Process Algebra and Dynamic Logic

Alban Ponse

University of Amsterdam, Programming Research Group

Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.

E-mail: `alban@fwi.uva.nl`

Abstract

An extension of process algebra is introduced which can be compared to (propositional) dynamic logic. The additional feature is a 'guard' construct, related to the notion of a test in dynamic logic. This extension of process algebra is semantically based on processes that transform data, and its operational semantics is defined relative to a structure describing these transformations via transitions between pairs of a process term and a data-state. The data-states are given by a structure that also defines in which data-states guards hold and how actions (non-deterministically) transform these states. The operational semantics is studied modulo strong bisimulation equivalence. For basic process algebra (without operators for parallelism) a small axiom system is presented which is complete with respect to a general class of data environments. In case a data environment satisfies some expressiveness constraints, (local) bisimilarity can be completely axiomatized by adding three axioms to this system.

Then process algebra with parallelism and guards is introduced. A two-phase calculus is provided that makes it possible to prove identities between parallel processes. Also this calculus is complete. The use of this calculus is demonstrated by an extended example. The last section of the paper consists of a short discussion on the operational meaning of the Kleene star operator.

1987 CR Categories: F.3.1, F.3.2, F.3.3, I.1.3.

1 Introduction

An interesting question is how dynamic logic and process algebra can be integrated. A well-known result along this line stems from HENNESSY and MILNER, who defined a modal logic that characterizes observable equivalence between processes [HM85]. In process algebra with guards a different approach is followed, reminiscent of the semantical setting of Propositional Dynamic Logic (PDL). The present paper outlines this approach.

In the generic sense 'process algebra' denotes an algebraic approach to the study of concurrent processes. Here a process is roughly "the behavior of a digital system", such as the execution of a computer program. The specific process theory introduced in this paper is based on ACP (the Algebra of Communicating Processes), developed by BERGSTRA and KLOP. For an overview of ACP see BAETEN and WEIJLAND in [BW90]. Other common algebraic concurrency theories are CCS (Calculus of Communicating Systems) developed by MILNER [Mil80] and CSP (Communicating Sequential Processes) overviewed by HOARE in [Hoa85].

Typical for the process algebra approach as followed in ACP is that one reasons with terms denoting processes, rather than with formulas expressing properties of a process (or a program) as is done in dynamic logic [Har84, KT90]. This reasoning is equational, and the resulting identities refer to a *behavioral* equivalence: two processes are equal if they cannot be distinguished according to some notion of observability. A well-known behavioral equivalence is *bisimulation* [Par81], which is considered in this paper. A small example: in ACP a standard axiom is

$$x + y = y + x$$

(the + represents choice) which expresses that processes are independent of any ordering. A PDL interpretation of this axiom is

$$\langle x \cup y \rangle p \leftrightarrow \langle y \cup x \rangle p$$

the derivability of which depends on the PDL axiom scheme $\langle x \cup y \rangle p \leftrightarrow \langle x \rangle p \vee \langle y \rangle p$ and propositional tautologies.

Another difference between dynamic logic and process algebra is that infinite behavior is a rather fundamental notion in the latter. A typical example of a (reactive) non terminating process is a communication protocol transmitting data through an unreliable channel, such that (despite the unreliability) no information will get lost. Correctness is then expressed recursively: its characteristic behavior is to transmit any received datum before a next datum can be received, i.e., the process behaves externally as a one-element buffer. Infinite processes can be defined as solutions of systems of recursive equations.

Process algebra with guards can be characterized by two starting points. The first typical ingredient is that processes are considered as having a separate *data-state*, contrary to the usual semantical setting in process algebra. The execution of a process is regarded in terms of atomic actions that transform data-states:

$$(a, s) \xrightarrow{a} (\epsilon, s')$$

where (a, s) represents the atomic action a in *initial* data-state s, the label a represents what single step can be observed, and $s' \in \mathit{effect}(a, s)$ is a data-state that can result from the execution of a (it is demanded that transformation sets of the form $\mathit{effect}(a, s)$ are not empty). The special constant ϵ ("empty process" or "**skip**") represents the possibility to terminate, and is associated with a termination transition characterized by the label $\surd$:

$$(\epsilon, s') \xrightarrow{\surd} (\delta, s')$$

where the special constant δ ("inaction" or "deadlock") indicates that no further activity can be performed. In the sequel a calculus is defined for deriving transitions from compound process terms, determining the operational semantics for process algebra with guards. As an example the process $a \cdot \delta$ (the $\cdot$ represents sequential composition) is able to perform an a-step to a configuration with process term $\epsilon \cdot \delta$ (which equals δ), that in turn allows no termination action. Bisimulation semantics can now be extended to a setting wherein data-states play an explicit role: e.g. the processes a and $a + a \cdot \epsilon$ are bisimilar if considered in *equal* initial date-states: each transition of the one process can be associated with (at least one) of the other process.

A second characteristic of process algebra with guards concerns the extension "with *guards*." Guards are comparable to tests in dynamic logic. Depending on the data-state, a guard can either be transparent such that it can be passed (so behaves like ϵ), or it can block and prevent subsequent processes from being executed (so behaves like δ). Typical for this extension is the one-sortedness: a guard *itself* represents a process. With this construct the guarded commands of DIJKSTRA [Dij76] can be easily expressed, as well as a restricted notion of tests in PDL (in terms of [Har84] comparable with PDL$^{0.5}$, and so called *poor* tests in [KT90]). Guards have several nice properties, e.g. they constitute a Boolean algebra. Furthermore, a partial correctness formula

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

can be expressed by the algebraic equation $\alpha\, p = \alpha\, p\, \beta$ where α and β are guards (cf. [MA86]): this equation expresses that termination of the process p started in an initial data-state satisfying α, cannot be blocked by the postcondition β, i.e., β "holds" in this case. In the related paper [GP90] on process algebra with guards it is shown that Hoare logic for processes defined by linear recursion can be captured in a completely algebraic way (cf. [Pon89]).

Parallel operators fit easily in the process algebra framework. In for instance Hoare logic, parallelism turns out to be rather intricate; proof rules for parallel operators are often substantial [OG76]. In the subsequent approach the difficulties caused by parallel operators in Hoare logic cannot be avoided, but can be dealt with in a simple algebraic way.

The paper is organized as follows. Section 2 concerns a small fragment of process algebra with guards and introduces the fundamentals of the approach. A complete axiomatization of bisimilarity between finite processes with respect to a class of structures is presented, as well as an extended soundness result for processes defined by recursive equations. In Section 3 it is described that in case some expressiveness constraints are satisfied, bisimilarity relative to a single structure can also be completely characterized. Section 4 introduces the technical means to reason about parallel processes. These are illustrated in Section 5 by an extended example on the correctness of a parallel process. The paper is concluded with a short discussion on the operational meaning of the Kleene star operator in Section 6. As to keep the paper short, most proofs are omitted. However, all proofs are spelled out in [GP90].

2 Basic Process Algebra with guards

Syntax and axioms. Basic Process Algebra with guards, notation BPA_G, is parameterized by

1. A non-empty set A of *atomic actions*,
2. A non-empty set G_{at} of *atomic guards* disjunct from $A \cup \{\delta, \epsilon\}$.

Before defining the exact signature of BPA_G, the set of atomic guards is extended to the set G of *basic* guards in the following way. The two constants δ ('deadlock' or 'inaction') and ϵ ('empty process' or '**skip**') are added to G_{at}, and G is obtained by closure under negation, so contains elements ϕ satisfying the BNF clause

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

The signature of BPA_G, notation $\Sigma(\mathrm{BPA}_G)$, is defined by constants $a, b, c, ...$ representing the elements of A and constants $\phi, \psi, ...$ representing the elements of G. Furthermore it contains the binary operators $+$ (choice) and $\cdot$ (sequential composition). In term formation brackets and variables of a set $V = \{x, y, z, ...\}$ are used. The function symbol $\cdot$ is often left out, and brackets are omitted according to the convention that $\cdot$ binds stronger than $+$. Finally, letters $t, t', ...$ are used to denote open terms, and letters $p, q, ...$ denote closed terms representing processes.

The axioms in Table 1 and those of equational logic express the basic identities between terms over $\Sigma(\mathrm{BPA}_G)$. This axiom system is called BPA_G^4. The axioms A1 – A9 are well-known in process algebra ($\mathrm{BPA}_{\delta\epsilon}$, [BW90]). Observe that there is no symmetric variant of the distributive axiom A4: an axiom $x(y + z) = xy + xz$ derives with atomic actions a, b for x, z and the constant δ for y the equation

$$ab = a\delta + ab$$

(use A6), identifying the process ab which is deadlock free with one that can behave as $a\delta$. The axioms G1 – G4 describe the fundamental 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. In the last new axiom G4 the following shorthand is used:

$$x \subseteq y \overset{def}{=} x + y = y \qquad \text{(and } x \supseteq y \overset{def}{=} y \subseteq x\text{)}$$

(this notation is called *summand inclusion*). This axiom can be motivated as follows: a process $a(\phi p + \neg\phi q)$ behaves either like ap or aq, depending on whether ϕ or $\neg\phi$ can be passed in the data-state resulting from the execution of a. As a consequence the process $a(\phi p + \neg\phi q)$ should be a provable summand of $ap + aq$. The atomicity of a in this axiom is necessary. If a is for instance replaced by the term ab, then after a has happened it can be that execution of b yields a data-state where ϕ holds *and* a data-state where $\neg\phi$ holds. Hence $ab(\phi p + \neg\phi q)$ need not be a summand of $abp + abq$. Note that the axiom G4 is not derivable from the first three 'guard'-axioms. The superscript 4 in BPA_G^4 indicates that there are four axioms referring to guards. Not all of these are always considered. In particular the system BPA_G^3, containing all BPA_G^4-axioms except G4 will play a role.

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	$x\epsilon = x$		

Table 1: The axioms of BPA_G^4 where $\phi \in G$ and $a \in A$

Up till now only 'basic' and 'atomic' guards were introduced. *Guards* as such, with typical elements $\alpha, \beta, ...$ are defined as terms over $\Sigma(\mathrm{BPA}_G)$ that contain only basic guards and the sequential and choice operators. The Boolean operator $\neg$ on guards can be defined by the *abbreviations*

$\neg(\alpha\beta)$	for	$\neg\alpha + \neg\beta$
$\neg(\alpha + \beta)$	for	$\neg\alpha\neg\beta.$

For guards there is the following theorem (cf. [Sio64]):

Theorem 2.1. *Let G_{at} be a set of atomic guards.* BPA_G^3 $(= \mathrm{BPA}_G^4 \setminus \mathrm{G4})$ *is an equational basis for the Boolean algebra* $(G_{at}, +, \cdot, \neg)$. □

Specifying processes by recursive equations.

Definition 2.2. A *recursive specification* $E = \{x = t_x \,|\, x \in V_E\}$ *over* the signature $\Sigma(\mathrm{BPA}_G)$ is a set of equations where V_E is a (possibly infinitely) set of (indexed) variables and t_x a term over $\Sigma(\mathrm{BPA}_G)$ such that the variables in t_x are also in V_E. □

A *solution* of a recursive specification $E = \{x = t_x \,|\, x \in V_E\}$ 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. The following syntactical restriction on recursive specifications turns out to enforce unique solutions:

Definition 2.3. Let t be a term over the signature $\Sigma(\mathrm{BPA}_G)$. 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 $E = \{x = t_x \,|\, x \in V_E\}$ be a recursive specification over $\Sigma(\mathrm{BPA}_G)$. The specification E is *guarded* iff all occurrences of variables in the terms t_x are guarded. □

Note that the property "guarded" of a recursive specification has nothing to do with the "guards" that form the main subject of this paper.

Now the signature $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$, containing representations of infinite processes, is defined as follows:

Definition 2.4. The signature $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ is obtained by extending $\Sigma(\mathrm{BPA}_G)$ in the following way: for each guarded specification $E = \{x = t_x \,|\, x \in V_E\}$ over $\Sigma(\mathrm{BPA}_G)$ a set of constants $\{<x\,|\,E>|\ x \in V_E\}$ is added, where the construct $<x\,|\,E>$ denotes the x-component of a solution of E. □

Some more notations: let $E = \{x = t_x \,|\, x \in V_E\}$ be a guarded specification over $\Sigma(\mathrm{BPA}_G)$, and t some term over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$. Then $<t\,|\,E>$ denotes the term in which each occurrence of a variable $x \in V_E$ in t is replaced by $<x\,|\,E>$, e.g. the expression $<aax\,|\,\{x = ax\}>$ denotes the term $aa<x\,|\,\{x = ax\}>$.

For the constants of the form $<x\,|\,E>$ there are two axioms in Table 2. In these axioms the letter E ranges over guarded specifications. The axiom REC states that the constant $<x\,|\,E>$ ($x \in V_E$) is a solution for the x-component of E, so expresses that each guarded recursive system has *at least* one solution for each of its (bounded) variables. The conditional rule RSP (Recursive Specification Principle) expresses that E has *at most* one solution for each of its variables: whenever one can find processes p_x ($x \in V_E$) satisfying the equations of E, notation $E(\vec{p_x})$, then $p_x = <x\,|\,E>$.

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

Table 2: Axioms for guarded recursive specifications

Finally, a convenient notation is to abbreviate $<x \mid E>$ for $x \in V_E$ by X once E is fixed, and to represent E only by its REC instances. The following example shows all notations concerning recursively specified processes, and illustrates the use of REC and RSP.

Example 2.5. Consider the guarded recursive specifications $E = \{x = ax\}$ and $E' = \{y = ayb\}$ over $\Sigma(\mathrm{BPA}_G)$. So by the convention just introduced, E can be represented by $X = aX$. With REC and RSP (and the congruence properties of $=$) one can prove

$$\mathrm{BPA}_G + \mathrm{REC} + \mathrm{RSP} \vdash X = Y.$$

First note that $Xb = aXb$ by REC, so $E(Xb)$ is derivable. Application of RSP yields

$$Xb = X. \tag{1}$$

Moreover, $Xb \overset{\text{REC}}{=} aXb \overset{(1)}{=} aXbb$, and hence $E'(Xb)$ is derivable. A second application of RSP yields $Xb = Y$. Combining this with (1) gives the desired result. □

Semantics. In the set-up of process algebra with guards, a process is considered as having a *data-state*: an atomic action is a (non-deterministic) data-state *transformer* and a guard is a *test* on data-states. The operational semantics is defined relative to a structure over A, G_{at} that defines these three components:

Definition 2.6. A *data environment* $\mathcal{S} = \langle S, \textit{effect, test}\rangle$ over a set A of atomic actions and a set G_{at} of atomic guards is specified by

- A non-empty set S of data-states,
- A function $\textit{effect}: S \times A \to 2^S \setminus \{\emptyset\}$,
- A predicate $\textit{test} \subseteq S \times G_{at}$.

□

Observe that the function *effect* defining the state transformations possibly introduces non-determinism in state transformations. The predicate *test* determines whether an atomic guard holds in some data-state. Whenever $(s, \phi) \in \textit{test}$, this means that in data-state s the atomic guard ϕ may be passed. In order to interpret basic guards, the predicate *test* is extended in the obvious way:

- for all $s \in S$ it holds that $(s, \epsilon) \in \textit{test}$ and that $(s, \delta) \notin \textit{test}$,
- for all $s \in S$ and $\phi \in G$ it holds that $(s, \neg\phi) \in \textit{test}$ iff $(s, \phi) \notin \textit{test}$.

Processes are provided with an operational semantics in the style of Plotkin [Plo81]. The behavior of a process p is defined by transitions between *configurations*.

Definition 2.7. Let S be a set of data-states. A *configuration* (p, s) *over* $(\Sigma(\mathrm{BPA}_G), S)$ is a pair containing a closed term p over $\Sigma(\mathrm{BPA}_G)$ and a data-state $s \in S$. The set of all configurations over $(\Sigma(\mathrm{BPA}_G), S)$ is denoted by $C(\Sigma(\mathrm{BPA}_G), S)$. □

Let $A_{\surd} \overset{\textit{def}}{=} A \cup \{\surd\}$. The transition relation

$$\longrightarrow_{\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}},\mathcal{S}} \;\subseteq\; C(\Sigma(\mathrm{BPA}_G), S) \times A_{\surd} \times C(\Sigma(\mathrm{BPA}_G), S)$$

contains exactly all transitions between the configurations over $(\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, S)$ that are derivable with the rules for ϕ, a, $+$, $\cdot$ and recursion in Table 5 (see Section 4). The $\surd$-transitions signal termination of a process. The possible behavior associated to a term p in initial data-state s is captured by all transitions reachable from (p, s) in $\longrightarrow_{\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}},\mathcal{S}}$.

The operational behavior embodied by such transitions can be characterized by *bisimulation equivalence* [Par81]. But following the traditional approach in semantics based on data-state transformations, processes with different data-states in their configurations are not compared with each other. To that end the standard notion of bisimilarity is adapted as follows:

Definition 2.8. Let $\mathcal{S}$ be a data environment with data-state space S. A binary relation $R \subseteq C(\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, S) \times C(\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, S)$ is an $\mathcal{S}$*-bisimulation* iff R satisfies the transfer property, i.e. for all $(p, s), (q, s) \in C(\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}, S)$ with $(p, s)R(q, s)$:

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

2. Whenever $(q,s)\xrightarrow{a}_{\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}},\mathcal{S}}(q',s')$ for some a and (q',s'), then, for some p', also $(p,s)\xrightarrow{a}_{\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}},\mathcal{S}}(p',s')$ and $(p',s')R(q',s')$.

Two closed terms p,q over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ are *$\mathcal{S}$-bisimilar*, notation $p \leftrightarrows_{\mathcal{S}} q$, iff for all $s \in S$ there is some $\mathcal{S}$-bisimulation R such that $(p,s)R(q,s)$. □

The following lemma allows reasoning about bisimilarity in an algebraic way, and is crucial for the next two results.

Lemma 2.9. *For any data environment $\mathcal{S}$ the relation $\leftrightarrows_{\mathcal{S}}$ between closed terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ is a congruence with respect to the operators of $\Sigma(\mathrm{BPA}_G)$.* □

Theorem 2.10. (Soundness) *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 $p\leftrightarrows_{\mathcal{S}} q$ for any data environment $\mathcal{S}$.* □

Theorem 2.11. (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 $\mathrm{BPA}_G^4\vdash r_1=r_2$.* □

3 BPA_G in a specific data environment

In this section bisimulation semantics for $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ in a *specific* data environment is investigated.

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	$\phi_0\cdot\ldots\cdot\phi_n=\delta$
A7	$\delta x=\delta$		if $\forall s\in S\ \exists i\le n\,\mathbf{.}\,(s,\phi_i)\notin \mathit{test}$
A8	$\epsilon x=x$	WPC1	$wp(a,\phi)a\phi=wp(a,\phi)a$
A9	$x\epsilon=x$	WPC2	$\neg wp(a,\phi)a\neg\phi=\neg wp(a,\phi)a$

Table 3: The axioms of $\mathrm{BPA}_G(\mathcal{S})$ where $\phi,\phi_i\in G$ and $a\in A$

In Table 3 the axiom system $\mathrm{BPA}_G(\mathcal{S})$ is presented. It contains the axioms of BPA_G^4 and three new axioms depending on $\mathcal{S}$ (this explains the $\mathcal{S}$ in $\mathrm{BPA}_G(\mathcal{S})$). 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 G1 follows from SI.

In the axioms WPC1 and WPC2 (Weakest Precondition 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 $\mathcal{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ 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 $s \in S$:

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

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 $\mathcal{S}$ iff there is a weakest precondition in G of any $a \in A$ and $\phi \in G_{at}$. □

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

$$\forall s, s', s'' \in S\ (s', s'' \in \mathit{effect}(a, s) \quad \Longrightarrow \quad (\mathit{test}(\phi, s') \Longleftrightarrow \mathit{test}(\phi, s''))).$$

□

Remark that a data environment $\mathcal{S}$ with a deterministic function *effect* is sufficiently deterministic. If $\mathcal{S}$ is a data environment such that weakest preconditions are expressible and that is sufficiently deterministic then the axioms WPC1 and WPC2 exactly characterize the weakest preconditions 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. If on the other hand weakest preconditions are expressible in $\mathcal{S}$, then the soundness of BPA$_G(\mathcal{S})$ implies that $\mathcal{S}$ is also sufficiently deterministic. With the axioms for weakest preconditions G4 becomes derivable, so both axioms G1 and G4 need not be considered in the following results characterizing $\leftrightarrows_{\mathcal{S}}$.

Theorem 3.3. (Soundness) *Let $\mathcal{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. Let r_1, r_2 be closed terms over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$. If* $\mathrm{BPA}_G(\mathcal{S}) + \mathrm{REC} + \mathrm{RSP} \vdash r_1 = r_2$ *then* $r_1 \leftrightarrows_{\mathcal{S}} r_2$. □

Theorem 3.4. (Completeness) *Let $\mathcal{S}$ 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 $r_1 \leftrightarrows_{\mathcal{S}} r_2$ then* $\mathrm{BPA}_G(\mathcal{S}) \vdash r_1 = r_2$. □

Example 3.5. Process algebra with guards can be used to express and prove partial correctness formulas in Hoare logic. In [GP90] the soundness of a Hoare logic for process terms (see also [Pon89]) is proved. Here a simple example that is often used as an illustration of Hoare logic is presented and its correctness is shown.

Let BPA$_G(\mathcal{S})$ represent a small programming language with Boolean guards and assignments. The language has the signature of $\Sigma(\mathrm{BPA}_G)$ and further assume a set $\mathcal{V} = \{x, y, ...\}$ of data variables. Actions have the form

$$[x := e]$$

with $x \in \mathcal{V}$ a variable ranging over the integers $\mathbb{Z}$ and e an integer expression. Let some interpretation $[\![\cdot]\!]$ from closed integer expressions to integers be given. Atomic guards have the form

$$\langle e = f\rangle$$

where e and f are both integer expressions.

The components of the data environment $\mathcal{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ are defined by:

1. $S = \mathbb{Z}^{\mathcal{V}}$, i.e., the set of mappings from $\mathcal{V}$ to the integers, with typical element ρ;
2. $\mathit{effect}([x := e], \rho) = \rho[[\![\rho(e)]\!]/x]$, assuming that the domain of ρ is extended to integer expressions in the standard way, and $\rho[n/x]$ is as the mapping ρ, except that x is mapped to n;
3. $\mathit{test}(\langle e = f\rangle, \rho) \Longleftrightarrow ([\![\rho(e)]\!] = [\![\rho(f)]\!])$.

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

$$\mathit{wp}([x := e], \langle e_1 = e_2\rangle) = \langle e_1[e/x] = e_2[e/x]\rangle.$$

The axiom SI cannot be formulated so easily, partly because integer expressions are not yet defined very precisely. However, it can be characterized by the scheme:

$$\langle e_0 = f_0\rangle \cdot \ldots \cdot \langle e_n = f_n\rangle = \delta \quad \text{iff} \quad \forall \rho \in S\ \exists i \leq n \,\mathbf{.}\, [\![\rho(e_i)]\!] \neq [\![\rho(f_i)]\!].$$

Consider the following tiny program $SWAP$ that exchanges the initial values of x and y without using any other variables.

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

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:

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

$\square$

4 Parallel processes and guards

The language of $\Sigma(\text{BPA}_G)$ is extended to $\Sigma(\text{ACP}_G)$ by adding the following four operators [BK84, BW90]: the *encapsulation operator* ∂_H, the *merge* $\|$, the *left-merge* $\mathbin{\|\!\_}$ and the *communication-merge* $|$, suitable to describe the behavior of parallel, communicating processes. Encapsulation is used to enforce communication between processes. Communication is modeled 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 action resulting from the communication between a and b. All general definitions for $\Sigma(\mathrm{BPA}_G)$ carry over to $\Sigma(\mathrm{ACP}_G)$, especially, $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ denotes $\Sigma(\mathrm{ACP}_G)$ extended with all constants denoting solutions of guarded recursive specifications over $\Sigma(\mathrm{ACP}_G)$.

In Table 4 the axiom system ACP_G is presented (note that the axiom G4 is absent). Most of these axioms are standard for ACP 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 $\phi x \mathbin{\|\!\_} y$ $(\phi x \mid y)$ may prevent both x and y from happening.

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 $\Sigma(\mathrm{BPA}_G)$, by structural induction.

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 $\mathrm{ACP}_G \vdash p = q$.* □

ACP_G and BPA_G^4 or $\mathrm{BPA}_G(\mathcal{S})$ cannot be combined in bisimulation semantics as $\leftrightarrows_{\mathcal{S}}$ is not a congruence for the merge operators; if G4 is added to ACP_G one can derive

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

So, in (2) it can be the case that after an a step ϕ holds, and a state is entered where a b or a d step can be performed. Performing the d step may yield a state were $\neg\phi$ holds, so the only possible step left is a c step. This situation cannot be mimicked in (1): the only possible execution of adc in (1) has no b option after the a-step. 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.

As is it still the objective to prove $\mathcal{S}$-bisimilarity between closed terms containing merge operators, a *two-phase* calculus that does avoid these problems can be defined.

Definition 4.2. *(A two-phase calculus $\vdash_2$)* Let p_1, p_2 be closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. 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 $\mathrm{ACP}_G \vdash p_i = q_i \quad (i = 1, 2)$ and $\mathrm{BPA}_G^4 \vdash q_1 = q_2$.

Furthermore, write

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

iff there are closed terms q_1, q_2 over $\Sigma(\mathrm{BPA}_G)_{\mathrm{REC}}$ such that $\mathrm{ACP}_G \vdash p_i = q_i \quad (i = 1, 2)$ and $\mathrm{BPA}_G(\mathcal{S}) \vdash q_1 = q_2$.

Writing REC + RSP in front of $\vdash_2$ indicates that REC and RSP may be used in proving $p_i = q_i$ $(i = 1, 2)$ and $q_1 = q_2$. □

Let $\mathcal{S} = \langle S, \mathit{effect}, \mathit{test}\rangle$ be some data environment over a set A of atomic actions and a set G_{at} of atomic guards. Table 5 contains the transition rules defining an operational semantics for $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. Let

$$\longrightarrow_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}},\mathcal{S}} \subseteq C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S) \times A_{\surd} \times C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S)$$

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	$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$	EM10	$\phi x \mathbin{\lfloor\!\lfloor} y = \phi(x \mathbin{\lfloor\!\lfloor} y)$
EM2	$\epsilon \mathbin{\lfloor\!\lfloor} x = \delta$	EM11	$\phi x \mid y = \phi(x \mid y)$
EM3	$ax \mathbin{\lfloor\!\lfloor} y = a(x \parallel y)$		
EM4	$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$		
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	$\partial_H(xy) = \partial_H(x)\partial_H(y)$

Table 4: The axioms of ACP$_G$, $a, b \in A$, $H \subseteq A$ and $\phi \in G$

be the transition relation containing all transitions that are derivable by these rules. The following definition introduces a different bisimulation equivalence, called *global $\mathcal{S}$-bisimilarity*, that is a congruence for the merge operators. The idea behind a global $\mathcal{S}$-bisimulation is that a context $p \parallel (.)$ around a process q can change the data-state of q at any time and global $\mathcal{S}$-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', q_1 is related to q_2 in *any* data-state:

Definition 4.3. Let $\mathcal{S}$ be a data environment with data-state space S. A binary relation $R \subseteq C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S) \times C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S)$ is a *global $\mathcal{S}$-bisimulation* iff R satisfies the following (global) version of the transfer property: for all $(p, s), (q, s) \in C(\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, S)$ with $(p, s)R(q, s)$:

1. Whenever $(p, s) \xrightarrow{a}_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, \mathcal{S}} (p', s')$ for some a and (p', s'), then, for some q', also $(q, s) \xrightarrow{a}_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, \mathcal{S}} (q', s')$ and $\forall s'' \in S\ ((p', s'')R(q', s''))$,
2. Whenever $(q, s) \xrightarrow{a}_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, \mathcal{S}} (q', s')$ for some a and (q', s'), then, for some p', also $(p, s) \xrightarrow{a}_{\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}, \mathcal{S}} (p', s')$ and $\forall s'' \in S\ ((p', s'')R(q', s''))$.

Two closed terms p, q over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$ are *globally $\mathcal{S}$-bisimilar*, notation $p \mathrel{\underline{\underline{\leftrightarrow}}}_{\mathcal{S}} q$, iff for each $s \in S$ there is a global $\mathcal{S}$-bisimulation relation R with $(p, s)R(q, s)$. $\square$

By definition of global $\mathcal{S}$-bisimilarity it follows that

$$p \mathrel{\underline{\underline{\leftrightarrow}}}_{\mathcal{S}} q \quad \Longrightarrow \quad p \mathrel{\underline{\leftrightarrow}}_{\mathcal{S}} q$$

$\phi \in G$	$(\phi, s) \xrightarrow{\surd} (\delta, s) \quad \text{if } \mathit{test}(\phi, s)$	
$a \in A$	$(a, s) \xrightarrow{a} (\epsilon, s') \quad \text{if } s' \in \mathit{effect}(a, s)$	
$+$	$\dfrac{(x, s) \xrightarrow{a} (x', s')}{(x + y, s) \xrightarrow{a} (x', s')}$	$\dfrac{(y, s) \xrightarrow{a} (y', s')}{(x + y, s) \xrightarrow{a} (y', s')}$
$\cdot$	$\dfrac{(x, s) \xrightarrow{a} (x', s')}{(xy, s) \xrightarrow{a} (x'y, s')} \quad \text{if } a \neq \surd$	$\dfrac{(x, s) \xrightarrow{\surd} (x', s') \quad (y, s) \xrightarrow{a} (y', s'')}{(xy, s) \xrightarrow{a} (y', s'')}$
$\parallel$	$\dfrac{(x, s) \xrightarrow{a} (x', s')}{(x \parallel y, s) \xrightarrow{a} (x' \parallel y, s')} \quad \text{if } a \neq \surd$	$\dfrac{(y, s) \xrightarrow{a} (y', s')}{(x \parallel y, s) \xrightarrow{a} (x \parallel y', s')} \quad \text{if } a \neq \surd$
	$\dfrac{(x, s) \xrightarrow{a} (x', s') \quad (y, s) \xrightarrow{b} (y', s'')}{(x \parallel y, s) \xrightarrow{\gamma(a,b)} (x' \parallel y', s''')}$	if $\gamma(a, b) \neq \delta,\ a, b \neq \surd$, and $s''' \in \mathit{effect}(\gamma(a, b), s)$
	$\dfrac{(x, s) \xrightarrow{\surd} (x', s') \quad (y, s) \xrightarrow{\surd} (y', s')}{(x \parallel y, s) \xrightarrow{\surd} (x' \parallel y', s')}$	
$\mathbin{\lfloor\!\lfloor}$	$\dfrac{(x, s) \xrightarrow{a} (x', s')}{(x \mathbin{\lfloor\!\lfloor} y, s) \xrightarrow{a} (x' \parallel y, s')} \quad \text{if } a \neq \surd$	
$\mid$	$\dfrac{(x, s) \xrightarrow{a} (x', s') \quad (y, s) \xrightarrow{b} (y', s'')}{(x \mid y, s) \xrightarrow{\gamma(a,b)} (x' \parallel y', s''')}$	if $\gamma(a, b) \neq \delta,\ a, b \neq \surd$, and $s''' \in \mathit{effect}(\gamma(a, b), s)$
	$\dfrac{(x, s) \xrightarrow{\surd} (x', s') \quad (y, s) \xrightarrow{\surd} (y', s')}{(x \mid y, s) \xrightarrow{\surd} (x' \parallel y', s')}$	
∂_H	$\dfrac{(x, s) \xrightarrow{a} (x', s')}{(\partial_H(x), s) \xrightarrow{a} (\partial_H(x'), s')} \quad \text{if } a \notin H \subseteq A$	
recursion	$\dfrac{(\langle t_x \mid E \rangle, s) \xrightarrow{a} (y, s')}{(\langle x \mid E \rangle, s) \xrightarrow{a} (y, s')} \quad \text{if } x = t_x \in E$	

Table 5: Transition rules ($a, b \in A_{\surd},\ H, I \subseteq A$)

for closed terms p, q over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$. Moreover, global $\mathcal{S}$-bisimilarity *is* a congruence relation:

Lemma 4.4. *For any data environment $\mathcal{S}$ the relation $\underline{\leftrightarrow}_{\mathcal{S}}$ is a congruence with respect to the operators of $\Sigma(\mathrm{ACP}_G)$.* □

Theorem 4.5. (Soundness) *Let p, q be closed terms over $\Sigma(\mathrm{ACP}_G)_{\mathrm{REC}}$.*

1. *If $\mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash p = q$, then $p \,\underline{\leftrightarrow}_{\mathcal{S}}\, q$ for any data environment $\mathcal{S}$.*

2. *If $\mathrm{ACP}^4_G + \mathrm{REC} + \mathrm{RSP} \vdash_2 p = q$, then $p \leftrightarrow_{\mathcal{S}} q$ for any data environment $\mathcal{S}$.*

3. *Let $\mathcal{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. If $\mathrm{ACP}_G(\mathcal{S}) + \mathrm{REC} + \mathrm{RSP} \vdash_2 p = q$, then $p \leftrightarrow_{\mathcal{S}} q$.* □

Theorem 4.6. (Completeness) *Let r_1, r_2 be closed terms over $\Sigma(\mathrm{ACP}_G)$.*

1. *If $r_1 \,\underline{\leftrightarrow}_{\mathcal{S}}\, r_2$ for all data environments $\mathcal{S}$, then $\mathrm{ACP}_G \vdash r_1 = r_2$.*

2. *If $r_1 \leftrightarrow_{\mathcal{S}} r_2$ for all data environments $\mathcal{S}$, then $\mathrm{ACP}^4_G \vdash_2 r_1 = r_2$.*

3. *Let $\mathcal{S}$ be a data environment such that weakest preconditions are expressible and that is sufficiently deterministic. If $r_1 \leftrightarrow_{\mathcal{S}} r_2$, then $\mathrm{ACP}_G(\mathcal{S}) \vdash_2 r_1 = r_2$.* □

5 An example: a parallel predicate checker

In this section the techniques introduced up till now are illustrated by an example. Let $f \subseteq \mathbb{Z}$ be some predicate, e.g. the set of all primes. Now, given some number n, the objective is to calculate the smallest $m \geq n$ such that $f(m)$. Assume two devices P_1 and P_2 that can calculate for some given number k whether $f(k)$ holds. In Figure 1 a system is depicted that enables a calculation of m using both P_1 and P_2. A Generator/Collector G generates numbers $n, n+1, n+2, ...$, 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.

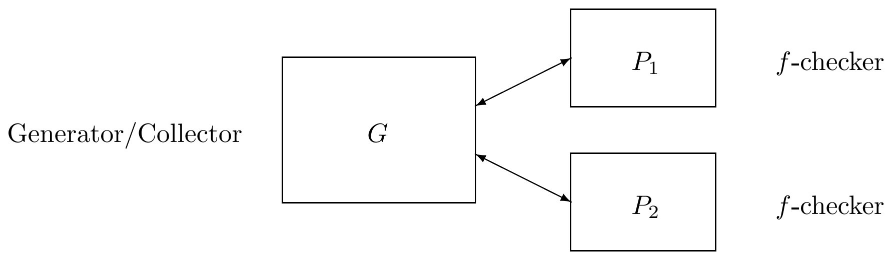


Figure 1: The parallel predicate checker Q

To describe this situation, Example 3.5 is extended with the atomic actions ($i = 1, 2$):

$s(!x)$	send value of x,
$s_{ok}(!x_i)$	send the value x_i for which the evaluation of $f(x_i)$ was a success,
s_{notok}	indicate that an evaluation of f was not successful,
$r(?x_i)$	read a value for x_i,
$r_{ok}(?y)$	read a value for y for which $f(y)$ succeeded,
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:

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

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.

Let new atomic guards $\langle f(t)\rangle$ for any integer expression t be added to the setting of Example 3.5. 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:

$$\begin{aligned}G &= [x := n]\, s(!x)\, [x := x + 1]\, s(!x)\, G_1\\ G_1 &= r_{notok}\, [x := x + 1]\, s(!x)\, G_1 + r_{ok}(?y)\, G_2\\ G_2 &= \neg\langle x = y\rangle\, w(y) + \langle x = y\rangle(r_{ok}(?y)\, w(y) + r_{notok}\, w(x))\\ \\ P_i &= r(?x_i)\, P_i' + \epsilon\\ P_i' &= \langle f(x_i)\rangle s_{ok}(!x_i) + \neg\langle f(x_i)\rangle\, s_{notok}\, P_i + \epsilon\\ \\ Q &= \partial_H(G \parallel (P_1 \parallel P_2))\end{aligned}$$

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 $f(m)$. Let new atomic guards $\langle\alpha(t, u)\rangle$ for integer expressions t, u be of help to express this formally:

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

Now Q is correct if $\mathrm{ACP}_G(\mathcal{S}) + \mathrm{REC} + \mathrm{RSP} \vdash_2 Q = Q'$, 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):

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

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

This proof is given by first expanding Q and Q' to the *merge*-free forms R and R':

$$\begin{array}{rcl}
R & = & [x := n]([x_1 := x]\, [x := x+1]\, [x_2 := x] \cdot R_1 + \\
 & & \qquad [x_2 := x]\, [x := x+1]\, [x_1 := x] \cdot R_1 \quad) \\
 & & \\
R_1 & = & \neg\langle f(x_1)\rangle c_{notok}\, [x := x+1]\, [x_1 := x] \cdot R_1 + \\
 & & \neg\langle f(x_2)\rangle c_{notok}\, [x := x+1]\, [x_2 := x] \cdot R_1 + \\
 & & \quad \langle f(x_1)\rangle\, [y := x_1]\, R_2 + \\
 & & \quad \langle f(x_2)\rangle\, [y := x_2]\, R_3 \\
 & & \\
R_2 & = & \neg\langle x = y\rangle\, w(y) + \\
 & & \quad \langle x = y\rangle(\langle f(x_2)\rangle\, [y := x_2]\, w(y) + \neg\langle f(x_2)\rangle\, c_{notok}\, w(x)) \\
 & & \\
R_3 & = & \neg\langle x = y\rangle\, w(y) + \\
 & & \quad \langle x = y\rangle(\langle f(x_1)\rangle\, [y := x_1]\, w(y) + \neg\langle f(x_1)\rangle\, c_{notok}\, w(x)).
\end{array}$$

The process R' is defined likewise, 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 that

$$\mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash Q = R \quad \text{and} \quad \mathrm{ACP}_G + \mathrm{REC} + \mathrm{RSP} \vdash Q' = R'. \tag{3}$$

In order to show that $\mathrm{BPA}_G(\mathcal{S}) + \mathrm{REC} + \mathrm{RSP} \vdash R = R'$ the following instances of SI, WPC1 and WPC2 are needed in addition to those given in Example 3.5. Let F be some function on integer expressions.

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

Note that these identities are valid. Let

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

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:

$$\begin{array}{rcl}
T &=& [x := n]([x_1 := x]\,[x := x+1]\,[x_2 := x]\cdot T_1 \;+ \\
 && \qquad\quad [x_2 := x]\,[x := x+1]\,[x_1 := x]\cdot T_1 \quad)\\[1ex]
T_1 &=& \beta\,(\neg\langle f(x_1)\rangle c_{notok}\,[x := x+1]\,[x_1 := x]\cdot T_1 \;+ \\
 && \quad \neg\langle f(x_2)\rangle c_{notok}\,[x := x+1]\,[x_2 := x]\cdot T_1 \;+ \\
 && \quad \langle f(x_1)\rangle\,[y := x_1]\,T_2 \;+ \\
 && \quad \langle f(x_2)\rangle\,[y := x_2]\,T_3\\[1ex]
T_2 &=& \langle y = x_1\rangle\langle f(x_1)\rangle\beta(\neg\langle x = y\rangle\, w(y) \;+ \\
 && \qquad\qquad\qquad \langle x = y\rangle(\langle f(x_2)\rangle\,[y := x_2]\,w(y) + \neg\langle f(x_2)\rangle\, c_{notok}\, w(x)))\\[1ex]
T_3 &=& \langle y = x_2\rangle\langle f(x_2)\rangle\beta(\neg\langle x = y\rangle\, w(y) \;+ \\
 && \qquad\qquad\qquad \langle x = y\rangle(\langle f(x_1)\rangle\,[y := x_1]\,w(y) + \neg\langle f(x_1)\rangle\, c_{notok}\, w(x)))
\end{array}$$

and thus $\mathrm{BPA}_G(\mathcal{S}) + \mathrm{REC} + \mathrm{RSP} \vdash R = R'$. Using (3) above it follows that

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

as was to be proved.

6 Discussion on the Kleene star

It is for the author still an open question whether the Kleene star operator * can be incorporated in process algebra such that its operational meaning is captured, and such that bisimulation semantics can be characterized by (conditional) equational laws.

Following [Fer92], a first alternative is to consider p^* as the process X defined by

$$X = pX + \epsilon$$

and therefore satisfying the axiom

$$x^* = x \cdot x^* + \epsilon$$

(which also implies what transition rules are appropriate). The obvious mismatch is that for e.g. an atomic action a the process a^* may give rise to an infinite execution, contrary to how it is often characterized. On the other hand, the conventional program **while** ϕ **do** a **od** would then translate in

$$(\phi \cdot a)^*\neg\phi \quad \text{(cf. [Har84, KT90]) i.e.,} \quad Y\neg\phi \quad \text{where} \quad Y = \phi a Y + \epsilon$$

which is a deterministic process: $Y\neg\phi$ performs from a data-state s an a-transition to $(Y\neg\phi, \mathit{effect}(s, a))$ iff $(s, \phi) \in \mathit{test}$, and a termination transition otherwise. Indeed, taking ϵ (i.e., **true**) for ϕ, this program represents an infinite a-loop, according to what should be expected.

A second alternative is to interpret for instance a^* as the process defined by

$$\epsilon + \sum_{0<i<\omega} a^i.$$

One of the problems in this case is that it would require rather strong proof principles to distinguish this process from

$$\epsilon + \sum_{0<i\leq\omega} a^i.$$

Adopting this third alternative again introduces an infinite a-branch. It should be remarked that the second alternative above matches most closely with the description of the * operator as given in [KT90]:

p^* = "Execute p repeatedly a nondeterministically chosen finite number of times."

Assume $\Sigma(\mathrm{BPA}_G)$ is extended with the Kleene star * with its operational meaning according to the first alternative, i.e.,

$$p^* = p \cdot p^* + \epsilon.$$

Consider the following extension of BPA^4_G with the five axioms Kl1 – Kl5 on the Kleene star (call the resulting system BPA^*_G):

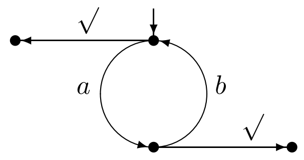
A1	$x+y=y+x$	G1	$\phi\cdot\neg\phi=\delta$
A2	$x+(y+z)=(x+y)+z$	G2	$\phi+\neg\phi=\epsilon$
A3	$x+x=x$	G3	$\phi(x+y)=\phi x+\phi y$
A4	$(x+y)z=xz+yz$	G4	$a(\phi x+\neg\phi y)\subseteq ax+ay$
A5	$(xy)z=x(yz)$	Kl1	$x^*=x\cdot x^*+\epsilon$
A6	$x+\delta=x$	Kl2	$x^*=(x(x+\epsilon))^*$
A7	$\delta x=\delta$	Kl3	$(x+y^*)^*=(x+y)^*$
A8	$\epsilon x=x$	Kl4	$(x\cdot y^*)^*=\epsilon+x(x+y)^*$
A9	$x\epsilon=x$	Kl5	$(x^*(y+\epsilon))^*=(x(y+\epsilon)+y)^*$

Table 6: The axioms of BPA^*_G where $\phi\in G$ and $a\in A$

The question whether these particular axioms completely characterize bisimilarity over $\Sigma(\mathrm{BPA}^*_G)$ is a topic of further research. However, the following typical identities are derivable in BPA^*_G:

1.	$\delta^*=\epsilon$	[Apply Kl1, A7 and A6],
2.	$(x^*)^*=x^*$	[Apply Kl3 on $(\delta+x^*)^*$],
3.	$x^*\cdot x^*=x^*$	[Apply Kl1 on $(x^*)^*$],
4.	$\epsilon^*=\epsilon$	[Use 1 and 2],
5.	$(x+\epsilon)^*=x^*$	[Use 1 and Kl3],
6.	$(x+\epsilon)(x+y)^*=(x+y)^*$	[Use Kl1 on $(x+y)^*$, extract $x(x+y)^*$, Kl1],
7.	$x^*(x+y)^*=(x+y)^*$	[Use Kl3, Kl1 on $(x+y)^*$],
8.	$(x^*y^*)^*=(x+y)^*$	[Use Kl4, Kl3, 7, omit '$\epsilon+$'],
9.	$(x+y^*z^*)^*=(x+y+z)^*$	[Use Kl3, 8, Kl3],
10.	$(x^*\cdot y^*\cdot z^*)^*=(x+y+z)^*$	[Use Kl4, 9, 7, omit '$\epsilon+$'],
11.	$((x+\epsilon)(x+\epsilon))^*=x^*$	[Use Kl1, 6 and omit '$+\epsilon$'].

As for the expressivity of $\Sigma(\mathrm{BPA}^*_G)$, it does not seem to be the case that all finitely branching *finite state* processes are specifiable (modulo bisimulation semantics). Consider for example the following transition system over a trivial data environment with singleton state space (its root is marked with a little arrow):



which seems not to be specifiable (the right-hand side termination step causes the problem: termination *within* an iteration can only occur (?) if at that state a new iteration exists). Observe that replacing b by a makes the transition system above specifiable, e.g. by a^*.

References

[BK84] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proceedings* 11*th* *ICALP,* Antwerp, volume 172 of *Lecture Notes in Computer Science*, pages 82–95. Springer-Verlag, 1984.

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

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

[Fer92] R.T.P. Fernando. Parallelism, partial evaluation and programs as relations on states. In preparation.

[GP90] J.F. Groote and A. Ponse. Process algebra with guards. Report CS-R9069, CWI, Amsterdam, 1990. To appear in *Formal Aspects of Computing.*

[Har84] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. Reidel, 1984.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1), 137–161, 1985.

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

[KT90] D. Kozen and J. Tiuryn. Logics of programs. *Handbook of Theoretical Computer Science*, pages 789–840. Elsevier Science Publishers, 1990.

[MA86] E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1986.

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

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, pages 319–340, 1976.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5*th* *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

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

[Pon89] A. Ponse. Process expressions and Hoare's logic. *Information and Computation*, 95(2):192–217, 1991.

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