{\mathscr{S}} (r_2, s)$, which finishes our proof. □

3.3. An Example: The Process *SWAP*

Process algebra with guards can be used to express and prove partial correctness formulas in Hoare logic. In section 5 we elaborate on this idea. Here a simple example that is often used as an illustration of Hoare logic is presented and its correctness is shown.

First we transform $\mathrm{BPA}_G(\mathscr{S})$ into a small programming language with Boolean guards and assignments (cf. the setting of the examples on **if - then - else - fi** and **while - do - od** in the previous section). Our language has the signature of $\Sigma(\mathrm{BPA}_G)$ and we have some set $\mathscr{V} = \{x, y, \ldots\}$ of data variables. Atomic actions have the form:

$$[x := t]$$

with $x \in \mathscr{V}$ a variable ranging over the set $\mathbb{Z}$ of integers and t an integer expression. We assume that some interpretation $[\![\cdot]\!]$ from closed integer expressions to integers is given. Atomic guards have the form

$$\langle t = u \rangle$$

where t and u are both integer expressions.

The components of the data environment $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test} \rangle$ are straightforward to define:

$$S = \mathbb{Z}^{\mathscr{V}}$$

i.e. the set of mappings from $\mathscr{V}$ to the integers. We write ρ, σ for data-states in S, and we assume that the domain $\mathscr{V}$ of these mappings is extended to integer expressions in the standard way. The function *effect* is defined by:

$$\mathit{effect}([x := t], \rho) = \{\rho[[\![\rho(t)]\!]/x]\}$$

where $\rho[n/x]$ is as the mapping ρ, except that x is mapped to n. We define the predicate *test* by:

$$\mathit{test}(\langle t = u \rangle, \rho) \Longleftrightarrow ([\![\rho(t)]\!] = [\![\rho(u)]\!]).$$

Note that the effect function is deterministic, so $\mathscr{S}$ is certainly sufficiently deterministic. Weakest preconditions can easily be expressed:

$$wp([x := t], \langle u = v \rangle) = \langle u[t/x] = v[t/x] \rangle.$$

The axiom SI cannot be formulated so easily, partly because we have not yet defined integer expressions very precisely. For this example we only need:

$$\langle t = u\rangle \cdot \neg\langle t' = u'\rangle = \delta \text{ and } \neg\langle t = u\rangle \cdot \langle t' = u'\rangle = \delta$$

if $\forall \rho \in S\ [\![\rho(t)]\!] = [\![\rho(t')]\!]$ and $[\![\rho(u)]\!] = [\![\rho(u')]\!]$.

In this language we can express the following tiny program *SWAP* that exchanges the initial values of x and y without using any other variables.

$$SWAP \quad = \quad [x := x + y] \cdot [y := x - y] \cdot [x := x - y].$$

The correctness of this program can be expressed by the following equation:

$$\langle x = n\rangle \cdot \langle y = m\rangle \cdot SWAP = \langle x = n\rangle \cdot \langle y = m\rangle \cdot SWAP \cdot \langle x = m\rangle \cdot \langle y = n\rangle.$$

This equation says that if *SWAP* is executed in an initial data-state where $x = n$ and $y = m$, then after termination of *SWAP* it must hold, i.e. it can be derived, that $x = m$ and $y = n$. So *SWAP* indeed exchanges the values of x and y.

The correctness of *SWAP* can be proved as follows:

$$\begin{aligned}
&\langle x = n\rangle \cdot \langle y = m\rangle \cdot SWAP \\
&\overset{\text{SI}}{=} \quad \langle (x + y) - y = n\rangle \cdot \langle (x + y) - ((x + y) - y) = m\rangle \cdot SWAP \\
&\overset{\text{WPC1,SI}}{=} \quad \langle x = n\rangle \cdot \langle y = m\rangle \cdot [x := x + y] \cdot \langle x - y = n\rangle \cdot \langle x - (x - y) = m\rangle \cdot \\
&\qquad\quad [y := x - y] \cdot [x := x - y] \\
&\overset{\text{WPC1}}{=} \quad \langle x = n\rangle \cdot \langle y = m\rangle \cdot [x := x + y] \cdot \langle x = n\rangle \cdot \langle y = m\rangle \cdot \\
&\qquad\quad [y := x - y] \cdot \langle y = n\rangle \cdot \langle x - y = m\rangle \cdot [x := x - y] \\
&\overset{\text{WPC1}}{=} \quad \langle x = n\rangle \cdot \langle y = m\rangle \cdot SWAP \cdot \langle x = m\rangle \cdot \langle y = n\rangle.
\end{aligned}$$

Note that we have used the identities

$$\langle x = n\rangle = \langle (x + y) - y = n\rangle$$

and

$$\langle x = m\rangle = \langle (x + y) - ((x + y) - y) = m\rangle.$$

We show below how the first one is derived:

$$\begin{aligned}
\langle x = n\rangle &= \langle x = n\rangle \cdot \epsilon \\
&= \langle x = n\rangle \cdot (\langle (x + y) - y = n\rangle + \neg\langle (x + y) - y = n\rangle) \\
&= \langle x = n\rangle \cdot \langle (x + y) - y = n\rangle + \langle x = n\rangle \cdot \neg\langle (x + y) - y = n\rangle \\
&= \langle x = n\rangle \cdot \langle (x + y) - y = n\rangle + \delta \\
&= \langle x = n\rangle \cdot \langle (x + y) - y = n\rangle + \neg\langle x = n\rangle \cdot \langle (x + y) - y = n\rangle \\
&= (\langle x = n\rangle + \neg\langle x = n\rangle) \cdot \langle (x + y) - y = n\rangle \\
&= \epsilon \cdot \langle (x + y) - y = n\rangle \\
&= \langle (x + y) - y = n\rangle
\end{aligned}$$

4. Parallel Processes with Guards

In this section Basic Process Algebra with guards is extended with operators for parallelism. We give Plotkin-style rules to express the operational behaviour of these operators and show that $\mathcal{S}$-bisimilarity is not a congruence any longer. We deal with this problem by introducing another bisimulation equivalence, called

constants:	a	for any atomic action $a \in A$
	ϕ	for any basic guard $\phi \in G$
unary operators:	∂_H	encapsulation, for any $H \subseteq A$
binary operators:	$+$	alternative composition (sum)
	$\cdot$	sequential composition (product)
	$\|$	parallel composition (merge)
	$\lfloor\!\lfloor$	left-merge
	$\mid$	communication-merge

Fig. 7. The signature $\Sigma(\mathrm{ACP}_G)$.

global $\mathcal{S}$-bisimulation equivalence which is finer than $\mathcal{S}$-bisimilarity. Global $\mathcal{S}$-bisimulation equivalence is a congruence, but it is not so natural. Moreover, the axioms WPC1, WPC2 and G4 are not valid anymore in global $\mathcal{S}$-bisimulation.

We present the axiom system ACP_G which is based on ACP (the Algebra of Communicating Processes [BeK84a]). ACP_G is sound for global $\mathcal{S}$-bisimilarity, and for finite processes also complete. This axiom system enables us to prove $\mathcal{S}$-bisimulation equivalence between processes: using ACP_G every closed process term can be proved equivalent to one without parallel operators, and then BPA_G^4 or $\mathrm{BPA}_G(\mathcal{S})$ can be used to prove $\mathcal{S}$-bisimilarity. This section is concluded with an example in which the correctness of a parallel process is proved in this way.

4.1. Axioms and a Two-Phase Calculus

We extend the language of $\Sigma(\mathrm{BPA}_G)$ to a concurrent one, suitable to describe the behaviour of parallel, communicating processes. Communication is modelled by a communication function $\gamma : A \times A \longrightarrow A_\delta$ that is commutative and associative. If $\gamma(a,b)$ is δ, then a and b cannot communicate, and if $\gamma(a,b) = c$, then c is the atomic action resulting from the communication between a and b.

Concurrency is described by three operators, the *merge* $\|$, the *left-merge* $\lfloor\!\lfloor$ and the *communication-merge* $|$.

$p \| q$ represents the parallel execution of p and q. It starts when one of its components starts, and terminates if both of them do.

$p \lfloor\!\lfloor q$ is as $p \| q$, but under the assumption that the first action that is performed comes from p (it may be the case that the behaviour of p starts with the evaluation of a guard).

$p \mid q$ is as $p \| q$, but the first action is a communication between p and q.

We present encapsulation operators ∂_H (for any $H \subseteq A$) that block atomic actions in H by renaming them into δ. Encapsulation is used to enforce communication between processes. The signature $\Sigma(\mathrm{ACP}_G)$ is summarised in Fig. 7.
For the terms over $\Sigma(\mathrm{ACP}_G)$ we have the axioms given in Fig. 8, where $a, b \in A$, $H \subseteq A$ and $\phi \in G$ (note that the axiom $a(\phi x + \neg\phi y) \subseteq ax + ay$ (G4) is absent). Most of these axioms are standard for ACP (see [BeK84a]), and, apart from G1, G2 and G3, only the axioms EM10, EM11 and D0 are new. The axiom EM10 (EM11) expresses that a basic guard ϕ in $\phi x \lfloor\!\lfloor y$ ($\phi x \mid y$, respectively) also may prevent that y happens.

(A1)	$x+y=y+x$	(G1)	$\phi \cdot \neg\phi = \delta$
(A2)	$x+(y+z)=(x+y)+z$	(G2)	$\phi + \neg\phi = \epsilon$
(A3)	$x+x=x$	(G3)	$\phi(x+y)=\phi x+\phi y$
(A4)	$(x+y)z=xz+yz$		
(A5)	$(xy)z=x(yz)$		
(A6)	$x+\delta=x$		
(A7)	$\delta x=\delta$		
(A8)	$\epsilon x=x$		
(A9)	$x\epsilon=x$		
(CF)	$a \mid b=\gamma(a,b)$		
(EM1)	$x \parallel y=x \lfloor\!\lfloor y+y \lfloor\!\lfloor x+x \mid y$	(EM10)	$\phi x \lfloor\!\lfloor y=\phi(x \lfloor\!\lfloor y)$
(EM2)	$\epsilon \lfloor\!\lfloor x=\delta$	(EM11)	$\phi x \mid y=\phi(x \mid y)$
(EM3)	$ax \lfloor\!\lfloor y=a(x \parallel y)$		
(EM4)	$(x+y) \lfloor\!\lfloor z=x \lfloor\!\lfloor z+y \lfloor\!\lfloor z$		
(EM5)	$x \mid y=y \mid x$	(D0)	$\partial_H(\phi)=\phi$
(EM6)	$\epsilon \mid \epsilon=\epsilon$	(D1)	$\partial_H(a)=a$ if $a \notin H$
(EM7)	$\epsilon \mid ax=\delta$	(D2)	$\partial_H(a)=\delta$ if $a \in H$
(EM8)	$ax \mid by=(a \mid b)(x \parallel y)$	(D3)	$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$
(EM9)	$(x+y) \mid z=x \mid z+y \mid z$	(D4)	$\partial_H(xy)=\partial_H(x)\partial_H(y)$

Fig. 8. The axioms of ACP_G where $\phi \in G$, $a,b \in A$ and $H \subseteq A$.

Using ACP_G any closed term over $\Sigma(\mathrm{ACP}_G)$ can be proved equal to one without merge operators, i.e. a closed term over $\Sigma(\mathrm{BPA}_G)$.

Theorem 4.1. *(Elimination) Let p be a closed term over $\Sigma(\mathrm{ACP}_G)$. There is a closed term q over $\Sigma(\mathrm{BPA}_G)$ such that $\mathrm{ACP}_G \vdash p=q$.*

Proof. By induction on the structure of terms. □

The axiom systems ACP_G and BPA_G^4 or $\mathrm{BPA}_G^3(\mathscr{S})$ cannot be combined in bisimulation semantics; if G4 is added to ACP_G we can derive the following:

$$\begin{aligned} \mathrm{ACP}_G + \mathrm{G4} \quad &\vdash \quad a(b \parallel d)+a(c \parallel d)+d(ab+ac) \qquad (1)\\ &= \quad (ab+ac) \parallel d \\ &\overset{\mathrm{G4}}{=} \quad (ab+ac+a(\phi b+\neg\phi c)) \parallel d \\ &\supseteq \quad a(\phi bd+\neg\phi cd+d(\phi b+\neg\phi c)). \qquad (2) \end{aligned}$$

So, in (2) it can be the case that after an a step ϕ holds, and we arrive in a state where we can do a b or a d step. Performing the d step can bring us in a state were $\neg\phi$ holds, so the only possible step left is a c step. This situation cannot be mimicked in (1). Therefore, every term with (2) as a summand is not bisimilar to (1) for any reasonable form of bisimulation. So $\mathrm{ACP}_G + \mathrm{G4}$ is not sound in any bisimulation semantics. (Note that the data environment in this example can be sufficiently deterministic.)

Because we still want to derive $\mathscr{S}$-bisimilarity between closed terms containing merge operators, we introduce a *two-phase* calculus that does not have these problems. Derivability in this calculus is denoted by $\vdash_2$.

Definition 4.2 *(A two-phase calculus $\vdash_2$).* Let p_1, p_2 be closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. We write

$$\mathrm{ACP}_G^4 \vdash_2 p_1 = p_2$$

iff there are closed terms q_1, q_2 over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ such that

$$\mathrm{ACP}_G \vdash p_i = q_i \quad (i = 1,2) \text{ and } \mathrm{BPA}_G^4 \vdash q_1 = q_2.$$

Furthermore, we write

$$\mathrm{ACP}_G(\mathscr{S}) \vdash_2 p_1 = p_2$$

iff there are closed terms q_1, q_2 over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ such that

$$\mathrm{ACP}_G \vdash p_i = q_i \quad (i = 1,2) \text{ and } \mathrm{BPA}_G(\mathscr{S}) \vdash q_1 = q_2.$$

We sometimes put REC + RSP in front of $\vdash_2$ which means that we may use REC and RSP in proving $p_i = q_i$ $(i = 1,2)$ and $q_1 = q_2$. □

4.2. Operational Semantics and Soundness

Let $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ be some data environment over a set A of atomic actions and a set G_{at} of atomic guards. The transition rules in Fig. 9 and the transition rule for guarded recursive specifications (see Fig. 5) determine the transition relation $\longrightarrow_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}},\mathscr{S}}$ over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. Remark that these rules formalise the informal description of the new operators given earlier, and that all rules given for $\Sigma(\mathrm{BPA}_G)$ in Fig. 3 are included. Let p be a closed term over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. For any $s \in S$ the transition system $\mathscr{A}(p,s)$ is defined as

$$\mathscr{A}(p,s) \stackrel{\mathrm{def}}{=} \langle C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S)\,,\, A_{\surd}\,,\, \longrightarrow_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}},\mathscr{S}}\,,\, (p,s)\rangle.$$

We first show by an example that the notion of '$\mathscr{S}$-bisimilarity' as defined in 2.13 for the configurations over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ gives in general no congruence relation between the closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$.

Example 4.3. Consider the data environment $\langle\{s_0, s_1\}, \mathit{effect}, \mathit{test}\rangle$ in which

- $\forall s \in S\ (\mathit{effect}(a,s) = \{s_0\})$ for some $a \in A$;
- $\forall s \in S\ (\mathit{effect}(b,s) = \{s_1\})$ for some $b \in A$;
- $\mathit{test}(\phi, s_0)$ and not $\mathit{test}(\phi, s_1)$ for some $\phi \in G$.

In this case we have $a\delta \leftrightarrows_{\mathscr{S}} a\neg\phi$ but not $a\delta \parallel b \leftrightarrows_{\mathscr{S}} a\neg\phi \parallel b$, for the transition system $\mathscr{A}(a\neg\phi \parallel b, s_0)$ has an execution path

$$(a\neg\phi \parallel b, s_0) \xrightarrow{a} (\epsilon\neg\phi \parallel b\neg\phi, s_0) \xrightarrow{b} (\epsilon\neg\phi \parallel \epsilon, s_1) \not\xrightarrow{\surd} (\delta \parallel \delta, s_1)$$

that is not present in $\mathscr{A}(a\delta \parallel b, s_0)$. □

We define a different bisimulation equivalence, called *global $\mathscr{S}$-bisimilarity*, that is a congruence for the merge operators. The idea behind a global $\mathscr{S}$-bisimulation is that a context $p \parallel (.)$ around a process q can change the data-state of q at any time and global $\mathscr{S}$-bisimulation equivalence must be resistant against such changes. So, a configuration (p_1, s) is related to a configuration (p_2, s) if $(p_1, s) \xrightarrow{a} (q_1, s')$ implies $(p_2, s) \xrightarrow{a} (q_2, s')$ and, as the environment may change s', the process q_1 is related to q_2 in *any* data-state:

$a \in A \quad (a,s) \xrightarrow{a} (\epsilon, s') \text{ if } s' \in \mathit{effect}(a,s)$

$\phi \in G \quad (\phi,s) \xrightarrow{\surd} (\delta,s) \text{ if } \mathit{test}(\phi,s)$

$+ \quad \dfrac{(x,s) \xrightarrow{a} (x',s')}{(x+y,s) \xrightarrow{a} (x',s')} \qquad \dfrac{(y,s) \xrightarrow{a} (y',s')}{(x+y,s) \xrightarrow{a} (y',s')}$

$\dfrac{(x,s) \xrightarrow{a} (x',s')}{(xy,s) \xrightarrow{a} (x'y,s')} \text{ if } a \neq \surd \qquad \dfrac{(x,s) \xrightarrow{\surd} (x',s') \quad (y,s) \xrightarrow{a} (y',s'')}{(xy,s) \xrightarrow{a} (y',s'')}$

$\| \quad \dfrac{(x,s) \xrightarrow{a} (x',s')}{(x \parallel y,s) \xrightarrow{a} (x' \parallel y,s')} \text{ if } a \neq \surd \qquad \dfrac{(y,s) \xrightarrow{a} (y',s')}{(x \parallel y,s) \xrightarrow{a} (x \parallel y',s')} \text{ if } a \neq \surd$

$\dfrac{(x,s) \xrightarrow{a} (x',s') \quad (y,s) \xrightarrow{b} (y',s'')}{(x \parallel y,s) \xrightarrow{\gamma(a,b)} (x' \parallel y',s''')} \text{ if } \gamma(a,b) \neq \delta,\ a,b \neq \surd, \text{ and } s''' \in \mathit{effect}(\gamma(a,b),s)$

$\dfrac{(x,s) \xrightarrow{\surd} (x',s') \quad (y,s) \xrightarrow{\surd} (y',s')}{(x \parallel y,s) \xrightarrow{\surd} (x' \parallel y',s')}$

$\lfloor \quad \dfrac{(x,s) \xrightarrow{a} (x',s')}{(x \lfloor y,s) \xrightarrow{a} (x' \parallel y,s')} \text{ if } a \neq \surd$

$| \quad \dfrac{(x,s) \xrightarrow{a} (x',s') \quad (y,s) \xrightarrow{b} (y',s'')}{(x \mid y,s) \xrightarrow{\gamma(a,b)} (x' \parallel y',s''')} \text{ if } \gamma(a,b) \neq \delta,\ a,b \neq \surd, \text{ and } s''' \in \mathit{effect}(\gamma(a,b),s)$

$\dfrac{(x,s) \xrightarrow{\surd} (x',s') \quad (y,s) \xrightarrow{\surd} (y',s')}{(x \mid y,s) \xrightarrow{\surd} (x' \parallel y',s')}$

$\partial_H \quad \dfrac{(x,s) \xrightarrow{a} (x',s')}{(\partial_H(x),s) \xrightarrow{a} (\partial_H(x'),s')} \text{ if } a \notin H \subseteq A$

Fig. 9. Transition rules for ACP_G ($a,b \in A_{\surd}$, $H \subseteq A$).

Definition 4.4. Let Σ be a signature, $\mathscr{S}$ a data environment with data-state space S and $\longrightarrow_{\mathscr{S}}$ a transition relation over $C(\Sigma,S)$.

- A binary relation $R \subseteq C(\Sigma,S) \times C(\Sigma,S)$ is a *global $\mathscr{S}$-bisimulation* iff R satisfies the following (global) version of the transfer property: for all $(p,s),(q,s) \in C(\Sigma,S)$ with $(p,s)R(q,s)$:

1. Whenever $(p,s) \xrightarrow{a}_{\mathscr{S}} (p',s')$ for some a and (p',s'), then, for some q', also $(q,s) \xrightarrow{a}_{\mathscr{S}} (q',s')$ and $\forall s'' \in S\ ((p',s'')R(q',s''))$,
2. Conversely, whenever $(q,s) \xrightarrow{a}_{\mathscr{S}} (q',s')$ for some a and (q',s'), then, for some p', also $(p,s) \xrightarrow{a}_{\mathscr{S}} (p',s')$ and $\forall s'' \in S\ ((p',s'')R(q',s''))$.

- A configuration $(p,s) \in C(\Sigma,S)$ is *globally S-bisimilar* to a configuration $(q,s') \in C(\Sigma,S)$, notation

  $$(p,s) \leftrightarrows_{\mathscr{S}} (q,s')$$

  iff $s = s'$ and there is a global $\mathscr{S}$-bisimulation containing $((p,s),(q,s'))$.
- A transition system $\mathscr{A}(p,s) = \langle C(\Sigma,S), A_{\surd}, \longrightarrow_{\mathscr{S}}, (p,s)\rangle$ is *globally $\mathscr{S}$-bisimilar* with a transition system $\mathscr{A}(q,s') = \langle C(\Sigma,S), A_{\surd}, \longrightarrow_{\mathscr{S}}, (q,s')\rangle$, notation

  $$\mathscr{A}(p,s) \leftrightarrows_{\mathscr{S}} \mathscr{A}(q,s')$$

  iff $(p,s) \leftrightarrows_{\mathscr{S}} (q,s')$.
- Two closed terms p,q over Σ are *globally $\mathscr{S}$-bisimilar*, notation

  $$p \leftrightarrows_{\mathscr{S}} q$$

  iff $\mathscr{A}(p,s) \leftrightarrows_{\mathscr{S}} \mathscr{A}(q,s)$ for all $s \in S$. □

By definition of global $\mathscr{S}$-bisimilarity we have for any two closed terms p,q over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$

$$p \leftrightarrows_{\mathscr{S}} q \implies p \leftrightarrows_{\mathscr{S}} q.$$

It is not difficult to see that for any data environment $\mathscr{S}$ the relation $\leftrightarrows_{\mathscr{S}}$ is an equivalence relation over the closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$.

Our goal, i.e. global $\mathscr{S}$-bisimilarity being a congruence relation, has been achieved:

Lemma 4.5. *For any data environment $\mathscr{S}$ the relation $\leftrightarrows_{\mathscr{S}}$ is a congruence with respect to the operators of $\Sigma(\mathrm{ACP}_G)$.*

Proof. We only prove the lemma for the merge operator. Let $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ and assume that $p \leftrightarrows_{\mathscr{S}} p'$ and $q \leftrightarrows_{\mathscr{S}} q'$. So for all $s \in S$ we have global $\mathscr{S}$-bisimulations R_p^s and R_q^s such that $(p,s)R_p^s(p',s)$ and $(q,s)R_q^s(q',s)$. We have to show $(p \parallel q,s) \leftrightarrows_{\mathscr{S}} (p' \parallel q',s)$ for all $s \in S$. Fix $s_0 \in S$, and let $R_p \stackrel{\mathrm{def}}{=} \cup_{s\in S} R_p^s$ and $R_q \stackrel{\mathrm{def}}{=} \cup_{s\in S} R_q^s$. We define a relation R as follows:

$$R \stackrel{\mathrm{def}}{=} \{((r \parallel u,s),(r' \parallel u',s)) \mid (r,s)R_p(r',s)\ ,\ (u,s)R_q(u',s)\}$$

We have $(p \parallel q,s_0)R(p' \parallel q',s_0)$ and we show that R is a global $\mathscr{S}$-bisimulation. Suppose

$$(r \parallel u,s)R(r' \parallel u',s) \quad \text{and} \quad (r \parallel u,s) \xrightarrow{a} (v \parallel w,s').$$

We systematically check which application of the transition rules may have led to this transition:

$(r,s) \xrightarrow{a} (v,s')$, $u \equiv w$ and $a \neq \surd$. Because $(r,s)R_p(r',s)$ and R_p is a global $\mathscr{S}$-bisimulation, there is a v' such that $(r',s) \xrightarrow{a} (v',s')$ and $\forall s''((v,s'')R_p(v',s''))$. We derive $(r' \parallel u',s) \xrightarrow{a} (v' \parallel u',s')$. As $\forall s''((r',s'')R_p(v',s''))$ and $\forall s''((u,s'')R_q(u',s''))$, we have $\forall s''((v \parallel u,s'')R(v' \parallel u',s''))$ by definition of R.

$(u,s) \xrightarrow{a} (w,s')$, $r \equiv v$ and $a \neq \surd$. Likewise.

$(r,s) \xrightarrow{b} (v,s'')$, $(u,s) \xrightarrow{c} (w,s''')$, $a = \gamma(b,c)$ and $s' \in \mathit{effect}(a,s)$. In a similar way as above we can find v' and w' satisfying $(r',s) \xrightarrow{b} (v',s'')$ and $(u',s) \xrightarrow{c} (w',s''')$, and hence $(r' \parallel u',s) \xrightarrow{a} (v' \parallel w',s')$. As $\forall s''((v,s'')R_p(v',s''))$ and $\forall s''((w,s'')R_q(w',s''))$, we conclude $\forall s''((v \parallel w,s'')R(v' \parallel w',s''))$.

$(r,s) \xrightarrow{\surd} (v,s')$, $(u,s) \xrightarrow{\surd} (w,s')$ and $a = \surd$. Likewise. □

Theorem 4.6. *(Soundness) Let p,q be closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. If $\mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash p = q$, then $p \leftrightarrows_{\mathscr{S}} q$ for any data environment $\mathscr{S}$.*

Proof. All the axioms of ACP_G, REC and RSP are sound and $\leftrightarrows_{\mathscr{S}}$ is a congruence. As an example we prove the soundness of the axiom EM1. Let $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ be a data environment over A and G_{at} and let p,q be closed over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. Consider the relation

$$R \stackrel{\mathrm{def}}{=} Id \cup \{((p \parallel q,s),(p \between q + q \between p + p \mid q,s)) \mid s \in S\}$$

where Id is the identity relation on $C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}},S)$. It is not difficult to see that R is a global $\mathscr{S}$-bisimulation satisfying $(p \parallel q)R(p \between q + q \between p + p \mid q)$. □

With this result we immediately obtain the soundness of two-phase derivability.

Corollary 4.7. *(Soundness) Let p,q be closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$.*

1. *If $\mathrm{ACP}_G^4 + \mathrm{REC} + \mathrm{RSP} \vdash_2 p = q$, then $p \leftrightarrows_{\mathscr{S}} q$ for any data environment $\mathscr{S}$.*
2. *Let $\mathscr{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. If $\mathrm{ACP}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash_2 p = q$, then $p \leftrightarrows_{\mathscr{S}} q$.*

4.3. Completeness

We show that the axiom system ACP_G completely axiomatises global $\mathscr{S}$-bisimilarity in *all* data environments for the closed terms over $\Sigma(\mathrm{ACP}_G)$. From Theorem 4.1. and Lemma 2.21., it follows that we can restrict our attention to the G-basic and A-basic terms over $\Sigma(\mathrm{BPA}_G)$ defined in section 2. Due to the fact that global $\mathscr{S}$-bisimilarity is a finer equivalence than ordinary $\mathscr{S}$-bisimilarity, we are able to prove the related version of Lemma 2.27. in a simple way.

Note that the results from section 2 that are used here, are all proved using BPA_G^3.

Lemma 4.8. *If p_1,p_2 are G-basic terms over some reference set R and $\mathrm{ACP}_G \nvdash p_1 = p_2$, then there is a data-state $\vec{\phi}$ in $\mathscr{S}(R)$ such that $(p_1,\vec{\phi}) \not\leftrightarrows_{\mathscr{S}(R)} (p_2,\vec{\phi})$.*

Proof. By induction on $|p_1| + |p_2|$. The case $|p_1| + |p_2| = 0$ is trivial, so assume $|p_1| + |p_2| > 0$. If $p_1 \neq p_2$, then $p_1 \not\subseteq p_2$ or $p_2 \not\subseteq p_1$. Assume $p_1 \not\subseteq p_2$, so there is an A-basic term q_1 over R such that $\vec{\phi}q_1 \sqsubseteq p_1$ and $\vec{\phi}q_1 \not\subseteq p_2$ (otherwise just sum up all syntactic summands of p_1 and conclude $p_1 \subseteq p_2$).

By definition p_2 has a syntactic summand $\vec{\phi}q_2$, but $q_1 \not\subseteq q_2$ (otherwise $\vec{\phi}q_1 \subseteq \vec{\phi}q_2 \subseteq p_2$). One of the following holds:

1. $\epsilon \sqsubseteq q_1$ and $\epsilon \not\subseteq q_2$,

2. $ar \sqsubseteq q_1$ and $ar \not\subseteq q_2$ for some $a \in A$ and G-basic term r.

(If all syntactic summands of q_1 would be *provable* summands of q_2, then $q_1 \subseteq q_2$.) In the first case we have $(p_1, \vec{\phi}) \not\rightarrow \surd$ ..., whereas by Lemma 2.23. $(p_2, \vec{\phi})$ has no such transition, so $(p_1, \vec{\phi}) \not\trianglerighteq_{\mathscr{S}} (p_2, \vec{\phi})$. We evaluate case 2:

either q_2 has no syntactic summands starting with a. Now $(p_1, \vec{\phi}) \not\trianglerighteq_{\mathscr{S}(R)} (p_2, \vec{\phi})$, for $(p_1, \vec{\phi})$ has an a-transition, whereas $(p_2, \vec{\phi})$ has no such transition by Lemma 2.23;

or q_2 has $n+1$ syntactic summands starting with a, say $ar_0, \ldots, ar_n$. It holds that $r_i \neq r$ for all $i = 0, \ldots, n$ (otherwise $ar = ar_i \subseteq q_2$ for some i). By the induction hypothesis $(r, \vec{\psi}_i) \not\trianglerighteq_{\mathscr{S}(R)} (r_i, \vec{\psi}_i)$ for a data-state $\vec{\psi}_i \in \mathscr{S}(R)$. By Lemma 2.23 we have for all $i, j = 0, \ldots, n$ $(p_1, \vec{\phi}) \xrightarrow{a} (\epsilon r, \vec{\psi}_i)$ and $(p_2, \vec{\phi}) \xrightarrow{a} (\epsilon r_j, \vec{\psi}_i)$. Suppose $(p_1, \vec{\phi}) \trianglerighteq_{\mathscr{S}(R)} (p_2, \vec{\phi})$, then by definition of global $\mathscr{S}$-bisimilarity $(\epsilon r, \vec{\psi}_i) \trianglerighteq_{\mathscr{S}(R)} (\epsilon r_j, \vec{\psi}_i)$ for all i, j, and hence

$$(r, \vec{\psi}_i) \trianglerighteq_{\mathscr{S}(R)} (r_j, \vec{\psi}_i).$$

But this was contradictory in case $i = j$.

The case $p_2 \not\subseteq p_1$ can be treated likewise. □

By this lemma, the previous completeness results and Theorem 4.1. we obtain the following results.

Corollary 4.9. *(Completeness) Let r_1, r_2 be closed terms over $\Sigma(\mathrm{ACP}_G)$.*

1. *If $r_1 \trianglerighteq_{\mathscr{S}} r_2$ for all data environments $\mathscr{S}$, then $\mathrm{ACP}_G \vdash r_1 = r_2$.*
2. *If $r_1 \trianglerighteq_{\mathscr{S}} r_2$ for all data environments $\mathscr{S}$, then $\mathrm{ACP}_G^4 \vdash_2 r_1 = r_2$.*
3. *Let $\mathscr{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. If $r_1 \trianglerighteq_{\mathscr{S}} r_2$, then $\mathrm{ACP}_G(\mathscr{S}) \vdash_2 r_1 = r_2$.*

4.4. An Example: A Parallel Predicate Checker

In this section we illustrate the techniques that we introduced up till now by an example. Let $f \subseteq \mathbb{Z}$ be some predicate, e.g. the set of all primes. Now, given some number n, we want to calculate the smallest $m \geq n$ such that $f(m)$. Assume we have two devices P_1 and P_2 that can calculate for some given number k whether $f(k)$ holds. In Fig. 10 we depict a system that enables us to calculate m using both P_1 and P_2. A Generator/Collector G generates numbers $n, n+1, n+2, \ldots$, sends them to P_1 or P_2, and collects their answers. Furthermore G selects the smallest number satisfying f from the answers and presents it to the environment.

To describe this situation, we extend the example of section 3.3 with the atomic actions $(i = 1, 2)$:

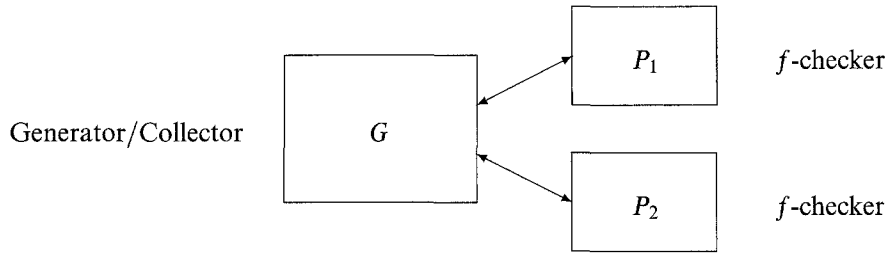


Fig. 10. The parallel predicate checker Q.

$s(!x)$	send value of x,
$r(?x_i)$	read a value for x_i,
$s_{ok}(!x_i)$	send the value x_i for which the evaluation of $f(x_i)$ was a success,
$r_{ok}(?y)$	read a value for y for which $f(y)$ succeeded,
s_{notok}	indicate that an evaluation of f was not successful,
r_{notok}	read that an evaluation of f has failed,
c_{notok}	a communication between r_{notok} and s_{notok},
$w(!x)$, $w(!y)$	write value of x, y to environment.

These atomic actions communicate according to the following scheme:

$$\gamma(s(!x), r(?x_i)) = \gamma(r(?x_i), s(!x)) = [x_i := x],$$
$$\gamma(s_{ok}(!x_i), r_{ok}(?y)) = \gamma(r_{ok}(?y), s_{ok}(!x_i)) = [y := x_i],$$
$$\gamma(s_{notok}, r_{notok}) = \gamma(r_{notok}, s_{notok}) = c_{notok}.$$

All new atomic actions do not change the data-state, i.e. for each new atomic action a:

$$\mathit{effect}(a, \rho) = \{\rho\}.$$

Probably, one would expect that for instance $\mathit{effect}(r(?y), \rho) = \{\rho[\mathit{new\ value}/y]\}$ as $r(?y)$ reads a new value for y. But this need not be so: the value of y is only changed if a communication takes place.

Add new atomic guards $\langle f(t)\rangle$ for any integer expression t to the setting of section 3.3. These guards have their obvious interpretation: $\mathit{test}(\langle f(t)\rangle, \rho)$ holds iff $f([\![\rho(t)]\!])$ holds.

The parallel predicate checker Q can now be specified by:

$$\begin{aligned} G &= [x := n]\, s(!x)\, [x := x+1]\, s(!x)\, G_1 \\ G_1 &= r_{notok}\, [x := x+1]\, s(!x)\, G_1 + r_{ok}(?y)\, G_2 \\ G_2 &= \neg\langle x = y\rangle\, w(y) + \langle x = y\rangle (r_{ok}(?y)\, w(y) + r_{notok}\, w(x)) \end{aligned}$$

$$\begin{aligned} P_i &= r(?x_i)\, P_i' + \epsilon \\ P_i' &= \langle f(x_i)\rangle s_{ok}(!x_i) + \neg\langle f(x_i)\rangle\, s_{notok}\, P_i + \epsilon \end{aligned}$$

$$Q = \partial_H(G \parallel (P_1 \parallel P_2))$$

with $H = \{r(?x_i), r_{ok}(?y), r_{notok}, s(!x), s_{ok}(!x_i), s_{notok} \mid i = 1, 2\}$.

The parallel predicate checker Q is correct if directly before the execution of an atomic action $w(x)$ or $w(y)$, x respectively y represents the smallest number $m \geq n$ such that $f(m)$. We introduce new atomic guards $\langle \alpha(t,u) \rangle$ for integer expressions t, u to express this formally:

$$test(\langle \alpha(t,u) \rangle, \rho) \iff [\![\rho(t)]\!] \leq [\![\rho(u)]\!] \wedge \Big(\bigwedge_{\substack{n \leq j < [\![\rho(u)]\!] \\ j \neq [\![\rho(t)]\!]}} \neg f(j) \Big).$$

Now Q is correct if

$$\mathrm{ACP}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash_2 Q = Q' \tag{3}$$

where Q' is defined by:

$$Q' = \partial_H(G' \parallel (P_1 \parallel P_2))$$

with H, P_1 and P_2 as above, and G' is defined by (the difference between G and G' is underlined):

$$\begin{aligned}
G' &= [x := n]\, s(!x)\, [x := x+1]\, s(!x)\, G'_1 \\
G'_1 &= r_{notok}\, [x := x+1]\, s(!x)\, G'_1 + r_{ok}(?y)\, G'_2 \\
G'_2 &= \neg\langle x = y \rangle \cdot \underline{\langle \alpha(y,y) \rangle \langle f(y) \rangle} \cdot w(y) + \\
&\quad \langle x = y \rangle (r_{ok}(?y) \cdot \underline{\langle \alpha(y,y) \rangle \langle f(y) \rangle} \cdot w(y) + \\
&\quad\quad r_{notok} \cdot \underline{\langle \alpha(x,x) \rangle \langle f(x) \rangle} \cdot w(x)).
\end{aligned}$$

This expresses that Q is correct if we can show that directly before a value, say x, is output via gate w, then f holds for x, and f does not hold for all values from n to up to x (i.e. $\alpha(x,x)$). Note that α is unnecessarily complex to state the correctness of Q. But this formulation is useful in the 'second phase' of the proof of (3).

This proof is given by first expanding Q and Q' to *merge*-free forms (the 'first phase' of the proof of (3)). With ACP_G we derive:

$$\begin{aligned}
Q &= \partial_H(G_1 \parallel (P_1 \parallel P_2)) \\
&= [x := n]([x_1 := x]\, [x := x+1]\, [x_2 := x] \cdot \partial_H(G_1 \parallel (P'_2 \parallel P'_1)) + \\
&\qquad [x_2 := x]\, [x := x+1]\, [x_1 := x] \cdot \partial_H(G_1 \parallel (P'_1 \parallel P'_2)) \quad)
\end{aligned}$$

$$\begin{aligned}
&\partial_H(G_1 \parallel (P'_1 \parallel P'_2)) \\
&\quad = \neg\langle f(x_1) \rangle c_{notok}\, [x := x+1]\, [x_1 := x] \cdot \partial_H(G_1 \parallel (P'_1 \parallel P'_2)) + \\
&\quad\quad \neg\langle f(x_2) \rangle c_{notok}\, [x := x+1]\, [x_2 := x] \cdot \partial_H(G_1 \parallel (P'_2 \parallel P'_1)) + \\
&\quad\quad \langle f(x_1) \rangle\, [y := x_1]\, \partial_H(G_2 \parallel P'_2) + \\
&\quad\quad \langle f(x_2) \rangle\, [y := x_2]\, \partial_H(G_2 \parallel P'_1)
\end{aligned}$$

$$\begin{aligned}
&\partial_H(G_2 \parallel P'_2) \\
&\quad = \neg\langle x = y \rangle\, w(y) + \\
&\quad\quad \langle x = y \rangle (\langle f(x_2) \rangle\, [y := x_2]\, w(y) + \neg\langle f(x_2) \rangle\, c_{notok}\, w(x))
\end{aligned}$$

$$\begin{aligned}
&\partial_H(G_2 \parallel P'_1) \\
&\quad = \neg\langle x = y \rangle\, w(y) + \\
&\quad\quad \langle x = y \rangle (\langle f(x_1) \rangle\, [y := x_1]\, w(y) + \neg\langle f(x_1) \rangle\, c_{notok}\, w(x)).
\end{aligned}$$

Now replacing
- $\partial_H(G_1 \parallel (P_1 \parallel P_2))$ by R,
- both $\partial_H(G_1 \parallel (P_2' \parallel P_1'))$ and $\partial_H(G_1 \parallel (P_1' \parallel P_2'))$ by R_1 (note that $\partial_H(G_1 \parallel (P_2' \parallel P_1')) \overset{\text{EM1,5}}{=} \partial_H(G_1 \parallel (P_1' \parallel P_2'))$),
- $\partial_H(G_2 \parallel P_2')$ by R_2, and
- $\partial_H(G_2 \parallel P_1')$ by R_3

yields the recursive specification of a process R over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ (and indeed over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$!) such that

$$\mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash Q = R. \tag{4}$$

Let the process R' be defined like R, except that $w(x)$ is replaced by $\langle \alpha(x,x)\rangle \langle f(x)\rangle w(x)$ and $w(y)$ by $\langle \alpha(y,y)\rangle \langle f(y)\rangle w(y)$. It can be proved in a similar way that

$$\mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash Q' = R'. \tag{5}$$

This concludes the 'first phase' results of our proof.

In order to show that

$$\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash R = R'$$

(the 'second phase' result needed) the following instances of SI, WPC1 and WPC2 are needed in addition to those given in section 3.3. Let F be some function on integer expressions.

$$\begin{aligned}
&\phi\, c_{notok}\, \phi = \phi\, c_{notok} \text{ for all } \phi \in G,\\
&\neg\langle t = t\rangle = \delta,\\
&\langle t = u\rangle \neg\langle u = t\rangle = \delta,\\
&\langle t = u\rangle \langle u = v\rangle \neg\langle t = v\rangle = \delta,\\
&\langle t_1 = u_1\rangle \cdots \langle t_k = u_k\rangle \neg\langle F(t_1,\ldots,t_k) = F(u_1,\ldots,u_k)\rangle = \delta,\\
&\langle t+1 = u\rangle \langle t = u\rangle = \delta,\\
&\neg\langle f(t)\rangle \langle \alpha(t,u)\rangle \neg\langle \alpha(u,u+1)\rangle = \delta,\\
&\neg\langle f(t)\rangle \langle \alpha(u,t)\rangle \neg\langle \alpha(u,t+1)\rangle = \delta,\\
&\langle \alpha(t,u-1)\rangle \langle t = u\rangle = \delta.
\end{aligned}$$

Note that these identities are valid. Let

$$\beta \overset{\text{def}}{=} \neg\langle x_1 = x_2\rangle(\langle \alpha(x_1,x_2)\rangle \langle x = x_2\rangle + \langle \alpha(x_2,x_1)\rangle\langle x = x_1\rangle).$$

It is easy to show that

$$R\,,\ \beta R_1\,,\ \langle y = x_1\rangle\langle f(x_1)\rangle\beta R_2\,,\ \langle y = x_2\rangle\langle f(x_2)\rangle\beta R_3$$

and

$$R'\,,\ \beta R_1'\,,\ \langle y = x_1\rangle\langle f(x_1)\rangle\beta R_2'\,,\ \langle y = x_2\rangle\langle f(x_2)\rangle\beta R_3'$$

are solutions for T, T_1, T_2 and T_3, respectively, in the following specification:

$$
\begin{aligned}
T &= [x := n]([x_1 := x]\,[x := x+1]\,[x_2 := x]\cdot T_1 + \\
&\qquad [x_2 := x]\,[x := x+1]\,[x_1 := x]\cdot T_1 \quad)
\end{aligned}
$$

$$
\begin{aligned}
T_1 &= \beta(\neg\langle f(x_1)\rangle c_{notok}\,[x := x+1]\,[x_1 := x]\cdot T_1 + \\
&\qquad \neg\langle f(x_2)\rangle c_{notok}\,[x := x+1]\,[x_2 := x]\cdot T_1 + \\
&\qquad \langle f(x_1)\rangle\,[y := x_1]\,T_2 + \\
&\qquad \langle f(x_2)\rangle\,[y := x_2]\,T_3
\end{aligned}
$$

$$
\begin{aligned}
T_2 &= \langle y = x_1\rangle\langle f(x_1)\rangle\beta(\neg\langle x = y\rangle\, w(y) + \\
&\qquad\qquad \langle x = y\rangle(\ \langle f(x_2)\rangle\,[y := x_2]\,w(y) + \\
&\qquad\qquad\qquad \neg\langle f(x_2)\rangle\, c_{notok}\, w(x)\ \ ))
\end{aligned}
$$

$$
\begin{aligned}
T_3 &= \langle y = x_2\rangle\langle f(x_2)\rangle\beta(\neg\langle x = y\rangle\, w(y) + \\
&\qquad\qquad \langle x = y\rangle(\ \langle f(x_1)\rangle\,[y := x_1]\,w(y) + \\
&\qquad\qquad\qquad \neg\langle f(x_1)\rangle\, c_{notok}\, w(x)\ \ ))
\end{aligned}
$$

and thus $\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash R = R'$. Using (4) and (5) above it follows that

$$\mathrm{ACP}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash_2 Q = Q'$$

as was to be proved.

5. Partial Correctness and Hoare Logic

In this section we show that we can capture Hoare logic for process terms [Pon91] in the algebraic framework developed thus far. We consider partial correctness formulas of the form $\{\alpha\}\,p\,\{\beta\}$, where p is a closed term over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ and α, β are guards over $\Sigma(\mathrm{ACP}_G)$. It turns out that the validity of partial correctness formulas can be elegantly expressed with $\mathscr{S}$-bisimulation equivalence: $\{\alpha\}\,p\,\{\beta\}$ is valid in $\mathscr{S}$ iff $\alpha p \leftrightarrows_{\mathscr{S}} \alpha p \beta$. We further show a soundness result for a Hoare logic for linear processes over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ by translating proofs in Hoare logic into process algebra proofs.

5.1. Hoare Logic for Process Terms

Hoare logic is meant for proving the correctness of programs that transform some input into some output. Proof systems are mostly given in a natural deduction format (see e.g. [Dal83] for 'natural deduction') and are parameterised with

1. A class of *programs*, and
2. A language of *assertions* to express correctness properties of programs (usually some first-order language with equality).

In general a *partial correctness formula* has the syntax

$$\{\mathtt{pre}\}\ \mathtt{P}\ \{\mathtt{post}\}$$

where `pre`, `post` are assertions and `P` is a program. The intuitive meaning of $\{\mathtt{pre}\}\ \mathtt{P}\ \{\mathtt{post}\}$ is that whenever the assertion `pre` holds *before* the execution of `P` and `P` terminates, then the assertion `post` holds after the execution of `P`.

Given a set A of atomic actions and a set G_{at} of atomic guards, we here consider the guards over $\Sigma(\mathrm{ACP}_G)$ as a language of assertions, and we take the closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ as the class of programs.

With respect to data-state transformations there are hardly any constraints on the way we provide process terms with an (operational) semantics. Therefore this instantiation is on a rather abstract level, and is suitable to express many programming primitives and constructs (cf. the examples in sections 2, 3.3. and 4.4.). We only require that data environments that are sufficiently deterministic and that weakest preconditions are expressible. These restrictions often occur in some related form in the study of Hoare logic (cf. [Bak80, Apt81]).

5.2. Partial Correctness Formulas and Bisimulation

We now present formal definitions for the interpretation of partial correctness formulas and assertions in any data environment. The main work is already done in section 4, where the operational semantics for the closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ was defined. Let $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ be some data environment. In this section we use the transition relation $\longrightarrow_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}},\mathscr{S}}$ as defined in section 4.2. which is here simply written as $\longrightarrow$.

The interpretation of basic guards is such that a basic guard ϕ holds in $s \in S$ iff

$$(\phi, s) \xrightarrow{\surd} (\delta, s).$$

We define the interpretation of our assertions in $\mathscr{S}$ using $\surd$-transitions.

Definition 5.1. Let α be an assertion and $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ some data environment.

1. The assertion α *holds in* $s \in S$, notation $\mathscr{S} \models \alpha[s]$, iff $(\alpha, s) \xrightarrow{\surd} (\delta, s)$.
2. The assertion α *is valid in* $\mathscr{S}$, notation $\mathscr{S} \models \alpha$, iff $\forall s \in S\ (\mathscr{S} \models \alpha[s])$. □

In order to define the interpretation of partial correctness formulas , we introduce sequences of transitions. Let A^* be the set of finite strings over A, with typical elements $\sigma, \sigma', \ldots$ and λ denoting the empty string. We define for all $\sigma \in A^*$ relations $\xrightarrow{\sigma}\!\!\!\!\rightarrow$ and $\xrightarrow{\sigma}\!\!\!\!\!\surd$ that describe sequences of transitions:

- $(x, s) \xrightarrow{\lambda}\!\!\!\!\rightarrow (x, s)$
- $$\frac{(x, s) \xrightarrow{\sigma}\!\!\!\!\rightarrow (x', s') \quad (x', s') \xrightarrow{a} (x'', s'')}{(x, s) \xrightarrow{\sigma a}\!\!\!\!\rightarrow (x'', s'')} \qquad (a \in A_{\surd})$$

Now the interpretation of a partial correctness formula in $\mathscr{S}$ is defined as follows:

Definition 5.2. A partial correctness formula $\{\alpha\}\ p\ \{\beta\}$ *is valid in* $\mathscr{S}$, notation $\mathscr{S} \models \{\alpha\}\ p\ \{\beta\}$, iff for all $s \in S$ and all $\sigma \in A^*$:

$$\mathscr{S} \models \alpha[s] \text{ and } (p, s) \xrightarrow{\sigma}\!\!\!\!\!\surd (p', s') \implies \mathscr{S} \models \beta[s'].$$ □

We show that for any partial correctness formula $\{\alpha\}p\{\beta\}$ it holds that $\mathscr{S} \models \{\alpha\}p\{\beta\}$ iff $\alpha p \leftrightarrows_{\mathscr{S}} \alpha p \beta$. This alternative characterisation of validity of partial correctness formulas gives us the means to use process algebra for proving partial correctness formulas.

Lemma 5.3. *(Decomposition) Let* $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ *be some data environment.*

For any closed term p over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$, *guard* α *over* $\Sigma(\mathrm{ACP}_G)$ *and* $\sigma \in (A \cup \{\surd\})^*$ *the following properties hold:*

1. *If* $(\alpha p, s) \xrightarrow{\sigma}\!\!\!\!\rightarrow (p', s')$ *and* $\sigma \not\equiv \lambda$, *then* $(\alpha, s) \xrightarrow{\surd} (\delta, s)$ *and* $(p, s) \xrightarrow{\sigma}\!\!\!\!\rightarrow (p', s')$.
2. *If* $(p\alpha, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$, *then* $(p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$ *and* $(\alpha, s') \xrightarrow{\surd} (\delta, s')$.

Proof. By induction on the length of σ (first proving some intermediate properties of sequences of non-terminating transitions). □

Lemma 5.4. *Let p be a closed term over* $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$, α *some guard over* $\Sigma(\mathrm{ACP}_G)$ *and* $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ *a data environment. Then the following statements are equivalent:*

1. *For all* $s \in S$ *and* $\sigma \in A^*$ *it holds that*
$$(p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s') \implies (\alpha, s') \xrightarrow{\surd} (\delta, s'),$$
2. $p \leftrightarrows_{\mathscr{S}} p\alpha$.

Proof. First observe that if $(p, s) \xrightarrow{\surd} (p', s')$, then $p' \equiv \delta$ and $s' \equiv s$.

$1 \implies 2$. Fix some $s \in S$ and take

$$R \stackrel{\mathrm{def}}{=} \{((\delta, s'), (\delta, s')) \mid s' \in S\} \cup \{((r, s'), (r\alpha, s')) \mid (p, s) \xrightarrow{\sigma}\!\!\!\!\rightarrow (r, s') \text{ for some } \sigma \in A^*\}.$$

Note that $(p, s)R(p\alpha, s)$. We show that R is an $\mathscr{S}$-bisimulation. For pairs $((\delta, s'), (\delta, s'))$ it is trivial to check the transfer property. Assume $(q, s')R(q\alpha, s')$.

- Suppose $(q, s') \xrightarrow{a} (q', s'')$ with $a \in A$. We derive $(q\alpha, s') \xrightarrow{a} (q'\alpha, s'')$ and by definition of R also $(q', s'')R(q'\alpha, s'')$.
- Suppose $(q, s') \xrightarrow{\surd} (\delta, s')$, so $(p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (\delta, s')$ for some σ. By assumption we have $(\alpha, s') \xrightarrow{\surd} (\delta, s')$ and derive $(q\alpha, s') \xrightarrow{\surd} (\delta, s')$. By definition $(\delta, s')R(\delta, s')$.
- Suppose $(q\alpha, s') \xrightarrow{a} (q', s'')$ with $a \in A$. By 'decomposition' it follows that $q' \equiv q''\alpha$ and $(q, s') \xrightarrow{a} (q'', s'')$. By definition $(q'', s'')R(q', s'')$.
- Suppose $(q\alpha, s') \xrightarrow{\surd} (\delta, s')$. It follows that $(q, s') \xrightarrow{\surd} (\delta, s')$ and $(\alpha, s') \xrightarrow{\surd} (\delta, s')$. By definition $(\delta, s')R(\delta, s')$.

$2 \implies 1$. Suppose $(p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$ for some $\sigma \in A^*$. By assumption then also $(p\alpha, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$, and by decomposition we have $(\alpha, s') \xrightarrow{\surd} (\delta, s')$. □

Now we can easily prove the following characterisation of the $\mathscr{S}$-validity of partial correctness formulas in terms of $\mathscr{S}$-bisimilarity.

Theorem 5.5. *Let p be a closed term over* $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$, α, β *guards over* $\Sigma(\mathrm{ACP}_G)$ *and* $\mathscr{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ *a data environment. Then*

$$\mathscr{S} \models \{\alpha\}\, p\, \{\beta\} \iff \alpha p \leftrightarrows_{\mathscr{S}} \alpha p \beta.$$

Proof.

$\implies$ Suppose $\mathscr{S} \models \{\alpha\}\, p\, \{\beta\}$. By the previous lemma it is sufficient to show that if $(\alpha p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$, then $(\beta, s') \xrightarrow{\surd} (\delta, s')$. So let $(\alpha p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$. By 'decomposition' we have $(\alpha, s) \xrightarrow{\surd} (p', s)$ and $(p, s) \xrightarrow{\sigma\surd}\!\!\!\!\rightarrow (p', s')$. By $\mathscr{S} \models \{\alpha\}\, p\, \{\beta\}$ this implies $(\beta, s') \xrightarrow{\surd} (p', s')$.

$\impliedby$ Suppose $\mathscr{S} \not\models \{\alpha\}\, p\, \{\beta\}$, so for some $s \in S$ and $\sigma \in A^*$:

$(\alpha, s) \xrightarrow{\surd} (\delta, s)$ and $(p, s) \xrightarrow{\sigma} (p', s')$ and $\mathcal{S} \not\models \beta[s']$.

We derive $(\alpha p, s) \xrightarrow{\sigma} (p', s')$ and by the assumption of $\mathcal{S}$-bisimilarity we have $(\alpha p \beta, s) \xrightarrow{\sigma} (p', s')$. By 'decomposition' this implies that $(\beta, s') \xrightarrow{\surd} (\delta, s')$, which contradicts the supposition. □

5.3. A Proof System for Deriving Partial Correctness Formulas

In this section we present a proof system H in a natural deduction format for deriving partial correctness formulas over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ (cf. [Pon91]). The proof system H is displayed in Fig. 11. Notice that the rules of H refer to terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ that need not be closed. Let Γ be a set of assertions and partial correctness formulas. We write $\Gamma \vdash_H \{\alpha\}\, t\, \{\beta\}$ iff we can derive $\{\alpha\}\, t\, \{\beta\}$ in H using elements of Γ as axioms.

- The axiom scheme H1 introduces partial correctness formulas over atomic actions. It only makes sense if weakest preconditions are expressible, and it is only valid in data environments that are sufficiently deterministic. Weakest preconditions are defined in Definition 3.1. and Remark 3.4.
- The axiom scheme H2 introduces partial correctness formulas over basic guards.
- Rules H3 and H4 express how the operators $+$ and $\cdot$ may be introduced in partial correctness formulas.
- Rule H5, *consequence*, is a standard proof rule in Hoare logic. The intended interpretation of an expression $\alpha \to \beta$ is as expected: $\mathcal{S} \models (\alpha \to \beta)[s]$ iff $\mathcal{S} \models \alpha[s] \Longrightarrow \mathcal{S} \models \beta[s]$.
- Rule H6, an instance of *Scott's induction rule* (see e.g. [Bak80, Apt81]), is suitable to derive partial correctness formulas with recursive terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$. This rule allows *cancellation* of hypotheses, indicated by the square brackets in its premises: let $E = \{x = t_x \mid x \in V_E\}$ be a guarded recursive specification and α_x, β_x $(x \in V_E)$ be guards. If for all $y \in V_E$ we can derive (indicated by the dots in the rule) $\{\alpha_y\}\, t_y\, \{\beta_y\}$ from a set of hypotheses Γ_y containing *no* other partial correctness formulas with free variables in V_E than those in $\{\{\alpha_x\}\, x\, \{\beta_x\} \mid x \in V_E\}$, then for any $z \in V_E$ the partial correctness formula $\{\alpha_z\} \langle z \mid E \rangle \{\beta_z\}$ can be derived from

$$\bigcup_{x \in V_E} \Gamma_x - \{\{\alpha_x\}\, x\, \{\beta_x\} \mid x \in V_E\}.$$

5.4. Soundness of the Proof System

In this section we prove a soundness result for H with respect to a data environment $\mathcal{S} = \langle S, \mathit{effect}, \mathit{test} \rangle$ over A and G such that weakest preconditions are expressible and $\mathcal{S}$ is sufficiently deterministic. Let $Tr_{\mathcal{S}}$ be the set of assertions that are *true* (valid) in $\mathcal{S}$. We prove that

$$Tr_{\mathcal{S}} \vdash_H \{\alpha\}\, p\, \{\beta\} \quad \Longrightarrow \quad \mathcal{S} \models \{\alpha\}\, p\, \{\beta\}$$

provided that recursive specifications have a finite number of equations and are *linear* (cf. *linear* context free grammars [HoU79]):

(H1) $\{wp(a,\alpha)\}\, a\, \{\alpha\}$ if $a \in A$

(H2) $\{\alpha\}\, \phi\, \{\alpha \cdot \phi\}$ if $\phi \in G$

(H3) $$\frac{\{\alpha\}\, t\, \{\beta\} \quad \{\alpha\}\, t'\, \{\beta\}}{\{\alpha\}\, t + t'\, \{\beta\}}$$

(H4) $$\frac{\{\alpha\}\, t\, \{\alpha'\} \quad \{\alpha'\}\, t'\, \{\beta\}}{\{\alpha\}\, t \cdot t'\, \{\beta\}}$$

(H5) $$\frac{\alpha \to \alpha' \quad \{\alpha'\}\, t\, \{\beta'\} \quad \beta' \to \beta}{\{\alpha\}\, t\, \{\beta\}}$$

(H6) For $E = \{x = t_x \mid x \in V_E\}$ a *guarded* recursive specification:

$$\frac{\begin{array}{c}[\{\{\alpha_x\}\, x\, \{\beta_x\} \mid x \in V_E\}] \\ \vdots \\ \{\alpha_y\}\, t_y\, \{\beta_y\} \qquad \text{for all } y \in V_E\end{array}}{\{\alpha_z\} <z \mid E> \{\beta_z\}} \quad z \in V_E$$

Fig. 11. The proof system H ($a \in A, \phi \in G$).

Definition 5.6. A process term t over $\Sigma(\mathrm{BPA}_G)$ is called *linear* over $V' \subseteq V$ iff

$$t ::= p \mid x \mid pt \mid tp \mid t + t$$

where p is a closed term over $\Sigma(\mathrm{BPA}_G)$ and $x \in V'$. A recursive specification $E = \{x = t_x \mid x \in V_E\}$ is *linear* iff the terms t_x are linear over V_E. □

In [Pon91] only processes definable by regular recursion were considered in the context of H6. This class is strictly contained in the class of processes definable by guarded, linear recursion.

By Lemma 5.4. and the soundness of $\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP}$, the soundness of H follows from the statement

$$Tr_{\mathscr{S}} \vdash_H \{\alpha\}\, p\, \{\beta\} \implies \mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash \alpha p = \alpha p \beta.$$

In the rest of this section we prove this statement by translating H-derivations in a canonical way to proofs in process algebra.

We first show that H is sound for the (recursion-free) terms over $\Sigma(\mathrm{BPA}_G)$.

Lemma 5.7. *(Soundness of H for recursion-free terms) Let p be a closed term over* $\Sigma(\mathrm{BPA}_G)$ *and* α, β *guards over* $\Sigma(\mathrm{BPA}_G)$. *Then*

$$Tr_{\mathscr{S}} \vdash_H \{\alpha\}\, p\, \{\beta\} \implies \mathrm{BPA}_G(\mathscr{S}) \vdash \alpha p = \alpha p \beta.$$

Proof. By induction on the length of derivations. The soundness of H1 – H4 is straightforward. We only show that rule H5 (*consequence*) is sound (we need not consider rule H6, as this rule introduces recursively defined processes). Rule H5 contains expressions of the form $\alpha \to \beta$ with the interpretation $\mathscr{S} \models (\alpha \to \beta)[s]$

iff $\mathscr{S} \models \alpha[s] \Longrightarrow \mathscr{S} \models \beta[s]$. It is easy to show that such expressions can be algebraically characterised as follows:

$$\alpha \to \beta \in Tr_{\mathscr{S}} \iff \mathrm{BPA}_G(\mathscr{S}) \vdash \alpha \cdot \beta = \alpha.$$

Assume

$$Tr_{\mathscr{S}} \vdash_H \{\alpha'\}\, p\, \{\beta'\} \text{ and } \alpha \to \alpha', \beta' \to \beta \in Tr_{\mathscr{S}}.$$

By induction we can prove $\alpha' p = \alpha' p \beta'$, $\alpha\alpha' = \alpha$ and $\beta'\beta = \beta'$ in $\mathrm{BPA}_G(\mathscr{S})$. We derive

$$\begin{aligned} \alpha p &= \alpha\alpha' p \\ &= \alpha\alpha' p\beta' \\ &= \alpha\alpha' p\beta'\beta \\ &= \alpha\alpha' p\beta \\ &= \alpha p\beta \end{aligned}$$

as was to be shown. □

Using this fact we can prove a general result concerning *linear* terms that connects H-derivability from $Tr_{\mathscr{S}}$ to provable equality in $\mathrm{BPA}_G(\mathscr{S})$.

Lemma 5.8. *Let $t(x_1,\ldots,x_n)$ be a term over $\Sigma(\mathrm{BPA}_G)$ and $\alpha, \beta, \alpha_i, \beta_i$ be guards over $\Sigma(\mathrm{BPA}_G)$ for $i = 1,\ldots,n$. If $t(x_1,\ldots,x_n)$ is linear over $\{x_1,\ldots,x_n\}$, and*

$$Tr_{\mathscr{S}}, \{\{\alpha_i\}\, x_i\, \{\beta_i\} \mid i = 1,\ldots,n\} \vdash_H \{\alpha\}\, t(x_1,\ldots,x_n)\, \{\beta\},$$

then

1. $\mathrm{BPA}_G(\mathscr{S}) \vdash \alpha \cdot t(\alpha_1 x_1,\ldots,\alpha_n x_n) = \alpha \cdot t(x_1,\ldots,x_n)$,
2. $\mathrm{BPA}_G(\mathscr{S}) \vdash \alpha \cdot t(x_1\beta_1,\ldots,x_n\beta_n) = \alpha \cdot t(x_1\beta_1,\ldots,x_n\beta_n) \cdot \beta$.

Proof. By induction on the length of the derivation of

$$Tr_{\mathscr{S}}, \{\{\alpha_i\}\, x_i\, \{\beta_i\} \mid i = 1,\ldots,n\} \vdash_H \{\alpha\}\, t(x_1,\ldots,x_n)\, \{\beta\}.$$

The cases in which one of H1 – H3 is applied last are straightforward. We give a proof for the cases in which H4 or H5 is applied last (note that by definition of linearity rule H6 of H again need not be considered):

As for H4. Because all terms in the proof are linear, we may assume that $t(x_1,\ldots,x_n) \equiv p \cdot u(x_1,\ldots,x_n)$ or $t(x_1,\ldots,x_n) \equiv u(x_1,\ldots,x_n) \cdot p$, with p a closed term over $\Sigma(\mathrm{BPA}_G)$. Let $t(x_1,\ldots,x_n) \equiv p \cdot u(x_1,\ldots,x_n)$ and

$$\dfrac{\begin{array}{c} Tr_{\mathscr{S}}, \{\{\alpha_i\}\, x_i\, \{\beta_i\} \mid i = 1,\ldots,n\} \\ \vdots \\ \{\alpha\}\, p\, \{\alpha'\} \quad \{\alpha'\}\, u(x_1,\ldots,x_n)\, \{\beta\} \end{array}}{\{\alpha\}\, p \cdot u(x_1,\ldots,x_n)\, \{\beta\}}$$

Apparently $Tr_{\mathscr{S}} \vdash_H \{\alpha\}\, p\, \{\alpha'\}$, so we have by Lemma 5.7 that $\mathrm{BPA}_G(\mathscr{S}) \vdash \alpha p = \alpha p \alpha'$. We derive

$$
\begin{array}{lrcl}
1. & \alpha p \cdot u(\alpha_1 x_1, \ldots, \alpha_n x_n) & = & \alpha p \alpha' \cdot u(\alpha_1 x_1, \ldots, \alpha_n x_n) \\
 & & \overset{IH}{=} & \alpha p \alpha' \cdot u(x_1, \ldots, x_n) \\
 & & = & \alpha p \cdot u(x_1, \ldots, x_n). \\
 & & & \\
2. & \alpha p \cdot u(x_1\beta_1, \ldots, x_n\beta_n) & = & \alpha p \alpha' \cdot u(x_1\beta_1, \ldots, x_n\beta_n) \\
 & & \overset{IH}{=} & \alpha p \alpha' \cdot u(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta \\
 & & = & \alpha p \cdot u(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta.
\end{array}
$$

The case in which $t(x_1, \ldots, x_n) \equiv u(x_1, \ldots, x_n) \cdot p$ with p a closed term over $\Sigma(\mathrm{BPA}_G)$ can be proved likewise.

As for H5. Assume

$$
\frac{\begin{array}{c} Tr_{\mathscr{S}}, \{\{\alpha_i\}\, x_i\, \{\beta_i\} \mid i = 1, \ldots, n\} \\ \vdots \\ \alpha \to \alpha' \quad \{\alpha'\}\, t(x_1, \ldots, x_n)\, \{\beta'\} \quad \beta' \to \beta \end{array}}{\{\alpha\}\, t(x_1, \ldots, x_n)\, \{\beta\}}
$$

By induction we have $\mathrm{BPA}_G(\mathscr{S})$-derivations of $\alpha\alpha' = \alpha$ and $\beta'\beta = \beta'$. We derive

$$
\begin{array}{lrcl}
1. & \alpha \cdot t(\alpha_1 x_1, \ldots, \alpha_n x_n) & = & \alpha\alpha' \cdot t(\alpha_1 x_1, \ldots, \alpha_n x_n) \\
 & & \overset{IH}{=} & \alpha\alpha' \cdot t(x_1, \ldots, x_n) \\
 & & = & \alpha \cdot t(x_1, \ldots, x_n). \\
 & & & \\
2. & \alpha \cdot t(x_1\beta_1, \ldots, x_n\beta_n) & = & \alpha\alpha' \cdot t(x_1\beta_1, \ldots, x_n\beta_n) \\
 & & \overset{IH}{=} & \alpha\alpha' \cdot t(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta' \\
 & & = & \alpha\alpha' \cdot t(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta'\beta \\
 & & \overset{IH}{=} & \alpha\alpha' \cdot t(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta \\
 & & = & \alpha \cdot t(x_1\beta_1, \ldots, x_n\beta_n) \cdot \beta.
\end{array}
$$

□

This result can be used to show the soundness of the proof system H for the following subset of terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$.

Theorem 5.9. *(Soundness of H) Let p be a closed term over* $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ *in which all occurrences of the form* $<x|E>$ *refer to a (guarded) recursive specification E over* $\Sigma(\mathrm{BPA}_G)$ *that is linear and contains only finitely many equations. Let* α, β *be guards over* $\Sigma(\mathrm{BPA}_G)$. *Then*

$$
\begin{array}{lcl}
Tr_{\mathscr{S}} \vdash_H \{\alpha\}\, p\, \{\beta\} & \Longrightarrow & \mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash \alpha p = \alpha p \beta \\
 & \Longrightarrow & \alpha p \leftrightarrows_{\mathscr{S}} \alpha p \beta \\
 & \Longleftrightarrow & \mathscr{S} \models \{\alpha\}\, p\, \{\beta\}.
\end{array}
$$

Proof. By Theorems 3.5. and 5.5. we only have to prove the first implication. We apply induction on the length of H-derivations. The proof of the soundness of H1 – H5 is straightforward (cf. the proof of Lemma 5.7.). We only give a proof of the soundness of H6. Let $E = \{x_i = t_i(x_1, \ldots, x_n) \mid i = 1, \ldots, n\}$ be a guarded linear recursive specification and assume

$$
Tr_{\mathscr{S}}, \{\{\alpha_i\}\, x_i\, \{\beta_i\} \mid i = 1, \ldots, n\} \vdash_H \{\alpha_j\}\, t_j(x_1, \ldots, x_n)\, \{\beta_j\}
$$

for $j = 1, \ldots, n$. So we have an H-derivation of the premises of rule H6. We have to show

$$\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash \alpha_j X_j = \alpha_j X_j \beta_j$$

for $j = 1,\ldots,n$ (recall that X_j abbreviates $<x_j \mid E>$, the constant representing a solution for the j_{th} equation of E). In order to do so we use the recursive specifications

$$E' = \{y_i = \alpha_i \cdot t_i(y_1,\ldots,y_n) \mid i = 1,\ldots,n\}$$

$$E'' = \{z_i = \alpha_i \cdot t_i(z_1\beta_1,\ldots,z_n\beta_n) \mid i = 1,\ldots,n\}$$

and show for any $j \in \{1,\ldots,n\}$ that

1. $\alpha_j X_j = Y_j$,
2. $Z_j \beta_j = Z_j$.
3. $Z_j = Y_j$.

As a consequence we can derive

$$\mathrm{BPA}_G(\mathscr{S}) + \mathrm{REC} + \mathrm{RSP} \vdash \alpha_j X_j = Y_j = Z_j = Z_j\beta_j = \alpha_j X_j \beta_j$$

as has to be shown. So we are left to prove 1,2 and 3. Observe that E', E'' are guarded linear recursive specifications, so we may use both RSP and the previous Lemma 5.8.

As for 1. We first show that $\alpha_j X_j = Y_j$ for all $j \in \{1,\ldots,n\}$. We derive

$$\alpha_j X_j \overset{\mathrm{REC}}{=} \alpha_j \cdot t_j(X_1,\ldots,X_n) \overset{5.8.1}{=} \alpha_j \cdot t_j(\alpha_1 X_1,\ldots,\alpha_n X_n).$$

So $\alpha_1 X_1,\ldots,\alpha_n X_n$ are solutions of $y_1,\ldots,y_n$ in E'. With RSP we conclude $\alpha_j X_j = Y_j$ for $j = 1,\ldots,n$.

As for 2. We show that $Z_j\beta_j = Z_j$ for all $j \in \{1,\ldots,n\}$. We derive

$$\begin{aligned} Z_j\beta_j &\overset{\mathrm{REC}}{=} \alpha_j \cdot t_j(Z_1\beta_1,\ldots,Z_n\beta_n)\cdot\beta_j \\ &\overset{5.8.2}{=} \alpha_j \cdot t_j(Z_1\beta_1,\ldots,Z_n\beta_n) \\ &= \alpha_j \cdot t_j((Z_1\beta_1)\beta_1,\ldots,(Z_n\beta_n)\beta_n). \end{aligned}$$

So $Z_1\beta_1,\ldots,Z_n\beta_n$ are solutions of $z_1,\ldots,z_n$ in E''. With RSP we conclude $Z_j\beta_j = Z_j$ for all $j \in \{1,\ldots,n\}$.

As for 3. We show (using 2) $Z_j = Y_j$ for all $j \in \{1,\ldots,n\}$ as follows:

$$Z_j \overset{\mathrm{REC}}{=} \alpha_j \cdot t_j(Z_1\beta_1,\ldots,Z_n\beta_n) \overset{2}{=} \alpha_j \cdot t_j(Z_1,\ldots,Z_n).$$

So $Z_1,\ldots,Z_n$ are solutions of $y_1,\ldots,y_n$ in E'. With RSP we conclude $Z_j = Y_j$ for all $j \in \{1,\ldots,n\}$. □

6. Conclusions

In this paper we use an operational semantics for process algebra that combines behavioural and (data-)state based aspects. Typical is the introduction of guards, i.e., predicates over data-states, as a special kind of processes. Thus a *one-sorted* framework is obtained, based on two sets of special constants: atomic actions and (the closure under $\neg$ of) atomic guards. This allows for a relatively simple type of complete axiomatisations, both with respect to a preferred data-state environment

and for a class of such environments. Furthermore this framework is suitable to reason about infinite processes defined by (guarded) recursive equations. Finally, as shown in the previous section, it is possible to deal with the essentials of Hoare Logic for partial correctness in our set-up.

We only know of one other approach in process or programming formalisms that involves guards in a one-sorted way, developed by Manes and Arbib: in [MaA86] guards and functions modelling programs are combined in a partially additive category. (Here the subset of "guard morphisms" forms a Boolean algebra.)

In the following the present work is related to some well-known other approaches, mixing Boolean expressions and behavioural constructs in a *two-sorted* way.

First we make a short comparison with "Process Algebra with Signals and Conditions" of Baeten and Bergstra [BaB90]. Given a Boolean algebra $\mathbb{B}$, the authors discuss three well-known operators that relate $\mathbb{B}$ and the sort of processes under consideration, say $\mathbb{P}$. The first operator is the *conditional*:

$$. \lhd . \rhd . \ : \mathbb{P} \times \mathbb{B} \times \mathbb{P} \to \mathbb{P}$$

that stems from Hoare *et al.* [HHJ87], where $p \lhd b \rhd q$ should be read as **if** b **then** p **else** q **fi**. Next there is the *guarded command*:

$$. :\to . \ : \mathbb{B} \times \mathbb{P} \to \mathbb{P}$$

where the expression $b :\to p$ is to be read as **if** b **then** p **fi**, and which cannot be defined (axiomatically) without the δ, for

$$\textbf{false} :\to x = \delta.$$

Finally *guards* are introduced as unary operators:

$$\{.\} \ : \mathbb{B} \to \mathbb{P}$$

with the same meaning as described in this paper. These 'guards' also presuppose the existence of the ϵ constant, as

$$\{\textbf{true}\} = \epsilon.$$

From a methodological point of view, of these three operators the conditional is regarded as basic in [BaB90] for its (axiomatic) definition does not presuppose any of the special process algebra constants δ or ϵ. This argument is not preserved in our set-up, as δ and ϵ represent in "Process Algebra with Guards" just the minimal generators for any Boolean *sub*algebra to be included. We finally remark that Baeten and Bergstra use 21 axioms to define conditionals and guarded commands over the BPA fragment.

In the paper "Laws of Programming" of Hoare *et al.* [HHJ87] 'programs' are constructed from assignments with operators for sequential composition, conditionals and nondeterminism. The operators are described in an equational style, just as in [BaB90] and as in the present paper. There is a unit program *SKIP* that behaves like our ϵ and a program *ABORT* that is reminiscent to our δ, but that behaves according to Murphy's Law: "If it can go wrong, it will", in our notation:

$$x + ABORT = ABORT$$
$$x \cdot ABORT = ABORT \cdot x = ABORT.$$

This latter program *ABORT* makes a comparison with our approach more difficult. In [HHJ87] there are about 30 laws for a fragment comparable to BPA with guards.

In Dijkstra [Dij76, Apt84] a class of programs is introduced, which contains the following "**if - fi** construct":

$$\textbf{if } e_1 \to S_1 \,\Box\, e_2 \to S_2 \,\Box\, \cdots \,\Box\, e_n \to S_n \textbf{ fi}$$

with $e_1, e_2, \ldots$ Boolean guards and $S_1, S_2, \ldots$ programs. The intuitive meaning of this construct is to choose nondeterministically a guard e_i that holds and to execute the program S_i. In the case that none of the e_i hold, the whole construct deadlocks. The translation of this construct into process algebra with guards would then be:

$$i(\ulcorner e_1 \urcorner \cdot \ulcorner S_1 \urcorner + \ulcorner e_2 \urcorner \cdot \ulcorner S_2 \urcorner + \cdots + \ulcorner e_n \urcorner \cdot \ulcorner S_n \urcorner)$$

with i some (internal) action and $\ulcorner . \urcorner$ denoting the translation. The role of the action i is to ensure deadlock if the construct is placed in a "+ context" and none of the guards holds. It is in this case assumed that i does not transform any data-state. A more precise modelling of this language can be given by replacing i with the constant τ (silent step) from process algebra [BaW90], for example relating **if false**$\to S$ **fi** to $\tau(\delta \cdot \ulcorner S \urcorner)(= \tau \cdot \delta)$.

In [Hen91] Hennessy presents a language and proof system for communicating processes with value-passing. Here also Boolean guards are incorporated in the form of conditionals. There is a completeness result based on the rewriting of terms to guard-free ones (note that this demands a fixed "Boolean expression" semantics).

A bit of a drawback in all these related approaches is the number of axioms and rules necessary to relate Boolean expressions to behavioural constructs (this number may in some cases even increase if completeness results are to be proved), whereas we only need a small number. Of course a general advantage of the 'conditional' or **if - then - else - fi** construct is that it is well-known and established, and therefore probably intuitively more appealing than our guards. Nevertheless we hope to have argued that for analytical purposes, guards as introduced here constitute a simpler and more fundamental approach.

Acknowledgement

We thank Jos Baeten, Jan Bergstra, Frank de Boer, Tony Hoare, Catuscia Palamidessi, Frits Vaandrager, Chris Verhoef, Fer-Jan de Vries and the referees for their constructive and helpful comments.

References

[AuB84] Austry, D. and Boudol, G.: Algèbre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

[Apt81] Apt, K.R.: Ten years of Hoare's logic: a survey – Part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981.

[Apt84] Apt, K.R.: Ten years of Hoare's logic: a survey – Part II; Nondeterminism. *Theoretical Computer Science*, 28:83–109, 1984.

[Bak80] de Bakker, J.W.:. *Mathematical theory of program correctness.* Prentice Hall International, 1980.

[BaB88] Baeten, J.C.M. and Bergstra, J.A.: Global renaming operators in concrete process algebra. *Information and Computation*, 78(3):205–245, 1988.

[BaB90] Baeten, J.C.M. and Bergstra, J.A.: Process algebra with signals and conditions. In M. Broy, editor, *Programming and Mathematical Methods, Proceedings Summer School Marktoberdorf 1991, NATO ASI Series F88*, pages 273–323, Springer-Verlag, 1991.

[BvG87] Baeten, J.C.M. and van Glabbeek, R.J.: Merge and termination in process algebra. In K.V. Nori, editor, *Proceedings 7th 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.

[BeK84a] Bergstra, J.A. and Klop, J.W.: The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings 11th ICALP*, Antwerp, volume 172 of *Lecture Notes in Computer Science*, pages 82–95. Springer-Verlag, 1984.

[BeK84b] Bergstra, J.A. and Klop, J.W.: Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[BeK86] Bergstra, J.A. and Klop, J.W.: 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. First appeared as: Report CS-R8404, CWI, Amsterdam, 1984.

[BKT85] Bergstra, J.A. Klop, J.W. and Tucker, J.V.: 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.

[BaW90] Baeten, J.C.M. and Weijland, W.P.: *Process algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[Dal83] van Dalen, D.: *Logic and Structure.* Springer-Verlag, 1983.

[Dij76] Dijkstra, E.W.: *A Discipline of Programming.* Prentice Hall International, Englewood Cliffs, 1976.

[Gla90] van Glabbeek, R.J.: The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90*, Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[Gla93] van Glabbeek, R.J.: The linear time – branching time spectrum II (the semantics of sequential systems with silent moves). In E. Best, editor, *Proceedings CONCUR 93*, Hildesheim, *Lecture Notes in Computer Science*, Springer-Verlag, to appear.

[vGV89] van Glabbeek, R.J. and Vaandrager, F.W.: Modular specifications in process algebra – with curious queues (extended abstract). In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications, Workshop Passau 1987*, volume 394 of *Lecture Notes in Computer Science*, pages 465–506. Springer-Verlag, 1989.

[Hen91] Hennessy, M.: A proof system for communicating processes with value-passing. *Formal Aspects of Computing*, 3:346–366, 1991.

[HHJ87] Hoare, C.A.R., Hayes, I.J., Jifeng, He., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M. and Sufrin, B.A.: Laws of programming. *Communications of the ACM*, 30(8):672–686, August 1987.

[Hoa69] Hoare, C.A.R.: An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), October 1969.

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

[HoU79] Hopcroft, J.E. and Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[ISO87] ISO. *Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour*, 1987. ISO/TC97/SC21/N DIS8807.

[Lam80] Lamport, L.: The 'Hoare logic' of concurrent programs. *Acta Informatica*, 14:21–37, 1980.

[MaA86] Manes, E.G. and Arbib, M.A.: *Algebraic Approaches to Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[Man74] Manna, Z.: *Mathematical Theory of Computation.* McGraw-Hill Book Co., 1974.

[Mil80] Milner, R.: *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[Mil89] Milner, R.: *Communication and concurrency.* Prentice Hall International, 1989.

[OwG76] Owicki, S. and Gries, D.: An axiomatic proof technique for parallel programs. *Acta Informatica*, pages 319–340, 1976.

[Par81] Park, D.M.R.: Concurrency and automata on infinite sequences. In P. Deussen, editor,

5th *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Plo81] Plotkin, G.D.: A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pon91] Ponse, A.: Process expressions and Hoare's logic. *Information and Computation*, 95(2):192–217, 1991.

[Sio64] Sioson, F.M.: Equational bases of Boolean algebras. *Journal of Symbolic Logic*, 29(3):115–124, September 1964.

[SPE90] SPECS-Semantics and Analysis. *Definition of MR and CRL Version 2.1*. Specification and Programming Environment for Communicating Software (SPECS), RACE Ref: 1046, Report 46/SPE/WP5/DS/A/017/b1, December 1990.

[Sti88] Stirling, C.: A generalization of Owicki-Gries's Hoare logic for a concurrent while-language. *Theoretical Computer Science*, 58:34–359, 1988.

[Vaa89] Vaandrager, F.W.: Specificatie en verificatie van communicatieprotocollen met procesalgebra. Dept. of Computer Science, University of Amsterdam, 1989. Lecture notes, in Dutch.

Received October 1991
Accepted in revised form December 1992 by J. Parrow