Process algebra with propositional signals

Baeten, J.C.M.; Bergstra, J.A.

Published in: Theoretical Computer Science

DOI: 10.1016/S0304-3975(96)00253-8

Published: 01/01/1997

Citation for published version (APA):
Abstract

We consider processes that have transitions labeled with atomic actions, and states labeled with formulas over a propositional logic. These state labels are called signals. A process in a parallel composition may proceed conditionally, dependent on the presence of a signal in the process in parallel. This allows a natural treatment of signal observation.

1. Introduction

This paper can be viewed as a revision and simplification of [3] in which we have introduced the so-called signals as labels for states in processes. Previous work in the same direction includes [5, 9]. Though useful in a multitude of examples, it has turned out that the mechanism of signal observation of [3] is quite complex. The approach taken was that actions observe signals. What we propose here is to require that the signals are propositions (i.e. elements of a boolean algebra; this is consistent with [3]) and then to use tests to read off information from these signals. In this way, conditions in conditional expressions (written as $\phi \rightarrow x$, or $x \leftarrow \phi \rightarrow y$) and propositional signals are complementary. A mechanism to localise or hide the propositional signals is important. In summary, our development is based on the position:

The visible part (signal) of the state of a process is a proposition.

Whatever the merits of this position, what we do establish is that it is a consistent one, and that it allows a wide range of examples.

The introduction of propositional signals in the context of process algebra occurs to us as a necessary step, it completes the picture that emerges if conditional process
expressions are introduced. Indeed, consider an expression \( x \triangleleft \phi \triangleright y \). If \( \phi \) is true or false, this is just \( x \) or \( y \). But in the more general case that \( \phi \) ranges over a class of propositions, what determines the meaning of \( x \triangleleft \phi \triangleright y \)? An answer is: \( \phi \) is to be evaluated over a state. This leads one into state operators (as in [2]) or global states (see [12]), thus departing from the core of process algebra where every dynamic entity is a process.

So we feel that the primary motivation of this paper is a conceptual one and that additional motivation in terms of potential applications is both premature and superfluous. This is not meant to imply that we are pessimistic about applications. It rather is the case that we would propose to view process algebra with propositional signals as a subject in pure logic at least initially. Many extensions or modifications can be imagined: first-order signals, higher-order signals, infinitary and non-classical logics for the entailment relation between signals and conditions, modal and temporal logics for processes with propositional signals.

We are not aware of any previous work aiming at objectives similar to our present ones. The present approach is also followed in [7]. Clearly, our approach, based on ACP [6] can be adapted to CCS [13], MEIJE [1] or ATP [15] without much effort. Adaptation to CSP [8] is more involved due to the different models, based on failure or ready sets.

2. Basic process algebra with propositional signals

2.1. BPA with inaction and nonexistence

Let \( A \) be a finite set. The elements of \( A \) will be called atomic actions. Every atomic action is an element of \( P \), the sort of processes. There are also two binary operators on \( P \), viz., \( + \) (alternative composition) and \( \cdot \) (sequential composition). The core system BPA (basic process algebra) over this signature has the axioms A1–A5 of Table 1 below \((x, y, z \in P)\) and is well-known from [6,4]. The constant \( \delta \) denoting inaction (or deadlock) is added to the language with axioms A6 and A7. We write \( A_\delta = A \cup \{ \delta \} \). In this paper, we introduce a new constant for process algebra, viz., \( \perp \). This constant stands for nonexistence, we will need it when we introduce signals further on. It is axiomatized by axioms NE1–NE3 of Table 1 \((x \in P, a \in A)\). Nonseness stands for an

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPA_Δ</td>
</tr>
<tr>
<td>x + y = y + x \quad A1</td>
</tr>
<tr>
<td>(x + y) + z = x + (y + z) \quad A2</td>
</tr>
<tr>
<td>x + x = x \quad A3</td>
</tr>
<tr>
<td>(x \cdot y) : z = x \cdot (y \cdot z) \quad A4</td>
</tr>
<tr>
<td>(x \cdot y) : z = x \cdot (y \cdot z) \quad A5</td>
</tr>
<tr>
<td>x + \delta = x \quad A6</td>
</tr>
<tr>
<td>\delta \cdot x = \delta \quad A7</td>
</tr>
<tr>
<td>x + \perp = \perp \quad NE1</td>
</tr>
<tr>
<td>\perp \cdot x = \perp \quad NE2</td>
</tr>
<tr>
<td>a \cdot \perp = \delta \quad NE3</td>
</tr>
</tbody>
</table>
inconsistent state of a process: such a state can never be exited (NE1 and NE2) and also, it is impossible to enter such a state from a consistent state (NE3). For example, in term \( a \cdot b \cdot \bot \), \( a \) can be executed, but then execution stops, since execution of \( b \) leads to an inconsistency. We see this term should equal \( a \cdot \delta \). This signature and these axioms together constitute the theory \( \text{BPA}_\bot \), basic process algebra with inaction and nonexistence.

2.2. Conditionals

Besides the sort of processes \( P \), we will have a second sort \( B \). Elements of this sort are propositional logic formulas over a set of basic propositional variables \( P_1, \ldots, P_n \) with constants \( T, F \) (true, false) and operators \( \neg, \vee, \wedge, \Rightarrow, \equiv \) (negation, disjunction, conjunction, implication, bi-implication). We use the "horse-shoe" sign for implication (as is common in philosophical logic) in order not to have too many different types of arrows.

We use letters \( \phi, \psi \) to range over \( B \). For \( \phi \) a formula of propositional logic, \([\phi]\) denotes the equivalence class modulo derivability, i.e. the set of formulas derivably equal (in classical logic) to \( \phi \). In derivations we can always use identities of propositional logic, so actually, we are using equivalence classes of formulas as our second sort, not the formulas themselves. We will sometimes be sloppy in this distinction. We can also identify each equivalence class with a set of valuations, i.e. the set of those valuations that make \( \phi \) true. Here, a valuation is just a mapping from \( \{P_1, \ldots, P_n\} \) to \( \{T, F\} \). Such a mapping can easily be extended to all formulas.

As in \([3]\), we use the guarded command. \( \phi \rightarrow x \) is read as if \( \phi \) then \( x \) (if \( \phi \) holds in the initial state of \( x \), then an initial action of \( x \) can be executed). We have the basic axioms of Table 2 below, using the numbering of \([3]\). Note that axiom GC9 is derivable from the other axioms: \( \phi \rightarrow \delta = \phi \rightarrow (F \rightarrow x) = (\phi \land F) \rightarrow x = F \rightarrow x = \delta \).

Note that propositions need not be persistent, so e.g. in the term \( (P \rightarrow a) \cdot (\neg P \rightarrow b) \) it is possible that \( a \) followed by \( b \) is executed.

2.3. Root and terminal signal emission operators

The next operators to be introduced are the signal emission operators. There is the root signal emission operator \( \wedge \) and the terminal signal emission operator \( \wedge \) (we

<table>
<thead>
<tr>
<th>Conditionals over propositional logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T \rightarrow x = x )</td>
</tr>
<tr>
<td>( F \rightarrow x = \delta )</td>
</tr>
<tr>
<td>( \phi \rightarrow \delta = \delta )</td>
</tr>
<tr>
<td>( \phi \rightarrow (x + y) = (\phi \rightarrow x) + (\phi \rightarrow y) )</td>
</tr>
<tr>
<td>( (\phi \lor \psi) \rightarrow x = (\phi \rightarrow x) + (\psi \rightarrow x) )</td>
</tr>
<tr>
<td>( \phi \rightarrow (\psi \rightarrow x) = (\phi \land \psi) \rightarrow x )</td>
</tr>
<tr>
<td>( \phi \rightarrow (x \cdot y) = (\phi \rightarrow x) \cdot y )</td>
</tr>
</tbody>
</table>
take this notation from [7]). The intuition behind these operators is that both assign labels (signals) to the states of processes. Root signal emission places a signal at the root node of a process. Terminal signal emission places one and the same signal at each terminal node of a process. The terminal signal emission operator is not needed if the language contains a constant for the empty process (the process that terminates immediately and successfully) or if one is interested solely in processes that emit signals exclusively in nonterminal states (see the remark about axiom TRSE1). Leaving out all axioms involving terminal emission from the coming sections one will obtain an appropriate description of root signal emission. In Table 3, we present equations for root signal emission. Note that axiom RSE6 is derivable, and that the addition of these axioms to BPA makes axioms A3, A6, A7 and NE1 and NE2 derivable.

The first axiom expresses the fact that the root of a sequential product is the root of its first component. Axiom RSE2 can be given in a more symmetric form as follows:

$$(\phi \land x) + (\psi \land y) = (\phi \land \psi) \land (x + y).$$

This equation depends on the fact that the roots of two processes in an alternative composition are identified. Therefore, signals must be combined. The third axiom expresses the fact that there is no sequential order in the presentation of signals. Of course, one might imagine that a sequential ordering on signals is introduced, but we think that the introduction of such a sequential ordering is far from obvious (it also leads to problems concerning the associativity of the parallel composition operator). The combination of the signals is taking 'both' of them whereas $x + y$ has to choose between $x$ and $y$. As an example, consider the following derivation:

$$a \cdot ((\phi \land b) + (\neg \phi \land b)) = a \cdot ((\phi \land \neg \phi) \land b) = a \cdot (F \land b) = a \cdot \bot = \delta.$$  

A state emitting a signal denotes that this signal holds in the state. Falsity never holds, so a state emitting (a formula equivalent to $F$) cannot occur, is inconsistent.

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Root signal emission</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\phi \land x) \cdot y = \phi \land (x \cdot y)$</td>
<td>RSE1</td>
</tr>
<tr>
<td>$(\phi \land x) + y = \phi \land (x + y)$</td>
<td>RSE2</td>
</tr>
<tr>
<td>$\phi \land (\psi \land x) = (\phi \land \psi) \land x$</td>
<td>RSE3</td>
</tr>
<tr>
<td>$T \land x = x$</td>
<td>RSE4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Remaining axioms of BPAps</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x \cdot y) \land \phi = x \cdot (y \land \phi)$</td>
<td>TSE1</td>
</tr>
<tr>
<td>$(x + y) \land \phi = x \land \phi + y \land \phi$</td>
<td>TSE2</td>
</tr>
<tr>
<td>$(\land \phi) \land \psi = x \land (\phi \land \psi)$</td>
<td>TSE3</td>
</tr>
<tr>
<td>$x \land T = x$</td>
<td>TSE4</td>
</tr>
<tr>
<td>$(x \land \phi) \cdot y = x \cdot (\phi \land y)$</td>
<td>TRSE1</td>
</tr>
</tbody>
</table>
This explains RSE5, and the necessity of the nonexistence constant. RSE7 expresses how to take a signal outside of a conditional: the signal $\psi$ will only be emitted if condition $\phi$ is true. The last axiom RSE8 is the signal inspection rule. If a signal $\phi$ is emitted, then $\phi$ holds in the current state. Note the following generalisation of RSE8:

$$\phi \land (\psi \imp x) = \phi \land (\phi \imp ((\phi \land \psi) \imp x)).$$

Another interesting identity that follows is the following: $\phi \land x = (\phi \land \delta) + x$. This equation is indeed very useful for writing efficient process specifications mainly because it allows to a large extent to work with process algebra expressions that are not cluttered with signal emissions.

The equations in Table 4 regard terminal signal emission, and are for the most part explained by the previous remarks. Now, also TSE6 and TSE7 and NE3 are derivable. The axiom TRSE1 denotes that the end of $x$ coincides with the beginning of $y$. If we would have a constant $\varepsilon$ that is a unit for sequential composition (i.e. $x \cdot \varepsilon = x = \varepsilon \cdot x$) this allows to remove any terminal signal emission, since $x \land \phi = (x \land \phi) \cdot \varepsilon = x \cdot (\phi \land \varepsilon)$. TSE6 and TSE7 express that terminal signals do not occur for processes that have no termination states. TSE8 holds since the start state of a process cannot be a termination state at the same time, so condition and terminal signal do not interfere.

Similar considerations explain TRSE2.

The axiom system BPAs, basic process algebra with propositional signals, consists of all axioms from Tables 1–4. The combination of conditionals and root signal emission allows to derive the following lemma, that will be useful in reduction of closed terms to normal form.

**Lemma 2.1.** $\text{BPAs} \vdash \phi \imp \bot = \neg \phi \land \delta$.

**Proof.** $\phi \imp \bot = \phi \imp (F \land x) = (\phi \lor F) \land (F \imp x) = \neg \phi \land \delta$.

### 2.4. Basic terms

We define a set of basic terms $\mathcal{B}$ inductively. We allow only signals and conditions different from false, so that all actions occurring can actually be executed. This will be needed in Lemma 2.6 further on.

(i) $\bot \in \mathcal{B}$,

(ii) if $\phi \notin [F]$ (i.e. $\phi$ not equivalent to $F$), then $\phi \land \delta \in \mathcal{B}$,

(iii) if $\phi, \psi \notin [F], a \in A$, then $(\phi \imp a \land \psi) \in \mathcal{B}$,

(iv) if $\phi \notin [F], a \in A, t \in \mathcal{B}$, then $(\phi \imp a \cdot t) \in \mathcal{B}$,

(v) if $t, s \in \mathcal{B}$, then $t + s \in \mathcal{B}$.

Each basic term can be written as $\bot$ or in the form

$$\zeta \land \delta + \sum_{i=1}^{n} \phi_{i} \imp a_{i} \cdot x_{i} + \sum_{j=1}^{m} \psi_{j} \imp b_{j} \land \chi_{j}$$

\((\zeta, \phi_{i}, \psi_{j}, \zeta_{j} \notin [F], a_{i} \in A, b_{j} \in A_{\delta}, \chi_{j} \in \mathcal{B}, m, n \geq 0), \)
or, equivalently,

\[ \zeta \left( \sum_{i=1}^{n} \phi_i \cdot a_i \cdot x_i + \sum_{j=1}^{m} \psi_j \cdot b_j \cdot \chi_j \right) \]

When a basic term has this form, we call \( \zeta \) its root signal, and the subterms \( \phi_i \cdot a_i \cdot x_i, \psi_j \cdot b_j \cdot \chi_j \) its summands.

**Basic Term Lemma 2.2.** For all closed terms \( s \) there is a basic term \( t \) such that \( BPAps \vdash t = s \).

**Proof (sketch).** Basically, this follows from the fact that the term rewriting system consisting of axioms A4 and A5 from Table 1 together with all axioms from Table 2–4 and the equation in Lemma 2.1 is strongly normalising. This can be proved by using the method of the lexicographical path ordering (making the signature one-sorted, adding rewrite rules for propositional logic, taking the ordering \( \wedge > \because > \cdot > + > \delta, + > \wedge > \top, \wedge, \because \), giving \( \cdot \) the lexicographical status for the first argument, and \( \because, \wedge, \because \) for the second argument). Each normal form of this term rewriting system can easily be converted into a basic term. \( \square \)

### 2.5. Structured operational semantics

We proceed to give the semantics of BPAps using structured operational rules (SOS). The semantics uses the following predicates and relations on closed terms:

- \( x \stackrel{\phi}{\xrightarrow{a}} x' \): term \( x \) can do an \( a \)-step under condition \( \phi \) to term \( x' \) (\( a \in A \)),
- \( x \xrightarrow{a} \psi \): term \( x \) can do a terminating \( a \)-step under condition \( \phi \) leaving terminal signal \( \psi \),
- \( sp(x) = \phi \): the root signal of \( x \) is \( \phi \).

Plotkin-style rules for the step relations and step predicates are given in Table 5; the rules for the root signal predicate are given in the form of axioms in Table 6. The Plotkin-style rules for the root signal operator actually take the form of axioms, but they can be rephrased in the form of rules without many problems. For instance, RSO2 and RSO6 can be rephrased as follows:

\[
\begin{align*}
sp(x) &= \phi, \\
sp(y) &= \psi, \\
sp(x + y) &= \phi \wedge \psi, \\
sp(\phi \Rightarrow x) &= \phi \Rightarrow \psi
\end{align*}
\]

Many of the rules have conditions that ensure consistency of states: we may not have arrows leading to or coming from inconsistent states. Note that by the last two rules, it is possible that the condition on the arrow becomes equal to \( F \).

### 2.6. Bisimulation

With this operational semantics involving conditions on the arrows comes a new definition for bisimulation. Instead of just requiring matching actions, we also require
matching conditions; however, one transition on one side may have to be matched with a number of transitions on the other side (possibly zero), depending on the truth value of the propositional constants. Therefore, the following definition starts from the set of valuations of the propositional constants, i.e. all mappings \( v: \{ P_1, \ldots, P_n \} \to \{ T, F \} \). Each such mapping naturally extends to a mapping on all formulas. We write \( \phi = \psi \) (also in the rules above) iff \( [\phi] \equiv [\psi] \) or, equivalently, for all valuations \( v \), \( v(\psi) = T \) iff \( v(\phi) = T \). Consequently, \( \phi \neq \psi \) iff \([\phi] \neq [\psi]\) iff there is a valuation \( v \) with \( v(\phi) = T \) and \( v(\psi) = F \), or \( v(\phi) = F \) and \( v(\psi) = T \).

Then we say that a relation \( R \) on closed terms is a (strong) bisimulation when the following holds:

1. if \( xRy \) then \( s_{\rho}(x) = s_{\rho}(y) \);
2. if \( xRy \) and \( x \xrightarrow{\phi,a} x' \), then for all valuations \( v \) such that \( v(s_{\rho}(x) \land \phi) = T \), there is a condition \( \psi \) and an expression \( y' \) such that \( v(\psi) = T \), \( y \xrightarrow{\psi,a} y' \) and \( x'Ry' \);
3. if \( xRy \) and \( y \xrightarrow{\phi,a} y' \), then for all valuations \( v \) such that \( v(s_{\rho}(y) \land \phi) = T \), there is a condition \( \psi \) and an expression \( x' \) such that \( v(\psi) = T \), \( x \xrightarrow{\psi,a} x' \) and \( x'Ry' \);
4. if \( xRy \) and \( x \xrightarrow{\phi,a} \psi \), then for all valuations \( v \) such that \( v(s_{\rho}(x) \land \phi) = T \), there are conditions \( \phi' \), \( \psi' \) with \( v(\phi') = T \), \( \psi' = \psi' \) and \( x \xrightarrow{\phi',a} \psi' \);
5. if \( xRy \) and \( y \xrightarrow{\phi,a} \psi \), then for all valuations \( v \) such that \( v(s_{\rho}(y) \land \phi) = T \), there are conditions \( \phi' \), \( \psi' \) with \( v(\phi') = T \), \( \psi' = \psi' \) and \( x \xrightarrow{\phi,a} \psi' \).

We call two expressions \( x, y \) (strongly) bisimilar, notated \( x \leftrightarrow y \), if there is a (strong) bisimulation relating \( x \) and \( y \).

A simple example where an arrow on one side matches no arrow on the other side, is \((F : a) \leftrightarrow \delta\), an example where one arrow on one side must be matched with more than one arrow on the other side is \( a \leftrightarrow (P : a) + (\neg P : a) \).

In order to prove that this relation of bisimulation is a congruence relation, we want to use the congruence theorem of [17]. Before we can do this, we need to reformulate the operational rules. This reformulation may also give the reader more insight into this definition.

**Proposition 2.3 (Panth format).** The SOS specification in Tables 5 and 6 is equivalent to one in the panth format of [17]. This reformulation makes the definition of bisimulation in 2.6 into the standard definition of strong bisimulation for the panth format.

**Proof.** We reformulate the rules above in such a way that every arrow on one side matches exactly one arrow on the other side. In order to do so, we label the arrows with pairs of valuations and atomic actions, and replace end conditions with sets of valuations (i.e. equivalence classes of formulas). To be precise, we have the following relations:

- for each valuation \( v: \{ P_1, \ldots, P_n \} \to \{ T, F \} \) and each \( a \in A \), a binary relation \( \xrightarrow{v,a} \);
- for each valuation \( v, a \in A \) and each nonempty set of valuations \( V \) a unary relation \( \xrightarrow{v,a} V \);
- a unary relation \( v \in [s_{\rho}(-)] \).
Table 5
SOS rules \((a \in A)\)

\[
\begin{array}{lll}
a \xrightarrow{T,a} T & \frac{x \xrightarrow{\delta,a} x'}{x \cdot y \xrightarrow{\delta,a} x' \cdot y} & \frac{x \xrightarrow{\delta,a} y, y \neq F}{x \cdot y \xrightarrow{\delta,a} y \\ \psi} \\
\hline
x \xrightarrow{\delta,a} x', s_p(x + y) \neq F & x + y \xrightarrow{\delta,a} x', y + x \xrightarrow{\delta,a} x' \\
x \xrightarrow{\delta,a} x', \psi \land s_p(x) \neq F & x \xrightarrow{\delta,a} x, \psi \land s_p(x) \neq F \\
\psi \land x \xrightarrow{\delta,a} x' & \psi \land x \xrightarrow{\delta,a} x' \\
x \xrightarrow{\delta,a} x' & x \xrightarrow{\delta,a} x' \\
(\psi \rightarrow x) \xrightarrow{\delta,a} x' & (\psi \rightarrow x) \xrightarrow{\delta,a} x'
\end{array}
\]

Table 6
Root signal operator \((a \in A_0)\)

\[
\begin{array}{lll}
s_p(\bot) = F & RS00 & s_p(\phi \land x) = \phi \land s_p(x) \\
s_p(a) = T & RS01 & s_p(\phi \land \psi) = s_p(x) \\
s_p(x + y) = s_p(x) \land s_p(y) & RS02 & s_p(\phi \rightarrow x) = \phi \Rightarrow s_p(x) \\
s_p(x + y) = s_p(x) & RS03 & \\
\hline
\end{array}
\]

We will only allow an outgoing arrow \(\xrightarrow{v,a}\) from a term \(x\) where \(v\) makes the root signal true, i.e. \(v \in [s_p(x)]\). We give a couple of examples of reformulations, and leave the rest to the reader.

Thus, the axiom \(a \xrightarrow{T,a} T\) will be replaced by the set of axioms \(a \xrightarrow{v,a} T\), one for each valuation \(v\), where \(T\) is the set of all valuations. The third rule of Table 5 will be replaced by

\[
x \xrightarrow{v,a} [\psi], \quad w \in [\psi], \quad w \in [s_p(y)] \\
\quad x \cdot y \xrightarrow{v,a} \psi \land y
\]

and axiom RS06 of Table 6 will be replaced by the following two rules:

\[
\begin{array}{ll}
v \in [s_p(x)] & \quad v \notin [\phi] \\
v \in [s_p(\phi \rightarrow x)] & \quad v \in [s_p(\phi \rightarrow x)]
\end{array}
\]

The last rule is the only instance where we have a negative premise.

It is not difficult to see that the notion of strong bisimulation that goes with these rules is exactly the notion of bisimulation of Section 2.6. Also, we can see that this set of rules is in the panth format of [17] and is stratifiable according to the definition of [17]. A consequence of the theorem of [17] is now the following proposition.
Proposition 2.4. Bisimulation is a congruence relation on process expressions.

As a consequence, we can consider the algebraic structure \( P/e \) of process expressions modulo bisimulation equivalence.

Theorem 2.5 (Soundness). \( P/e \models BPAps. \)

Proof. By the previous proposition, it is enough to verify each axiom separately. We confine ourselves to give the bisimulation relation. Note that \( P/e \not\models \perp = \delta \), since \( s_\delta(\perp) = F \neq T = s_\delta(\delta). \)

For axiom A1, take the relation relating left-hand and right-hand side and relating each term to itself. A2–A4 go similarly. For A5, relate in addition all pairs of the form \( x \cdot (y \cdot z) \) to \( (x \cdot y) \cdot z \), and all pairs of the form \( (\phi \wedge x) \cdot y \) to \( \phi \wedge (x \cdot y) \). A6 goes like A1, and for A7 it suffices to relate right-hand and left-hand side. NE1–NE3 go like A7.

GC1, GC10, GC13 and GC12 go like A1, GC2 and GC9 like A7. GC11 also goes like A1, but note that here we use the fact that for a valuation \( v, v((4 \wedge x) \lor (\psi \wedge x)) = T \) iff \( v(4 \wedge x) = T = v(\psi \wedge x) = T. \)

RSE1–RSE4 and RSE7 go like A1, RSE5 and RSE6 like A7. RSE8 also goes like A1, but here we also use the fact that we only need to consider valuations that make the root signal true. For TSE1, relate all terms to themselves, and all terms of the form \( (x \cdot y) \wedge \phi \) to \( x \cdot (y \wedge \phi) \). TSE2, TSE8, TRSE2 go like A1. For TSE3, relate all terms to themselves, and all terms of the form \( (x \cdot y) \wedge \phi \) to \( x \cdot (y \wedge \phi) \). TSE2, TSE8, TRSE2 go like A1. For TSE3, relate all terms to themselves, and all terms of the form \( (x \wedge \phi) \wedge \psi \) to \( x \wedge (\phi \wedge \psi) \). For TSE4, relate all terms to themselves, and all terms of the form \( x \wedge T \) to \( x. \) TSE5–TSE7 go like A7. For TRSE1, relate all terms to themselves, all terms of the form \( (x \wedge \phi) \cdot y \) to \( x \cdot (\phi \wedge y) \), and all terms of the form \( \phi \wedge (\psi \wedge x) \) to \( (\phi \wedge \psi) \cdot x. \)

For basic terms, there is a direct relation between syntax and semantics.

Lemma 2.6. Let \( t \in \mathcal{B}. \) The root signal and the summands of a basic term were defined in 2.4.

(i) The root signal of \( t \) is \( s_\rho(t). \)
(ii) \( t \rightarrow^{\phi \cdot a} s \) iff \( \phi \cdot a \cdot s \) is a summand of \( t. \)
(iii) \( t \rightarrow^{\psi \cdot a} \psi \cdot a \wedge \psi \) is a summand of \( t. \)

Theorem 2.7 (Completeness). Let \( t, s \) be two closed BPAps terms. Then \( t \models s \) implies \( BPAps \vdash t = s. \)

Proof. By the basic term lemma and soundness, it is enough to prove this for basic terms. The proof can be completed using Lemma 2.6. \( \Box \)

As a corollary, we have \( P/e \models t = s \iff BPAps \vdash t = s \) for all closed \( t, s. \)
2.7. Global signal emission

In the next section, we will extend BPAps with parallel composition. There, we will need as an extra operator the global signal emission operator, that adds a signal to each state of a process. We give axioms for this operator in Table 7, and semantical rules in Table 8. With the help of the global signal emission operator, we can define a notion of invariance: \( \phi \) is an invariant of \( x \) if \( \phi \land x = \phi \upharpoonright x \).

2.8. Root signal operator and root signal deletion operator

We used the root signal operator \( s_p \) in the operational semantics. We can also add this operator to the theory with the axioms of Table 6. The operator \( s_p \) determines the root signal of a process. If \( s_p(x) = T \) we say that \( x \) has a trivial root signal; otherwise \( x \) has a non-trivial root signal. Processes that were studied until now in the context of process algebra always have a trivial root signal.

We can also define an operator \( p_p \) that removes the root signal from its argument (see Table 9). We found no use for this operator in the present paper. Axiom RSD2 is explained as follows: the root signal deletion operator only distributes over alternative composition if the root signal of the combination is not false, if the root signals of the alternatives do not contradict each other, for otherwise the root of the sum denotes an inconsistent state. Therefore, a case distinction is necessary. Notice that the equation \( s_p(x \uparrow \phi) = s_p(x) \) is derivable:

\[
 s_p(x \uparrow \phi) = s_p((x \uparrow \phi) \cdot y) = s_p(x \cdot (\phi \uparrow y)) = s_p(x).
\]

Also \( x = s_p(x) \uparrow p_p(x) \) will now be derivable for finite closed process expressions. As a rewrite rule it is useless, however, because it will immediately introduce an infinite loop.

Table 7
Global signal emission (\( a \in A_2 \))

<table>
<thead>
<tr>
<th>( a \downarrow x )</th>
<th>( \phi \downarrow )</th>
<th>( \psi \land s_p(x) \neq F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi \downarrow a )</td>
<td>( \phi \downarrow a \uparrow \phi )</td>
<td>( \phi \downarrow (\phi \downarrow x) = \phi \downarrow (\phi \downarrow x) )</td>
</tr>
<tr>
<td>( \phi \downarrow (x + y) = (\phi \downarrow x) + (\phi \downarrow y) )</td>
<td>( \phi \downarrow (\psi \to x) = \psi \to (\phi \downarrow x) + \neg \psi \to (\phi \downarrow \delta) )</td>
<td></td>
</tr>
<tr>
<td>( \phi \downarrow (x \cdot y) = (\phi \downarrow x) \cdot (\phi \downarrow y) )</td>
<td>( \phi \downarrow \psi )</td>
<td></td>
</tr>
</tbody>
</table>

Table 8
Operational semantics of global signal emission (\( a \in A \))

<table>
<thead>
<tr>
<th>( x \uparrow a )</th>
<th>( x', \psi \land s_p(x) \neq F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \psi \uparrow x - a \to \psi \uparrow x' )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x \uparrow a )</th>
<th>( x, \psi \land s_p(x) \neq F, \chi \land \psi \neq F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \psi \uparrow x - a \to \psi \uparrow (\chi \land \psi) )</td>
<td></td>
</tr>
</tbody>
</table>

\( s_p(\phi \downarrow x) = \phi \land s_p(x) \)
Table 9
Root signal deletion operator \( (a \in A) \)

<table>
<thead>
<tr>
<th>Expression</th>
<th>RSD0</th>
<th>RSD1</th>
<th>RSD2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_a(\bot) = \bot )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(a) = a )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(x + y) = s_p(x + y) \rightarrow (p_a(x) + p_a(y)) + \neg s_p(x + y) \rightarrow \bot )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(x \cdot y) = p_a(x) \cdot y )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(\phi \land x) = (\phi \rightarrow p_a(x)) + (\neg \phi \rightarrow \bot) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(\phi \land \phi) = p_a(x) \land \phi )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_a(\phi \rightarrow x) = \phi \rightarrow p_a(x) )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10
Signal hiding \( (a \in A) \)

\[
P \Delta \bot = \bot\]

\[
P \Delta (\phi \land \delta) = (\phi[T/P] \lor \phi[F/P]) \land \delta\]

\[
P \Delta (x + y) = P \Delta(s_p(x + y) \land x) + P \Delta(s_p(x + y) \land y)\]

\[
P \Delta (\phi \land \psi \rightarrow a \cdot x) = (\phi[T/P] \lor \phi[F/P]) \land ((\phi \land \psi)[T/P] \rightarrow a \cdot (P \Delta x)) + (\phi \land \psi)[F/P] \rightarrow a \cdot \chi[T/P] \lor \chi[F/P])\]

2.9. Signal hiding

An important operator in applications is the signal hiding operator \( \Delta \) that hides a propositional constant \( P \). We give axioms based on the structure of basic terms in Table 10, and provide semantics in Table 11.

A propositional constant that we want to hide will be replaced by true or by false. If the choice made allows also to satisfy its condition, then an action can be executed. Note that in the signal (both in the root signal and in the terminal signal), the different choices are combined, as the signals of different alternatives are combined according to axiom RSE2 of Table 3. Note that after an action is executed, the choice to replace by true or false must be made all over again, as signals are not persistent.

We give a simple example of a calculation: \( P \Delta(a \cdot (P \land b) + a \cdot (\neg P \land b)) = a \cdot (T \land (T \rightarrow b)) + a \cdot (T \land (T \rightarrow b)) = a \cdot b \). If we assume the linear time law \( a \cdot (x + y) = a \cdot x + a \cdot y \), this leads to the unwanted identity \( a \cdot b = \delta \) (combine with the example in Section 2.3). Thus, this theory only exists in a branching time setting.
The global signal emission operator of Section 2.7, the root signal operator and
root signal deletion operator of Section 2.8 and the signal hiding operator here can all
be eliminated from closed terms, using the axioms given. Thus, we have the basic term
lemma also for this extended signature.

We can establish that the extended theory is a conservative extension of BPAPs,
and that the axiomatisation is sound and complete for the term model modulo
bisimulation. As the axiomatisation of signal hiding involves substitution, the theory
of [10] is needed in order to do so.

2.10. Recursion

We will not deal with full recursion in this paper, as in our examples, it is enough to
consider linear recursion. It is not difficult, but rather involved, to define operational
semantics of general recursion.

A recursion equation is linear if it is of the form

\[ X = \zeta \land \delta + \sum_{i=1}^{n} \phi_i : \rightarrow a_i \cdot X_i + \sum_{j=1}^{m} \psi_j : \rightarrow b_j \land \chi_j \]

where \( \zeta, \phi_i, \psi_j, \xi_j \notin [F] \), \( a_i \in A, b_j \in A_d \), and the \( X, X_i \) are recursion variables. The
operational semantics for processes given by systems of such equations is easily given:
for an equation as above we have for (a solution of) \( X \) the following rules:

\[ X^\phi_i.a_i \rightarrow X_i, \quad X^\psi_j.b_j \rightarrow \chi_j, \quad s_{\rho}(X) = \zeta \]

Recursion equations in the following can be brought into linear form without much
trouble.

2.11. Some examples

A specification of a (FIFO) queue can be given as follows. This queue signals
whether or not it is empty. We have a given finite data set \( D \), and the following
specification has variables indexed by sequences over \( D \).

The atomic action \( r(d) \) stands
for the reading or input of data \( d \), \( s(d) \) stands for sending or output of \( d \):

\[ Q_e = \text{empty} \land \sum_{d \in D} r(d) \cdot Q_d \]

\[ Q_{od} = \neg \text{empty} \land s(d) \cdot Q_d + \sum_{e \in D} r(e) \cdot Q_{eod}. \]

A similar specification can be given for a bag. Now, variables are indexed by
multi-sets.

\[ B_b = \text{empty} \land \sum_{d \in D} r(d) \cdot B_{(d)} \]

\[ V \neq \emptyset \Rightarrow B_v = \neg \text{empty} \land \sum_{d \in V} s(d) \cdot B_{(d) + \{d\}} + \sum_{d \in D} r(d) \cdot B_{(d) \cup \{d\}}. \]
3. Parallel composition

In this section, we extend the basic theory of Section 2 with parallel composition. First, we consider parallel composition without synchronisation or communication, the so-called free merge.

3.1. PAPS

The theory PAPS (process algebra with propositional signals) extends BPAps with operators $\sqcap$, $\Vert$, $\|$, and the axioms of Tables 7 and 12.

Note: $x = (F \land a) \land x = F \land (a \land x) = \bot$, and $a \land y = (a \land T) \land x = a \cdot (T \land x)$, where the last expression can be proven equal to $a \cdot x$ for all closed terms.

Note that in axiom M2TS, we use the global signal emission operator on the right-hand side, as the left component of the parallel composition terminates leaving a signal $\phi$, and this signal persists through the execution of actions of the right component.

3.2. Signal inspection

Now we have all the ingredients necessary to describe the inspection of an emitted signal. A very simple example will serve to make the point. Let us consider a traffic light. The set of propositional constants is $\{\text{green}, \text{yellow}, \text{red}\}$.

$TL(\text{green}) = (\text{green} \land \neg \text{yellow} \land \neg \text{red}) \land \text{change} \cdot TL(\text{yellow})$

$TL(\text{yellow}) = (\neg \text{green} \land \text{yellow} \land \neg \text{red}) \land \text{change} \cdot TL(\text{red})$

$TL(\text{red}) = (\neg \text{green} \land \neg \text{yellow} \land \text{red}) \land \text{change} \cdot TL(\text{green})$.

Now we describe a careful car driver.

$CD = \text{approach} \cdot ((\neg \text{green} \Rightarrow \text{stop}) \cdot (\text{green} \Rightarrow \text{start}) \cdot (\neg \text{red} \Rightarrow \text{drive})$

$\lor (\text{green} \Rightarrow \text{drive})$.

Expression $TL(x) \Vert CD$ now describes a correct interaction between light and driver. Note that deadlock occurs if a car does not drive through the intersection fast enough when the light turns green.

| Table 12 |
| Frce mergc (a ∈ A₁) |
| \hline |
| x \Vert y = x \land y + y \Vert x | M1 |
| (a \land \phi) \land x = a \cdot (\phi \land x) | M2TS |
| a \cdot x \land y = a \cdot (x \land y) | M3 |
| (x + y) \land z = x \land z + y \land z | M4 |
| (\phi \land x) \land y = \phi \land (x \land y) | MR3 |
| (\phi \Rightarrow x) \land y = \phi \Rightarrow (x \land y) | MGC |
3.3. ACPps

The theory ACPps (algebra of communicating processes with propositional signals) extends PAps with operators $|$, $\partial_H$, $s_p$, and replaces the axioms of Table 12 by the axioms of the root signal operator and the axioms in Table 13 below. We assume given a partial commutative and associative binary function on $A$, the communication function $\gamma$. In order to explain CMG2, CMG3 notice that any communication action

<table>
<thead>
<tr>
<th>Table 13</th>
<th>Merge with communication and encapsulation ($a \in A_d$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a</td>
<td>b = \gamma(a, b)$ if defined</td>
</tr>
<tr>
<td>$x \parallel y = x \parallel y + y \parallel x + x</td>
<td>y$</td>
</tr>
<tr>
<td>$(a \wedge \phi) \parallel x = a \cdot (x \parallel \phi)$</td>
<td>CM2TS</td>
</tr>
<tr>
<td>$a \cdot x \parallel y = a \cdot (x \parallel y)$</td>
<td>CM3</td>
</tr>
<tr>
<td>$(x + y) \parallel z = x \parallel z + y \parallel z$</td>
<td>CM4</td>
</tr>
<tr>
<td>$(\phi \wedge x) \parallel y = \phi \wedge (x \parallel y)$</td>
<td>CMRS1</td>
</tr>
<tr>
<td>$(\phi \rightarrow x) \parallel y = \phi \rightarrow (x \parallel y)$</td>
<td>CMGC1</td>
</tr>
<tr>
<td>$(a \wedge \phi)(b \wedge \psi) = (a \wedge b) \wedge (\phi \wedge \psi)$</td>
<td>CMSTS</td>
</tr>
<tr>
<td>$(a \wedge \phi) \cdot x = (a \wedge b) \cdot (x \parallel \phi)$</td>
<td>CM5TS</td>
</tr>
<tr>
<td>$\partial_H(a) = a$ if $a \in H$</td>
<td>D1</td>
</tr>
<tr>
<td>$\partial_H(a) = \delta$ if $a \notin H$</td>
<td>D2</td>
</tr>
<tr>
<td>$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$</td>
<td>D3</td>
</tr>
<tr>
<td>$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$</td>
<td>D4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 14</th>
<th>Semantics of ACPps ($a, b, c \in A$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \parallel y \parallel x', y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) \neq F, s_p(x' \parallel y) \neq F$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
<tr>
<td>$x \parallel y \parallel x' \parallel y \parallel x \parallel x' \parallel y \parallel x'$</td>
<td>$s_p(x \parallel y) = s_p(x) \wedge s_p(y)$</td>
</tr>
</tbody>
</table>
involves an action from both sides, so $\phi$ must hold for an action to occur, but the signal of $y$ shows even if $\phi$ does not hold (and no action can occur).

We provide the semantics of ACPps in Table 14. The semantics of PAps can be extracted, by omitting all parts referring to the communication merge operator.

4. State operator

In this section, we extend each of the theories BPAps, PAps, ACPps with the state operator of [2]. The interesting aspect here is, that we allow the state to be (partly) visible to the process, i.e. a state can emit a signal.

4.1. Syntax and semantics

Let us assume that a state operator in the sense of [2] is given by a domain $S$ and functions $act: A \times S \rightarrow A_S$ and $eff: A \times S \rightarrow S$. The expression $\lambda_x(s)$ with $s \in S$ denotes process $x$ working on the state space $S$ with the current state being $s \in S$.

We can assume that there is an additional function $\text{sig}: S \rightarrow B$ which determines for each state the signal that is emitted by that state. The absence of signals is modeled by taking $\text{sig}(s) = T$ for all $s$ of course. Now the eight equations for the state operator are as shown in Table 15, the operational semantics is given in Table 16.

Using a state operator that generates signals one can define signaling processes in such a way that the equations need not contain any signal at all, thus considerably optimizing the notation. We will illustrate this in a simple example.

**Example 4.1.** Let $D$ be a finite alphabet of data, and let $D^*$ be the collection of finite sequences over $D$. The empty sequence is denoted by $\epsilon$ and adding an element $d$ to the

<table>
<thead>
<tr>
<th>Table 15</th>
<th>State operator generating signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_\varepsilon(\varepsilon) = \bot$</td>
<td>SOS0</td>
</tr>
<tr>
<td>$\lambda_\delta(s) = \text{sig}(s) \land \delta$</td>
<td>SOS1</td>
</tr>
<tr>
<td>$\lambda_{act}(a, s) = \text{sig}(s) \land \text{act}(a, s) \land \text{sig}(\text{eff}(a, s))$</td>
<td>SOS2</td>
</tr>
<tr>
<td>$\lambda_{act}(a \cdot x) = \text{sig}(s) \land \text{act}(a, s) \land \text{sig}(\text{eff}(a, s))$</td>
<td>SOS3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 16</th>
<th>Operational semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \xrightarrow{a} x', \text{sig}(s) \land s_p(x) \neq F, \text{sig}(\text{eff}(a, s)) \land s_p(x') \neq F, \text{act}(a, s) = b \neq \delta$</td>
<td>$\lambda_x(x') \xrightarrow{a, b} \lambda_{\text{eff}(a, s)}(x')$</td>
</tr>
<tr>
<td>$x \xrightarrow{\delta, \psi} \psi, \text{sig}(s) \land s_p(x) \neq F, \text{sig}(\text{eff}(a, s)) \land \psi \neq F, \text{act}(a, s) = b \neq \delta$</td>
<td>$\lambda_x(x) \xrightarrow{\delta, b} \text{sig}(\text{eff}(a, s)) \land \psi$</td>
</tr>
<tr>
<td>$s_p(\lambda_x(x)) = s_p(x) \land \text{sig}(s)$</td>
<td>$s_p(\lambda_x(x)) = s_p(x) \land \text{sig}(s)$</td>
</tr>
</tbody>
</table>
list \( \sigma \) results in \( \sigma d \). The propositional constants are as follows: \( \text{on}_\text{top}(d) \) (for \( d \in D \)), and \( \text{empty} \).

We will assume that these signals are exclusive, i.e. we will assume that the following formula always holds:

\[
\Phi \equiv \left( \text{empty} \Rightarrow \bigwedge_{d \in D} \neg \text{on}_\text{top}(d) \right) \land \bigwedge_{d \in D} \left( \text{on}_\text{top}(d) \Rightarrow \bigwedge_{e \neq d} \neg \text{on}_\text{top}(e) \land \neg \text{empty} \right).
\]

\( D^* \) will be the state space for a process that represents a stack over \( D \). The signal function \( \text{sig} \) is defined by \( \text{sig}(s) = \text{empty} \), \( \text{sig}(\sigma d) = \text{on}_\text{top}(d) \). The atomic actions are:

- \( \text{push}_\text{int}(d) \), \( \text{push}(d) \) for \( d \in D \) (the suffix \( \text{int} \) denotes an intended action),
- \( \text{pop}_\text{int} \), \( \text{pop} \).

The functions \( \text{act} \) and \( \text{eff} \) are given by

\( \text{act}(\text{push}_\text{int}(d), \sigma) = \text{push}(d) \)

(the \( \text{act} \) function transforms an intended action into an actual action),

\( \text{act}(\text{pop}_\text{int}, \sigma) = \text{pop} \),

\( \text{eff}(\text{push}_\text{int}(d), \sigma) = \sigma d \) (the \( \text{eff} \) function gives the resulting contents of the stack),

\( \text{eff}(\text{pop}_\text{int}, \varepsilon) = \varepsilon \),

\( \text{eff}(\text{pop}_\text{int}, \sigma d) = \sigma \).

(For \( \text{act} \) only those cases are given where \( \text{act} \) will not lead to \( \delta \).) The behavior of a stack over \( D \) is given by the following process definition.

\[
S = \Phi \prod_{n} \lambda_{\varepsilon} \left( \left( \sum_{d \in D} \text{push}_\text{int}(d) + \text{pop}_\text{int} \right) \cdot S \right).
\]

**Example 4.2.** In this example two buffers \( A \) and \( B \) with data from the finite set \( D \) are maintained in the state. Both buffers have length \( k > 1 \). The process to be defined allows to read data in both buffers in a concurrent mode. For both buffers \( A \) and \( B \) there is a propositional constant: \( \text{open}_A \) indicates that there is still room in \( A \) (likewise for \( B \)). When both buffers have been loaded the process \( C \) compares the contents of the buffers. The comparison will send value \( \text{true} \) if the buffers were equal and \( \text{false} \) otherwise. Thereafter the buffers are made empty again and the process restarts. We will describe the system in a top-down fashion, first explaining the overall architecture and then completing the details:

\[
\text{SYSTEM} = \lambda_{(\varepsilon, \varepsilon)} (A \parallel B \parallel C).
\]
The state consists of a pair of buffers $A$ and $B$. Initially, both are empty. The signals produced by a state $\langle \alpha, \beta \rangle$ are as follows:

$$\text{sig}(\langle \alpha, \beta \rangle) = \begin{cases} 
  \text{open}A \land \text{open}B & \text{if length}(\alpha) < k \text{ and length}(\beta) < k, \\
  \neg\text{open}A \land \text{open}B & \text{if length}(\alpha) = k \text{ and length}(\beta) < k, \\
  \text{open}A \land \neg\text{open}B & \text{if length}(\alpha) < k \text{ and length}(\beta) = k, \\
  \neg\text{open}A \land \neg\text{open}B & \text{if length}(\alpha) = k \text{ and length}(\beta) = k.
\end{cases}$$

The processes $A$, $B$, $C$ are defined by

$$A = \left( \text{open}A \rightarrow \sum_{d \in D} \text{read}A(d) \right) \cdot A$$

$$B = \left( \text{open}B \rightarrow \sum_{d \in D} \text{read}B(d) \right) \cdot B$$

$$C = (\neg\text{open}A \land \neg\text{open}B \rightarrow \text{comp}) \cdot C.$$

The next step is to explain the effect function.

$$\text{eff}(\text{read}A(d), \langle \alpha, \beta \rangle) = \langle \alpha d, \beta \rangle$$

$$\text{eff}(\text{read}B(d), \langle \alpha, \beta \rangle) = \langle \alpha, \beta d \rangle$$

$$\text{eff}(\text{comp}, \langle \alpha, \beta \rangle) = \langle \epsilon, \epsilon \rangle.$$

Finally, the action function must be specified:

$$\text{act}(\text{comp}, \langle \alpha, \beta \rangle) = \begin{cases} 
  \text{write}(\text{true}) & \text{if } \alpha = \beta, \\
  \text{write}(\text{false}) & \text{otherwise}.
\end{cases}$$

and the action function is the identity otherwise.

The use of the state operator in this example is hard to avoid because of the parallel reading of data that must be used simultaneously later on. This issue is worked out in [16].

5. Abstraction

We give some remarks (without proof) about how silent step and abstraction could be introduced in the setting of branching bisimulation of [11]. The thoughts in this section should be read as a starting point for further research.

5.1. ACP^PS

The theory ACP^PS extends ACPps by the addition of a special constant $\tau \notin A$, the silent step, and a unary operator $\tau_I$ for each $I \subseteq A$, the abstraction operator. As axioms we have all axioms of ACPps, with now $a, b \in A \cup \{\delta, \tau\}$, plus the additional
axioms of Table 17 below. Note that the axiom \(x \cdot \tau = x\) of ACP' does not hold any more, and so we do not have a conservative extension of ACP'. It is easy to see why this axiom cannot be kept: \(a \cdot \tau \cdot \bot = a \cdot \delta\) should hold: execution of \(a\) leads to a consistent state, but \(\tau\) cannot be executed for it leads to an inconsistent state; on the other hand \(a \cdot \bot = \delta\), so \(a \cdot \tau \cdot \bot \neq a \cdot \bot\) and so \(a \cdot \tau \neq a\).

5.2. Semantics

The operational semantics now also has arrow labels of the form \(\phi, \tau\). In the previous rules, we now have \(a \in A \cup \{\tau\}\). The additional rules for the abstraction operator are shown in Table 18.

With this comes a new definition of bisimulation. In the following, \(x, y, x', y', \ldots\) range over terms and \(\phi, \psi, \ldots\) range over propositions. A relation \(R\) on process expressions is a branching bisimulation when the following holds:

(i) if \(x R y\) then \(s_\phi(x) = s_\phi(y)\)

(ii) if \(x R y\) and \(x^{\phi,a} \rightarrow x'\), then either.

(a) \(a = \tau, s_\phi(x) \Vdash \phi = T\) and \(x' R y\), or

(b) for all valuations \(v\) such that \(v(s_\phi(x) \land \phi) = T\), there are propositions 
   \(\psi_1, \ldots, \psi_n (n \geq 0)\) and \(y_1, y_1, \ldots, y_n\) such that \(s_\phi(x) \Vdash \psi_i = T\) for all \(i\), \(v(\psi) = T\), \(y_1^{\psi_1}, \ldots, y_n^{\psi_n}\) and \(x' R y_i\) for all \(i\) and \(x' \R y_i\).

(iii) if \(x R y\) and \(x^{\phi,a} \rightarrow \xi\), then for all valuations \(v\) such that \(v(s_\phi(x) \land \phi) = T\), there are propositions, \(\chi, \psi_1, \ldots, \psi_n (n \geq 0)\) and expressions \(y_1, \ldots, y_n\) such that 
   \(\xi = \chi, s_\phi(x) \Vdash \psi_i = T\) for all \(i\), \(v(\psi) = T\), \(y_1^{\psi_1}, \ldots, y_n^{\psi_n}\) and \(x R y_i\) for all \(i\).

(iv), (v) Like (ii), (iii) with the role of \(x\) and \(y\) interchanged.

Table 17
Silent step and abstraction

<table>
<thead>
<tr>
<th>Rule</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x \cdot (s_\psi(y) \land (\tau \cdot (y + \tau) + z)) = x \cdot (y + z)</td>
<td>BS</td>
</tr>
<tr>
<td>(\tau_\phi(x \cdot y) = \tau_\phi(x) \cdot \tau_\phi(y))</td>
<td>T14</td>
</tr>
<tr>
<td>(\tau_\phi(a) = \begin{cases} a &amp; \text{if } a \notin I \ \tau &amp; \text{if } a \in I \end{cases})</td>
<td>T11</td>
</tr>
<tr>
<td>(\tau_\phi(x + y) = \tau_\phi(x) + \tau_\phi(y))</td>
<td>T12</td>
</tr>
<tr>
<td>(\tau_\phi(x + y) = \tau_\phi(x) + \tau_\phi(y))</td>
<td>T13</td>
</tr>
</tbody>
</table>

Table 18
Semantics of ACP'p's

<table>
<thead>
<tr>
<th>Rule</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x^{\phi,a} x', a \notin I)</td>
<td>(x^{\phi,a} \psi, a \notin I)</td>
</tr>
<tr>
<td>(\tau_\phi(x) = \tau_\phi(x'))</td>
<td>(\tau_\phi(x^{\phi,a} \psi) = \tau_\phi(x)^{\phi,a} \psi)</td>
</tr>
<tr>
<td>(\tau_\phi(x) = \tau_\phi(x'))</td>
<td>(\tau_\phi(x^{\phi,a} \psi, a \in I) = \tau_\phi(x) = s_\phi(x))</td>
</tr>
<tr>
<td>(\tau_\phi(x)^{\phi,a} \psi = \tau_\phi(x)^{\phi,a} \psi)</td>
<td>(\tau_\phi(x) = \tau_\phi(x'))</td>
</tr>
<tr>
<td>(\tau_\phi(x) = \tau_\phi(x'))</td>
<td>(\tau_\phi(x) = \tau_\phi(x'))</td>
</tr>
</tbody>
</table>
We say a branching bisimulation \( R \) satisfies the root condition for \( x \) and \( y \) if \( x \sim R y \) and in addition:

(vi) if \( x \xrightarrow[\phi, a]{} x' \), then for all valuations \( v \) such that \( v(s_p(x) \land \phi) = T \), there is a proposition \( \psi \) and a term \( y' \) such that \( v(\psi) = T, y \xrightarrow[\psi]{} y' \) and \( x' \sim R y' \)

(vii) if \( x \xrightarrow[\phi, a]{} \xi \) then for all valuations \( v \) such that \( v(s_p(x) \land \phi) = T \), there are propositions \( \psi, \chi \) such that \( v(\psi) = T, y \xrightarrow[\psi]{} \chi \) and \( \xi = \chi \)

(viii) (ix) Like (vii), (vii) with the roles of \( x \) and \( y \) interchanged.

We call two expressions \( x, y \) branching bisimilar, notated \( x \sim_R y \), if there is a branching bisimulation relating \( x \) and \( y \). Two expressions \( x, y \) are rooted branching bisimilar, \( x \sim_{rb} y \), if there is a branching bisimulation that satisfies the root condition for \( x \) and \( y \).

Open problem 5.1. We leave as an open problem whether ACP'ps is a sound and complete axiomatisation of the bisimulation model, i.e. whether for all closed ACP'ps terms \( t, s \) we have

\[ \text{ACP'ps} \vdash t = s \iff t \sim_{rb} s. \]

6. Examples

In this section we discuss a number of examples of the use of signals and inspection.

6.1. Stack

We give a number of alternatives for a specification of a stack. In each case, variables are indexed by sequences of elements, the current contents of the stack. First we specify a stack that shows no signals:

\[ S^1_t = \sum_{d \in D} \text{push}(d) \cdot S^1_d \]

\[ S^1_{sd} = \text{pop} \cdot S^1_t + \text{top}(d) \cdot S^1_d + \sum_{e \in D} \text{push}(e) \cdot S^1_{ade}. \]

Next, we add a signal showing the top of the stack:

\[ S^2_t = \sum_{d \in D} \text{push}(d) \cdot S^2_d \]

\[ S^2_{sd} = \text{pop} \cdot S^2_t + \text{top} \cdot (\text{show}(d) \land S^2_{sd}) + \sum_{e \in D} \text{push}(e) \cdot S^2_{ade}. \]

In the third specification, we add a signal empty, and also allow actions top, pop in case the stack is empty. If this happens, an error signal is emitted, and no
further action is possible:

\[ S_3^a = \text{empty} \land \sum_{d \in D} \text{push}(d) \cdot S_3^a + \text{top} \land \text{error} + \text{pop} \land \text{error} \]

\[ S_{ad}^3 = \text{pop} \cdot S_3^a + \text{top} \cdot (\text{show}(d) \land S_{ad}^3) + \neg \text{empty} \land \sum_{e \in D} \text{push}(e) \cdot S_{ade}^3. \]

The fourth specification has a state of underflow, when an empty stack is popped. A subsequent push leads out of the error situation:

\[ S_4^a = \text{empty} \land \sum_{d \in D} \text{push}(d) \cdot S_4^a + \text{top} \land \text{error} + \text{pop} \land U^4 \]

\[ U^4 = \text{underflow} \land \sum_{d \in D} \text{push}(d) \cdot S_4^a + \text{top} \land \text{error} + \text{pop} \land U^4 \]

\[ S_{ad}^4 = \text{pop} \cdot S_4^a + \text{top} \cdot (\text{show}(d) \land S_{ad}^4) + \neg \text{empty} \land \sum_{e \in D} \text{push}(e) \cdot S_{ade}^4. \]

The fifth stack keeps functioning, when a pop or top is executed on an empty stack:

\[ S_5^a = \text{empty} \land \sum_{d \in D} \text{push}(d) \cdot S_5^a + \text{top} \land (\text{show}(\bot) \land S_5^a) + \text{pop} \land (\text{error} \land S_5^a) \]

\[ S_{ad}^5 = \text{pop} \cdot S_5^a + \text{top} \cdot (\text{show}(d) \land S_{ad}^5) + \neg \text{empty} \land \sum_{e \in D} \text{push}(e) \cdot S_{ade}^5. \]

The sixth specification, a pop or top executed on an empty stack leads to an irrecoverable error state, but actions can still be executed:

\[ S_6^a = \text{empty} \land \sum_{d \in D} \text{push}(d) \cdot S_6^a + (\text{top} + \text{pop}) \cdot \left( \text{error} \land \left( \text{top} + \text{pop} + \sum_{d \in D} \text{push}(d) \right) \right) \]

\[ S_{ad}^6 = \text{pop} \cdot S_6^a + \text{top} \cdot (\text{show}(d) \land S_{ad}^6) + \neg \text{empty} \land \sum_{e \in D} \text{push}(e) \cdot S_{ade}^6. \]

6.2. Communicating buffers

In this example we study a system where both signal inspection and communication play a role. We will show that communication can be replaced by inspection. We start out from a standard specification of one element buffers, that in addition always signal the contents on the output port (all specifications can be simply brought into a form that uses iteration rather than recursion). The buffer \( B^{ij} \) has input port \( i \) and output port \( j \), and can buffer messages from some finite set \( D \). Let \( \emptyset \notin D \). The signal \( \text{show}_j(d) \) means that message \( d \) is offered at port \( j(d \in D) \), \( \text{show}_j(\emptyset) \) means that the buffer is empty:

\[ B^{ij} = \text{show}_j(\emptyset) \land \sum_{d \in D} \text{read}_i(d) \cdot B^{ij}_d \]

\[ B^{ij}_d = \text{show}_j(d) \land \text{send}_j(d) \cdot B^{ij} \]

\[ X = \delta_H(B^{12} \parallel B^{23}) \]

where \( \text{send}_2(d) \land \text{read}_2(d) = \text{comm}_2(d) \) (communication gives \( \delta \) otherwise), and \( H = \{ \text{read}_2(d), \text{send}_2(d) : d \in D \} \).
Some calculations result in the following recursive specification:

\[
X = (\text{show}_2(\emptyset) \land \text{show}_3(\emptyset)) \land \sum_{d \in D} \text{read}_1(d) \cdot X^d_1
\]

\[
X^d_1 = (\text{show}_2(d) \land \text{show}_3(\emptyset)) \land \text{comm}_2(d) \cdot X^d_2
\]

\[
X^d_2 = (\text{show}_2(\emptyset) \land \text{show}_3(d)) \land \text{send}_3(d) \cdot X + \sum_{e \in D} \text{read}_1(e) \cdot X^d_{3e}
\]

\[
X^d_{3e} = (\text{show}_2(e) \land \text{show}_3(d)) \land \text{send}_3(d) \cdot X^d_1
\]

Hiding all signals gives back the usual specification of two coupled one-element buffers (as in [4, p. 106]).

As a first step in replacing communication by inspection, we omit the parametrisation of the communication action in favor of signal inspection. To make this correct, we need to require that signals are exclusive, formalised by proposition

\[
\Phi_i \equiv \left( \text{show}_i(\emptyset) \lor \bigwedge_{d \in D} \neg \text{show}_i(d) \right) \land \bigwedge_{e \neq d} \left( \text{show}_i(d) \lor \neg \text{show}_i(\emptyset) \land \neg \text{show}_i(e) \right).
\]

\[
C^{ij} = \text{show}_j(\emptyset) \land \sum_{d \in D} \text{show}_i(d) \land \text{read}_i \cdot C^{ij}_d
\]

\[
C^{ij}_d = \text{show}_j(d) \land \text{send}_j \cdot C^{ij}_j
\]

\[
Y = (\Phi_1 \land \Phi_2 \land \Phi_3) \land \delta_H(C^{12} || C^{23}),
\]

where communication is given by \text{send}_2 | \text{read}_2 = \text{comm}_2, and encapsulation by \(H = \{\text{read}_2, \text{send}_2\} \).

Some calculations result in the following recursive specification (omitting the exclusivity propositions):

\[
Y = (\text{show}_2(\emptyset) \land \text{show}_3(\emptyset)) \land \sum_{d \in D} \text{show}_1(d) \land \text{read}_1 \cdot Y^d_1
\]

\[
Y^d_1 = (\text{show}_2(d) \land \text{show}_3(\emptyset)) \land \text{comm}_2 \cdot Y^d_2
\]

\[
Y^d_2 = (\text{show}_2(\emptyset) \land \text{show}_3(d)) \land \text{send}_3 \cdot Y + \sum_{e \in D} \text{show}_1(e) \land \text{read}_1 \cdot Y^d_{3e}
\]

\[
Y^d_{3e} = (\text{show}_2(e) \land \text{show}_3(d)) \land \text{send}_3 \cdot Y^d_1
\]

Let us now abstract from actions and signals at port 2. Put \(I = \{\text{comm}_2\} \) and \(\text{show}_2 = \{\text{show}_2(d) : d \in D\} \lor \{\text{show}_2(\emptyset)\} \), and derive the following specification for \(Z = \text{show}_2 \Delta \tau_I(Y)\):

\[
Z = \text{show}_2 \Delta \tau_I(Y) = \text{show}_3(\emptyset) \land \sum_{d \in D} \text{show}_1(d) \land \text{read}_1 \cdot Z^d_1
\]

\[
Z^d_1 = \text{show}_2 \Delta \tau_I(Y^d_1) = \text{show}_3(\emptyset) \land \tau \cdot Z^d_2
\]

\[
Z^d_2 = \text{show}_2 \Delta \tau_I(Y^d_2) = \text{show}_3(d) \land \text{send}_3 \cdot Z + \sum_{e \in D} \text{show}_1(e) \land \text{read}_1 \cdot Z^d_{3e}
\]

\[
Z^d_{3e} = \text{show}_2 \Delta \tau_I(Y^d_{3e}) = \text{show}_3(d) \land \text{send}_3 \cdot Z^d_1
\]
Next, we can do away with the synchronisation in favour of two extra signals at the connecting port. First, we consider the specification without extra actions.

\[
E^*_i = (\text{show}_i(\emptyset) \land \neg \text{flag}_i) \land \sum_{d \in D} (\text{show}_i(d) \land \neg \text{flag}_i) \rightarrow \text{read}_i \cdot E^*_i
\]

\[
E^*_d = (\text{show}_d(d) \land \text{flag}_d) \land (\text{show}_i(\emptyset) \land \neg \text{flag}_i) \rightarrow \text{send}_d \cdot E^*_d
\]

\[
W = (\Phi_1 \land \Phi_2 \land \Phi_3) \downarrow E^{12} \parallel E^{23}
\]

where this is the free merge, i.e. this is a specification in PAPs. Unfortunately, this system does not behave as a two-item buffer but as a one-item buffer. If we want the intended behaviour, we have to put in extra actions:

\[
F^*_i = (\neg \text{ready}_i \land \text{show}_i(\emptyset) \land \text{flag}_i) \land (\text{show}_i(d) \land \neg \text{flag}_i) \rightarrow \text{read}_i \cdot F^*_d
\]

\[
F^*_d = (\neg \text{ready}_d \land \text{show}_d(d) \land \text{flag}_d) \land (\text{ready}_i \land \neg \text{flag}_i) \rightarrow \text{send}_d \cdot G^*_i
\]

\[
G^*_d = (\neg \text{ready}_d \land \text{show}_d(d) \land \text{flag}_d) \land (\text{ready}_d \land \neg \text{flag}_d) \rightarrow \text{reset}_d \cdot F^*_j
\]

\[
V = (\Phi_1 \land \Phi_2 \land \Phi_3) \downarrow F^{12} \parallel F^{23}
\]

where this is the free merge, i.e. this is a specification in PAPs.

We can derive the following specification for \(V\):

\[
V = (\text{ready}_1 \land \text{ready}_2 \land \text{show}_2(\emptyset) \land \text{show}_3(\emptyset) \land \neg \text{flag}_2 \land \neg \text{flag}_3)^\lor
\]

\[
\sum_{d \in D} (\text{flag}_1 \land \text{show}_1(d)) \rightarrow \text{read}_1 \cdot V_{1,d}
\]

\[
V_{1,d} = (\text{ready}_1 \land \text{ready}_2 \land \text{show}_2(d) \land \text{show}_3(\emptyset) \land \neg \text{flag}_2 \land \neg \text{flag}_3)^\lor
\]

\[
\neg \text{flag}_1 : \rightarrow \text{send}_2 \cdot V_{2,d}
\]

\[
V_{2,d} = (\neg \text{ready}_1 \land \text{ready}_2 \land \text{show}_2(d) \land \text{show}_3(\emptyset) \land \text{flag}_2 \land \neg \text{flag}_3)^\lor \text{read}_2 \cdot V_{3,d}
\]

\[
V_{3,d} = (\neg \text{ready}_1 \land \neg \text{ready}_2 \land \text{show}_2(d) \land \text{show}_3(d) \land \text{flag}_2 \land \neg \text{flag}_3)^\lor
\]

\[
\text{reset}_2 \cdot V_{4,d}
\]

\[
V_{4,d} = (\text{ready}_1 \land \neg \text{ready}_2 \land \text{show}_2(\emptyset) \land \text{show}_3(d) \land \neg \text{flag}_2 \land \neg \text{flag}_3)^\lor
\]

\[
\text{ready}_3 : \rightarrow \text{send}_3 \cdot V_{5,d} + \sum_{e \in D} (\text{flag}_1 \land \text{show}_1(e)) \rightarrow \text{read}_1 \cdot V_{6,de}
\]

\[
V_{5,d} = (\text{ready}_1 \land \neg \text{ready}_2 \land \text{show}_2(\emptyset) \land \text{show}_3(d) \land \neg \text{flag}_2 \land \text{flag}_3)^\lor
\]

\[
\neg \text{ready}_3 : \rightarrow \text{reset}_3 \cdot V + \sum_{e \in D} (\text{flag}_1 \land \text{show}_1(e)) \rightarrow \text{read}_1 \cdot V_{7,de}
\]

\[
V_{6,de} = (\neg \text{ready}_1 \land \neg \text{ready}_2 \land \text{show}_2(e) \land \text{show}_3(d) \land \neg \text{flag}_2 \land \neg \text{flag}_3)^\lor
\]

\[
\text{ready}_3 : \rightarrow \text{send}_3 \cdot V_{7,de}
\]

\[
V_{7,de} = (\neg \text{ready}_1 \land \neg \text{ready}_2 \land \text{show}_2(e) \land \text{show}_3(d) \land \neg \text{flag}_2 \land \text{flag}_3)^\lor
\]

\[
\neg \text{ready}_3 : \rightarrow \text{reset}_3 \cdot V_{1,e}
\]
Now we hide all signals at port 2, and the additional signals introduced in the last step, i.e. we hide the propositional constants in the set

\[ P = \{\text{show}_2(d) : d \in D\} \cup \{\text{show}_2(\emptyset), \text{ready}_1, \text{ready}_2, \text{ready}_3, \text{flag}_1, \text{flag}_2, \text{flag}_3\} \]

We obtain the following specification for \( P \triangle V \):

\[
P \triangle V = \text{show}_3(\emptyset) \overset{\sum_{d \in D} \text{show}_1(d)}{\Rightarrow} \text{read}_1 \cdot (P \triangle V_{1,d})
\]

\[
P \triangle V_{1,d} = \text{show}_3(\emptyset) \overset{\text{send}_2}{\Rightarrow} (P \triangle V_{2,d})
\]

\[
P \triangle V_{2,d} = \text{show}_3(\emptyset) \overset{\text{read}_2}{\Rightarrow} (P \triangle V_{3,d})
\]

\[
P \triangle V_{3,d} = \text{show}_3(d) \overset{\text{reset}_2}{\Rightarrow} (P \triangle V_{4,d})
\]

\[
P \triangle V_{4,d} = \text{send}_3 \cdot (P \triangle V_{5,d}) + \sum_{e \in D} \text{show}_1(e) \overset{\text{read}_1}{\Rightarrow} (P \triangle V_{6,d})
\]

\[
P \triangle V_{5,d} = \text{reset}_3 \cdot (P \triangle V) + \sum_{e \in D} \text{show}_1(e) \overset{\text{read}_1}{\Rightarrow} (P \triangle V_{7,d})
\]

As the next step, we abstract from the action set \( I = \{\text{read}_2, \text{send}_2, \text{comm}_2, \text{reset}_2, \text{reset}_3\} \). We obtain

\[
\tau_I(P \triangle V) = \text{show}_3(\emptyset) \overset{\sum_{d \in D} \text{show}_1(d)}{\Rightarrow} \text{read}_1 \cdot \tau_I(P \triangle V_{1,d})
\]

\[
\tau_I(P \triangle V_{1,d}) = \text{show}_3(\emptyset) \overset{\tau \cdot \tau_I(P \triangle V_{2,d})}{\Rightarrow}
\]

\[
\tau_I(P \triangle V_{2,d}) = \text{show}_3(\emptyset) \overset{\tau \cdot \tau_I(P \triangle V_{3,d})}{\Rightarrow}
\]

\[
\tau_I(P \triangle V_{3,d}) = \text{show}_3(d) \overset{\tau \cdot \tau_I(P \triangle V_{4,d})}{\Rightarrow} \]

\[
+ \sum_{e \in D} \text{show}_1(e) \overset{\text{read}_1}{\Rightarrow} \tau_I(P \triangle V_{6,d})
\]

\[
\tau_I(P \triangle V_{5,d}) = \text{show}_3(d) \overset{\tau \cdot \tau_I(P \triangle V)}{\Rightarrow} + \sum_{e \in D} \text{show}_1(e) \overset{\text{read}_1}{\Rightarrow} \tau_I(P \triangle V_{7,d})
\]

\[
\tau_I(P \triangle V_{6,d}) = \text{show}_3(d) \overset{\text{send}_3 \cdot \tau_I(P \triangle V_{7,d})}{\Rightarrow}
\]

\[
\tau_I(P \triangle V_{7,d}) = \text{show}_3(d) \overset{\tau \cdot \tau_I(P \triangle V_{1,d})}{\Rightarrow}
\]

Using the laws for branching bisimulation, we can reduce this to

\[
\tau_I(P \triangle V) = \text{show}_3(\emptyset) \overset{\sum_{d \in D} \text{show}_1(d)}{\Rightarrow} \text{read}_1 \cdot \tau_I(P \triangle V_{2,d})
\]

\[
\tau_I(P \triangle V_{2,d}) = \text{show}_3(\emptyset) \overset{\tau \cdot \tau_I(P \triangle V_{4,d})}{\Rightarrow}
\]
\[
\tau_I(P \Delta V_{4,d}) = show_3(d) \uparrow send_3 \cdot \tau_1(P \Delta V) \\
+ \sum_{e \in \mathcal{D}} show_1(e) : \rightarrow read_1 \cdot \tau_1(P \Delta V_{6,d}) \\
\tau_I(P \Delta V_{6,d}) = show_3(d) \uparrow send_3 \cdot \tau_I(P \Delta V_{1,e})
\]

and we see that this is the same specification as for \( Z \) above.

7. Conclusions

We conclude that we have described the interplay between the execution of actions of a process (giving the state changes, the dynamics of a process) and the propositions that hold in a state of a process (giving the static part of a process). The signal emitted by a state is a proposition that constitutes the visible part of this state, and an action leading out of a state can be conditional on a proposition that should hold in a state. In a parallel composition of two processes, an action executed by one process can be conditional, depending on the signal emitted by the other process. This described a mechanism called signal inspection or signal observation.

We have given some small examples. Further work would be to construct larger examples, and to extend both logic and process theory, for instance, with timing constructs.

Acknowledgements

Helpful remarks by the anonymous referees, Jan Joris Vereijken (Eindhoven University of Technology) and Joris Boselie, Leon Moonen (University of Amsterdam) are appreciated.

References