Process Algebra with Propositional Signals

J.C.M. Baeten
Department of Computer Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

J.A. Bergstra
Programming Research Group, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
and
Department of Philosophy, Utrecht University,
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

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.

1991 Mathematics Subject Classification: 68Q60, 68Q10, 68Q40.
Key words & Phrases: process algebra, process logic, signals.

Note: This research was supported in part by ESPRIT basic research action 7166,
CONCUR2. The second author is also partially supported by ESPRIT basic research action
6454, CONFERENCE.

1. INTRODUCTION.
This paper can be viewed as a revision and simplification of [BAB92] in which we have introduced so-
called signals as labels for states in processes (see also [BRO90]). Though useful in a multitude of
examples, it has turned out that the mechanism of signal observation of [BAB92] 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 [BAB92]) 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 $x \triangleleft \phi \triangleright y$) 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 position, 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 [BAB88]) or global states (see [GRP94]), 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 [BEP94]. Clearly, our approach, based on ACP [BeK84] can be adapted to CCS [Mil80], MEIJE [AUB84] or ATP [NiS94] without much effort. Adaptation to CSP [BRHR84] 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 · (sequential composition). The core system BPA (Basic Process Algebra) over this signature has the axioms A1-5 of table 1 below ($x,y,z \in P$). The constant $\partial$ denoting inaction (or deadlock) is added to the language with axioms A6,7. 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 NE1-3 of table 1 ($x \in P$, $a \in A$). Nonexistence stands for an inconsistent state of a process: such a state can never be exited (NE1,2) and also, it is impossible to enter such a state from a consistent state (NE3). This signature and these axioms together constitute the theory $\text{BPA}_\perp$, Basic Process Algebra with inaction and nonexistence.
Process algebra with propositional signals

2.2 CONDITIONALS.
Besides the sort of processes $\mathbb{P}$, we will have a second sort $\mathbb{B}$. 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 $\lor, \land, \Rightarrow, \neg$ (disjunction, conjunction, implication, negation). In derivations we can use identities of propositional logic. We use letters $\phi, \psi$ to range over $\mathbb{B}$.

As in [BAB92], we use the guarded command $\phi.f \rightarrow x P P$. The expression $\phi.f \rightarrow x$ is read as if $\phi$ then $x$. We have the basic axioms of table 2 below, using the numbering of [BAB92].

\[
\begin{align*}
T.f \rightarrow x &= x & \text{GC1} \\
F.f \rightarrow x &= \delta & \text{GC2} \\
\phi \rightarrow \delta &= \delta & \text{GC9} \\
\phi \rightarrow (x + y) &= (\phi \rightarrow x) + (\phi \rightarrow y) & \text{GC10} \\
\phi \rightarrow (x \cdot y) &= (\phi \rightarrow x) \cdot y & \text{GC13} \\
(\phi \lor \psi).f \rightarrow x &= (\phi.f \rightarrow x) + (\psi.f \rightarrow x) & \text{GC11} \\
\phi.f \rightarrow (\psi.f \rightarrow x) &= (\phi \land \psi) \rightarrow x & \text{GC12}
\end{align*}
\]

TABLE 2. Conditionals over propositional logic.

2.3 ROOT AND TERMINAL SIGNAL EMISSION OPERATORS.
The next operators to be introduced are the signal emission operators. $\check{\cdot}$ is the root signal emission operator and $\hat{\cdot}$ is the terminal signal emission operator (we take this notation from [BEP94]). 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. If one is interested solely in processes that emit signals exclusively in nonterminal states one may as well forget about the terminal signal emission operator. Leaving out all axioms involving terminal emission from the coming sections one will obtain an appropriate description of root signal emission. The following equations are added to $\text{BPA}_{\perp}$ thus obtaining $\text{BPA}_{\text{ps}}$ (BPA with Propositional Signals).
The first axiom expresses the fact that the root of a sequential product is the root of its first component. Axiom RS2 can be given in a more symmetric form as follows:

\[(\phi \wedge x) + (\psi \wedge y) = (\phi \wedge \psi) \wedge (x + y).\]

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:

\[a \cdot ((\phi \wedge x) + (\neg \phi \wedge b)) = a \cdot ((\phi \wedge \neg b) \wedge b) = a \cdot (F \wedge b) = a \cdot \bot = \delta.\]

The last axiom RSE8 is the signal inspection rule. If a signal \(\phi\) is emitted, then \(\phi\) holds in the current state (this is why the signal \(F\) denotes an inconsistent state). Note the following generalisation of RSE8:

\[\phi \wedge x = (\phi \wedge \delta) + x.\]

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 axiom system BPAps, Basic Process Algebra with Propositional Signals, consists of all axioms from tables 1-4.
2.4 BASIC TERMS. 

Define a set of basic terms $\mathcal{B}$ as follows.

i. $\bot \in \mathcal{B}$

ii. $\phi \in B - \{F\} \Rightarrow \phi \vdash \delta \in \mathcal{B}$

iii. $\phi, \psi \in B - \{F\}, a \in A \Rightarrow (\phi \stackrel{\rightarrow}{\rightarrow} a \vdash \psi) \in \mathcal{B}$

iv. $\phi \in B - \{F\}, a \in A, t \in B \Rightarrow (\phi \stackrel{\rightarrow}{\rightarrow} a \vdash t) \in \mathcal{B}$

v. $t, s \in B \Rightarrow t + s \in \mathcal{B}$

Note that each basic term can be written as $\bot$ or in the form:

$$\zeta \vdash \delta + \sum_{i=1}^{n} \phi_{i} \stackrel{\rightarrow}{\rightarrow} a_{i} \vdash x_{i} + \sum_{j=1}^{m} \psi_{j} \vdash b_{j} \vdash \chi_{j}$$

or, equivalently,

$$\zeta \vdash \left(\sum_{i=1}^{n} \phi_{i} \stackrel{\rightarrow}{\rightarrow} a_{i} \vdash x_{i} + \sum_{j=1}^{m} \psi_{j} \vdash b_{j} \vdash \chi_{j}\right).$$

When a basic term has this form, we call $\zeta$ its root signal, and the subterms $\phi_{i} \stackrel{\rightarrow}{\rightarrow} a_{i} \vdash x_{i}$, $\psi_{j} \vdash b_{j} \vdash \chi_{j}$ its summands.

2.5 BASIC TERM LEMMA. For all closed terms $s$ there is a basic term $t$ such that $\text{BPAps} \vdash t = s$.

SKETCH OF PROOF. Basically, this follows from the fact that the term rewriting system consisting of axioms A4,5 from table 1 together with all axioms from tables 2,3,4 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 $\vdash > \rightarrow > \cdot > + > \delta$, $+ > \vdash > \bot \land \lor \Rightarrow$, giving $\cdot$ the lexicographical status for the first argument, and $\rightarrow$, $\vdash$, $\vdash$ for the second argument). Each normal form of this term rewriting system can easily be converted into a basic term.

2.6 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:

- \( x \phi, a \rightarrow x' \) term \( x \) can do an \( a \)-step under condition \( \phi \) to term \( x' \)
- \( x \phi, a \rightarrow \¥ \) term \( x \) can do a terminating \( a \)-step under condition \( \phi \) leaving terminal signal \( \¥ \)
- \( s_p(x) = \phi \) 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. This SOS specification is in the path format of [BAV93].

\[
\begin{align*}
\text{a} & \rightarrow \\
\frac{x \phi, a \rightarrow (x+y) \neq F}{x+y \phi, a \rightarrow} \\
\frac{x \phi, a \rightarrow (x) \neq F}{\psi \not\setminus x \phi, a \rightarrow} \\
\frac{x \phi, a \rightarrow \¥}{x \not\setminus \psi \phi, a \rightarrow} \\
\frac{(\psi; \rightarrow x) \phi \land \¥ a \rightarrow}{x \phi, a \rightarrow}
\end{align*}
\]

**Table 5. SOS rules.**

\[
\begin{align*}
s_p(\bot) = F & \quad \text{RSO0} \\
s_p(a) = T & \quad \text{RSO1} \\
s_p(x + y) = s_p(x) \land s_p(y) & \quad \text{RSO2} \\
s_p(x \cdot y) = s_p(x) & \quad \text{RSO3} \\
s_p(\phi \not\setminus x) = \phi \land s_p(x) & \quad \text{RSO4} \\
s_p(x \not\setminus \phi) = s_p(x) & \quad \text{RSO5} \\
s_p(\phi \rightarrow x) = \phi \supset s_p(x) & \quad \text{RSO6}
\end{align*}
\]

**Table 6. Root signal operator.**

Based on 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 several transitions on the other side, 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 \( v: \{ P_1, \ldots, P_n \} \to \text{BOOL} \). Each such mapping naturally extends to a mapping on all formulas. We write \( \phi = \psi \) (also in the rules above) iff for all valuations \( v, v(\phi) = T \) iff \( v(\psi) = T \). Similarly, \( \phi \neq \psi \) 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:

i. if \( xRy \) then \( s \overset{\phi,a}{\to} x \) and \( x \overset{\phi,a}{\to} y \) imply \( (x \land \phi) = T \), there is a condition \( \psi \) and an expression \( x' \) such that \( v(\psi) = T \), \( x \overset{\psi,a}{\to} x' \), and \( v(x') = T \).

ii. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (y \land \phi) = T \), there is a condition \( \psi' \) and an expression \( y' \) such that \( v(\psi') = T \), \( y \overset{\psi',a}{\to} y' \), and \( v(y') = T \).

iii. if \( xRy \) and \( x \overset{\psi,a}{\to} z \) imply \( (x \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( x \overset{\psi,a}{\to} z \), and \( \psi \neq \psi' \).

iv. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (y \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( y \overset{\psi,a}{\to} z \), and \( \phi \neq \phi' \).

v. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (x \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( x \overset{\psi,a}{\to} z \), and \( \phi \neq \phi' \).

vi. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (y \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( y \overset{\psi,a}{\to} z \), and \( \phi \neq \phi' \).

vii. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (x \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( x \overset{\psi,a}{\to} z \), and \( \phi \neq \phi' \).

viii. if \( xRy \) and \( y \overset{\psi,a}{\to} z \) imply \( (y \land \phi) = T \), there are conditions \( \phi', \psi' \) with \( v(\phi') = T \), \( v(\psi') = T \), \( y \overset{\psi,a}{\to} z \), and \( \phi \neq \phi' \).

We call two expressions \( x,y \) (strongly) bisimilar, notated \( x \approx y \), if there is a (strong) bisimulation relating \( x \) and \( y \). We state the following proposition without proof.

2.7 PROPOSITION. Bisimulation is an congruence relation on process expressions.

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

2.8 THEOREM (SOUNDNESS). \( \mathcal{P}/\approx \models \text{BPAPs} \).

PROOF: By the previous proposition, it is enough to verify each axiom separately. We confine ourselves to give the bisimulation relation. Note that \( \mathcal{P}/\approx \not\models \bot = \delta \), since \( s_\bot(\bot) = F \neq T = s_\bot(\delta) \).

For axiom A1, take the relation relating left-hand and right-hand side and relating each term to itself. A2,3,4 go similarly. For A5, relate in addition all pairs of the form \( x(yz) \) to \( (xy)z \), and all pairs of the form \( (\phi \land x)y \) to \( \phi \land (xy) \). A6 goes like A1, and for A7 it suffices to relate right-hand and left-hand side. NE1,2,3 go like A7.

GC1,10,13,12 go like A1, GC2,9 like A7. GC11 also goes like A1, but note that here we use the fact that for a valuation \( v, v((\phi \land \chi) \lor (\psi \land \chi)) = T \) iff \( v(\phi \land \chi) = T \lor v(\psi \land \chi) = T \).

RSE1,2,3,4,7 go like A1, RSE5,6 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 \( x(y \land \phi) \to x(\phi \land y) \). TSE2,8, TRSE2 go like A1. For TSE3, relate all terms to themselves, and all terms of the form \( x(\phi \land y) \to x(y \land \phi) \). For TSE4, relate all terms to themselves, and all terms of the form \( x \land (y \land \phi) \to x \land (\phi \land y) \). For TRSE1, relate all terms to themselves, all terms of the form \( x(\phi \land y) \to x((\phi \land y) \land x) \), and all terms of the form \( \phi \land (\psi \land x) \to (\phi \land \psi) \land x \).

For basic terms, there is a direct relation between syntax and semantics.
2.9 LEMMA. Let $t \in B$.

i. The root signal of $t$ is $s \circ (t)$

ii. $t, a \xrightarrow{\phi} s$ iff $\phi : a \searrow s$

iii. $t, a \xrightarrow{\phi} \psi$ is a summand of $t$ for all $s$.

2.10 THEOREM (COMPLETENESS). Let $t,s$ be two closed BPAps terms. Then $x \equiv y$ implies $\text{BPAps} \vdash t = s$.

PROOF: By the basic term lemma, it is enough to prove this for basic terms. The proof can be completed using lemma 2.9.

As a corollary, we have $P/\equiv \vdash t = s \iff \text{BPAps} \vdash t = s$ for all closed $t,s$.

2.11 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.

| $\phi \Downarrow \perp = \perp$ | GSE0  |
| $\phi \Downarrow \Downarrow a = \phi \Downarrow x \Downarrow \phi$ | GSE1  |
| $\phi \Downarrow (x + y) = (\phi \Downarrow x) + (\phi \Downarrow y)$ | GSE2  |
| $\phi \Downarrow (x \cdot y) = (\phi \Downarrow x) \cdot (\phi \Downarrow y)$ | GSE3  |
| $\phi \Downarrow (\psi \cdot x) = \psi \Downarrow (\phi \Downarrow x)$ | GSE4  |
| $\phi \Downarrow (x \cdot \psi) = (\phi \Downarrow x) \cdot \psi$ | GSE5  |
| $\phi \Downarrow (\psi \Rightarrow x) = \psi \Rightarrow (\phi \Downarrow x) + \neg \psi \Rightarrow (\phi \Downarrow \delta)$ | GSE6  |

TABLE 7. Global signal emission.

| $x \Downarrow \phi, a \rightarrow (x) \neq F$ | $x \Downarrow \phi, a \rightarrow (x) \neq F,$ $x \Downarrow \chi \wedge \psi \Rightarrow F$ |
| $\psi \Downarrow x \Downarrow \phi, a \rightarrow \psi \Downarrow x'$ | $\psi \Downarrow x \Downarrow \phi, a \rightarrow$ |
| $s_p(\phi \Downarrow x) = \phi \wedge s_p(x)$ |

TABLE 8. Operational semantics of global signal emission.

With the help of the global signal emission operator, we can define a notion of invariance:

DEFINITION: $\phi$ is an invariant of $x$ if $\phi \Downarrow x = \phi \Downarrow x$. 
2.12 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 $s_p(x) = \top$ 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. It remains to be seen if this operator is useful.

Notice that the equation $s_p(x \cdot \phi) = s_p(x)$ is derivable:

$$s_p(x \cdot \phi) = s_p((x \cdot \phi) \cdot y) = s_p(x \cdot (\phi \cdot y)) = s_p(x).$$

Also $x = s_p(x) \cdot p_p(x)$ 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>
<thead>
<tr>
<th>p_p(\bot) = \bot</th>
<th>RSD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_p(a) = a</td>
<td>RSD1</td>
</tr>
<tr>
<td>p_p(x + y) = s_p(x+y) \cdot (p_p(x) + p_p(y)) + \neg s_p(x+y) \cdot \bot</td>
<td>RSD2</td>
</tr>
<tr>
<td>p_p(x \cdot y) = p_p(x) \cdot y</td>
<td>RSD3</td>
</tr>
<tr>
<td>p_p(\phi \cdot x) = (\phi \rightarrow p_p(x)) + (\neg \phi \rightarrow \bot)</td>
<td>RSD4</td>
</tr>
<tr>
<td>p_p(x \cdot \phi) = p_p(x) \cdot \phi</td>
<td>RSD5</td>
</tr>
<tr>
<td>p_p(\phi \rightarrow x) = \phi \rightarrow p_p(x)</td>
<td>RSD6</td>
</tr>
</tbody>
</table>

| TABLE 9. Root signal deletion operator. |

2.13 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.

<table>
<thead>
<tr>
<th>P \Delta \bot = \bot</th>
<th>SH0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P \Delta (\phi \cdot \delta) = (\phi[T/P] \lor \phi[F/P]) \cdot \delta</td>
<td>SH1</td>
</tr>
<tr>
<td>P \Delta (x + y) = P \Delta (s_p(x+y) \cdot x) + P \Delta (s_p(x+y) \cdot y)</td>
<td>SH2</td>
</tr>
<tr>
<td>P \Delta (\phi \cdot \psi) \rightarrow a \cdot x = (\psi[T/P] \lor \psi[F/P]) \cdot (a \cdot (P \Delta x) + (\phi \land \psi)[F/P]) \rightarrow a \cdot (P \Delta x))</td>
<td>SH3</td>
</tr>
<tr>
<td>P \Delta (\phi \cdot \psi) \rightarrow a \cdot \chi = (\phi[T/P] \lor \phi[F/P]) \cdot (a \cdot (\chi[T/P] \lor \chi[F/P]) + (\phi \land \psi)[F/P] \rightarrow a \cdot (\chi[T/P] \lor \chi[F/P]))</td>
<td>SH4</td>
</tr>
</tbody>
</table>

| TABLE 10. Signal hiding. |
J.C.M. Baeten & J.A. Bergstra

Example: $P \land (a \cdot (P \land b) + a \cdot (\neg P \land b)) = a \cdot (T \land (T \rightarrow b)) + a \cdot (T \land (T \rightarrow b)) = a \cdot b$. 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 2.3). Thus, this theory only exists in a branching time setting.

The global signal emission operator of 2.11, the root signal operator and root signal deletion operator of 2.12 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. It is not difficult to 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 (use the recipe of [BAV94]).

2.14 ITERATION.
We will not deal with full recursion in this paper. It is enough to consider linear recursion and iteration. For iteration, we use the operator $*$ (Binary Kleene Star) of [BEBP94]. We present axioms in table 12, operational semantics in table 13. We have one extra axiom for the Kleene star and terminal signal emission.

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

**Table 12. Binary Kleene Star.**
Process algebra with propositional signals

2.15 LINEAR RECURSION.

A recursion equation is linear if it is of the form:

\[ X = \delta + \sum_{i=1}^{n} \phi_{i} \rightarrow a_{i} X_{i} + \sum_{j=1}^{m} \psi_{j} \rightarrow b_{j} x_{j} \]

where \( \delta, \phi_{i}, \psi_{j}, x_{j} \in B - \{F\}, a_{i} \in A, b_{j} \in A_{\partial} \), and the \( X, X_{i} \) are recursion variables. The semantics for processes given by systems of such equations is easily given: for an equation as above we have for each summand a transition, of one the forms

\[ X_{i} \phi_{i} \rightarrow a_{i} (- X_{i} \psi_{j} \rightarrow b_{j} x_{j}) \]

Recursion equations in the following can simply be brought into this form.

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 \( \ll, \|, \ll\| \), and the axioms of tables 7 and 14.

| \( x \ll y = x \ll y + y \ll x \) | M1 |
| \( (a \ast (\phi)) \ll x = a \ast (\phi \ll x) \) | M2TS |
| \( a \ll y = a \ll (x \ll y) \) | M3 |
| \( (x + y) \ll z = x \ll z + y \ll z \) | M4 |
| \( (\phi \ast \ast x) \ll y = \phi \ast \ast (x \ll y) \) | MRS |
| \( (\phi : \rightarrow x) \ll y = \phi : \rightarrow (x \ll y) \) | MGC |

TABLE 14. Free merge.

Note: \( \bot \ll x = (F \ast \ast a) \ll x = F \ast \ast (a \ll x) = \bot \), and \( a \ll x = (a \ast \ast T) \ll x = a \ast (T \ast \ast x) \), where the last expression can be proven equal to \( a \ast x \) for all closed terms.
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 \{green, yellow, red\}.

\[
\begin{align*}
\text{TL(green)} &= (\text{green} \land \neg \text{yellow} \land \neg \text{red}) \land \neg \text{change}\cdot\text{TL(yellow)} \\
\text{TL(yellow)} &= (\neg \text{green} \land \text{yellow} \land \neg \text{red}) \land \neg \text{change}\cdot\text{TL(red)} \\
\text{TL(red)} &= (\neg \text{green} \land \neg \text{yellow} \land \text{red}) \land \neg \text{change}\cdot\text{TL(green)}.
\end{align*}
\]

Now we describe a careful car driver.

\[
\text{CD} = \text{approach}\cdot((\neg \text{green} \Rightarrow \text{stop})\cdot(\text{green} \Rightarrow \text{start})\cdot\text{drive} + (\text{green} \Rightarrow \text{drive})).
\]

Expression $\text{TL}(x) \parallel \text{CD}$ now describes a correct interaction between light and driver.

3.3 ACPps.

The theory ACPps, Algebra of Communicating Processes with Propositional Signals, extends PAp\$ with operators $\parallel, \delta, \sigma, \gamma$, and replaces the axioms of table 14 by the axioms of the root signal operator and the axioms in table 15 below. We assume given a partial commutative and associative binary function on $A$, the communication function $\gamma$. 

We provide the semantics of ACPps in table 16. The semantics of PAp's can be extracted, by omitting all parts referring to the communication merge operator.
TABLE 16. Semantics of ACPps.

---

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, s(x \parallel y) \neq F}{x \parallel y \, \phi, a \rightarrow ...} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, \psi \land s(y) \neq F}{x \parallel y \, \phi, a \rightarrow ... y} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, s(x \parallel y) \neq F, \, a \parallel b=c}{x \parallel y \, \phi \land \psi \land c \rightarrow ...} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, \chi \land s(y) \neq F, \, a \parallel b=c}{x \parallel y \, \phi \land \psi \land c \rightarrow ...} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, s(x \parallel y) \neq F, \, a \parallel b=c}{x \parallel y \, \phi \land \psi \land c \rightarrow ... y'} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, \chi \land s(y) \neq F, \, a \parallel b=c}{x \parallel y \, \phi \land \psi \land c \rightarrow ... y'} \\
\end{array}
\]

\[
\begin{array}{l}
\frac{x \, \phi, a \rightarrow \, (x \parallel y) \neq F, \, s(x \parallel y) \neq F, \, a \parallel b=c}{x \parallel y \, \phi \land \psi \land c \rightarrow ... y'} \\
\end{array}
\]

---

4. STATE OPERATOR.

In this section, we extend any of the theories BPAps, PAps, ACPps with the state operator of [BAB88]. 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 [BAB88] is given by a domain $S$ and functions $\text{act}: A \times S \rightarrow A \cup \{\emptyset\}$ and $\text{eff}: A \times S \rightarrow S$. The expression $\lambda_\Psi(x)$ 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 $\text{sig}: S \rightarrow \mathbb{B}$ which determines for each state the signal that is emitted by that state. The absence of signals is modeled by taking $\text{sig}(s) = T$ of course.
Now the eight equations for the state operator are as shown in table 17, the operational semantics is given in table 18.

| SOS0 | λₜ(⊥) = ⊥ |
| SOS1 | λₜ(δ) = sig(s) ⊙ δ |
| SOS2 | λₜ(a) = sig(s) ⊙ act(a, s) ⊙ sig(eff(a, s)) |
| SOS3 | λₜ(x + y) = λₜ(x) + λₜ(y) |
| SOS4 | λₜ(∂) = sig(s) |
| SOS5 | λₜ(a˘x) = sig(s) ⊙ act(a, s)˘¬ eff(a, s)(x) |
| SOS6 | λₜ(ƒ \ x) = sig(s) ⊙ ƒ \ λₜ(x) |
| SOS7 | λₜ(φ → x) = sig(s) \ φ → λₜ(x) |

TABLE 17. State operator generating signals.

| x φ,a → (x)\F, sig(eff(a,s)) \(x')\F, act(a,s)=b\δ |
| λₜ(x) φ,b → (x') |

| x φ,a → (x)\F, sig(eff(a,s)) \(x')\F, act(a,s)=b\δ |
| λₜ(x) φ,b → |

| sp(λₜ(x)) = sp(x) \ sig(s) |

TABLE 18. Operational semantics.

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 two examples.

4.2 EXAMPLE.
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 ε and adding an element d to the list x results in x.d. The propositional constants are as follows:

on_top(d) for d ∈ D,
empty.

We will assume that these signals are exclusive, i.e. we will assume that the following formula always holds: Φ = (empty ⊆ \∧ deD on_top(d)) \∧ \∧ deD (on_top(d) ⊆ \∧ ε≠d on_top(ε)).
D* will be the state space for a process that represents a stack over D. The signal function sig is defined by sig(ε) = empty, sig(od) = top(d). The atomic actions are:

push_int(d), push(d) for d ∈ D (the suffix int denotes an intended action),
The functions \( \text{act} \) and \( \text{eff} \) are given by:

\[
\text{act}(\text{push}_\text{int}(d), \sigma) = \text{push}(d) \quad \text{(the act function transforms an intended action into an actual action)},
\]

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

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

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

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

(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.

\[
\text{stack}(D) = \lambda_e((\sum_{d \in D} \text{push}_\text{int}(d) + \text{pop}_\text{int})^* \delta).
\]

4.3 EXAMPLE.

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.

\[
\text{SYSTEM} = \lambda_{\langle \varepsilon, \varepsilon \rangle} (A \parallel B \parallel C).
\]

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.

\[
\text{sig}(\langle \alpha, \beta \rangle) =
\begin{align*}
\text{open}_A \land \text{open}_B & \quad \text{if length}(\alpha) < k \text{ and length}(\beta) < k, \\
\neg \text{open}_A \land \text{open}_B & \quad \text{if length}(\alpha) = k \text{ and length}(\beta) < k, \\
\text{open}_A \land \neg \text{open}_B & \quad \text{if length}(\alpha) < k \text{ and length}(\beta) = k, \\
\neg \text{open}_A \land \neg \text{open}_B & \quad \text{if length}(\alpha) = k \text{ and length}(\beta) = k.
\end{align*}
\]

The processes \( A, B, C \) are defined by:

\[
A = (\text{open}_A \rightarrow \sum_{d \in D} \text{read}_A(d))^* \delta
\]

\[
B = (\text{open}_B \rightarrow \sum_{d \in D} \text{read}_B(d))^* \delta
\]

\[
C = (\neg \text{open}_A \land \neg \text{open}_B \rightarrow \text{comp})^* \delta
\]

The next step is to explain the effect function:

\[
\text{eff}(\text{read}_A(d), \langle \alpha, \beta \rangle) = \langle \alpha d, \beta \rangle
\]

\[
\text{eff}(\text{read}_B(d), \langle \alpha, \beta \rangle) = \langle \alpha, \beta d \rangle
\]

\[
\text{eff}(\text{comp}, \langle \alpha, \beta \rangle) = \langle \varepsilon, \varepsilon \rangle
\]
Finally the action function must be specified:
\[
\text{act}(\text{comp}, (\alpha, \beta)) = \begin{cases} 
\text{write(true)} & \text{if } \alpha = \beta \\
\text{write(false)} & \text{otherwise}, 
\end{cases}
\]
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 [VER90].

5. ABSTRACTION.
We provide axioms for silent step and abstraction in the setting of branching bisimulation of [GLW89].

5.1 ACP\textsuperscript{TPS}.
The theory ACP\textsuperscript{TPS} 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 19 below.

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
$a(\phi \quad \tau) = a \quad \phi$ & B1S \\
$x(\sum p(y) \quad x (\tau (y + z) + z)) = x(y + z)$ & B2S \\
$\tau_I(a) = a$ & if $a \notin I$ \\
$\tau_I(a) = \tau$ & if $a \in I$ \\
$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$ & T13 \\
$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$ & T14 \\
$\tau_I(\phi \quad x) = \phi \quad \tau_I(x)$ & TIRS \\
$\tau_I(x \quad \phi) = \tau_I(x) \quad \phi$ & TITS \\
$\tau_I(\phi \rightarrow x) = \phi \rightarrow \tau_I(x)$ & TIGC \\
\hline
\end{tabular}
\caption{Silent step and abstraction.}
\end{table}

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 20.
With this comes a new definition of bisimulation. We consider the set $\mathbb{P}B$ of closed terms and propositional formulas. In the following, $x,y,x',y'$ range over terms, $\phi,\psi,...$ range over propositions and $s,t,...$ over $\mathbb{P}B$.

A relation $R$ on $\mathbb{P}B$ is a branching bisimulation when the following holds:

i. if $xRy$ then $s \triangleright (x) = s \triangleright (y)$, if $xR\phi$ then $s \triangleright (x) = \phi$, if $\phi Rx$ then $s \triangleright (x) = s \triangleright (x)$ and if $\phiR\psi$ then $\phi = \psi$

ii. if $xRs$ and $x \phi \rightarrow t$, $t \phi \rightarrow s \phi$, and $sRt$, then:
   a. $s \phi \rightarrow t$ and $t \phi \rightarrow s \phi$
   b. for all valuations $v$ such that $v(s \triangleright (x)) = T$, there are propositions $\psi_1$, $\psi_2$, ..., $\psi_n$ ($n \geq 0$) and $s'$, $y_1$, ..., $y_n$ such that $s \triangleright (x) \triangleright (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_n) = T$ for all $i$, $v(\psi_i) = T$, $s \phi \rightarrow v_1$, $s \phi \rightarrow v_2$, ..., $s \phi \rightarrow v_n$, and $tR\psi_i$, $tR\psi_1$, $tR\psi_2$, ..., $tR\psi_n$, $tR\psi_{n+1}$, $tR\psi_{n+2}$, ...

iii. vice versa: if $sRx$ and $x \phi \rightarrow t$, $t \phi \rightarrow s \phi$, then:
   a. $s \phi \rightarrow t$ and $t \phi \rightarrow s \phi$
   b. for all valuations $v$ such that $v(s \triangleright (x)) = T$, there are propositions $\psi_1$, $\psi_2$, ..., $\psi_n$ ($n \geq 0$) and $s'$, $y_1$, ..., $y_n$ such that $s \triangleright (x) \triangleright (\psi_1 \lor \psi_2 \lor \cdots \lor \psi_n) = T$ for all $i$, $v(\psi_i) = T$, $s \phi \rightarrow v_1$, $s \phi \rightarrow v_2$, ..., $s \phi \rightarrow v_n$, and $tR\psi_i$, $tR\psi_1$, $tR\psi_2$, ..., $tR\psi_n$, $tR\psi_{n+1}$, $tR\psi_{n+2}$, ...

We say a branching bisimulation $R$ satisfies the root condition for $x$ and $y$ if $xRy$ and in addition:

iv. if $x \phi \rightarrow (x \land \phi) = T$, there is a proposition $\psi$ and a term or proposition $t$ such that $v(\psi) = T$, $x \phi \rightarrow v_1$, $x \phi \rightarrow v_2$, ..., $x \phi \rightarrow v_n$, and $y \phi \rightarrow (x \land \phi) = T$, there is a proposition $\psi$ and a term or proposition $s$ such that $v(\psi) = T$, $x \phi \rightarrow v_1$, $x \phi \rightarrow v_2$, ..., $x \phi \rightarrow v_n$, and $y \phi \rightarrow (x \land \phi) = T$, there is a proposition $\psi$ and a term or proposition $t$ such that $v(\psi) = T$, $x \phi \rightarrow v_1$, $x \phi \rightarrow v_2$, ..., $x \phi \rightarrow v_n$.

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

5.3 THEOREM. For all closed ACP$^\phi$ terms $t,s$ we have

\[ \text{ACP}^\phi \vdash t = s \iff \mathbb{P}B/\equiv_{rb} \vdash t = s, \]

i.e. ACP$^\phi$ is a sound and complete axiomatisation of the model $\mathbb{P}B/\equiv_{rb}$.

PROOF: Similar to the standard proof of the completeness of ACP$^\phi$ (see [BAW90]).

---

**Table 20. Semantics of ACP$^\phi$**

<table>
<thead>
<tr>
<th>$\phi, a \rightarrow$</th>
<th>$\phi, a \rightarrow$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1(x)$</td>
<td>$\tau_1(x)$</td>
</tr>
<tr>
<td>$\phi, a \rightarrow$</td>
<td>$\phi, a \rightarrow$</td>
</tr>
<tr>
<td>$\phi, \tau \rightarrow$</td>
<td>$\phi, \tau \rightarrow$</td>
</tr>
<tr>
<td>$s \triangleright (\tau_1(x)) = s \triangleright (x)$</td>
<td>$s \triangleright (\tau_1(x)) = s \triangleright (x)$</td>
</tr>
</tbody>
</table>
6. EXAMPLES.
In this section we discuss a number of examples of the use of signals and inspection.

6.1 QUEUE.
A specification of a (FIFO) queue can be given as follows. We have a given finite data set \( D \), and the following specifications have variables indexed by sequences over \( D \).

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

\[
Q_{od} = \neg \text{empty} \wedge s(d) \cdot Q_o + \sum_{e \in D} r(e) \cdot Q_{ead}.
\]

6.2 BAG.
The bag is indexed by multi-sets.

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

\[
V \neq \emptyset \Rightarrow B_V = \neg \text{empty} \wedge \sum_{d \in V} s(d) \cdot B_{V \setminus \{d\}} + \sum_{d \in D} r(d) \cdot B_{V \cup \{d\}}.
\]

6.3 STACK.
We use conventions as above. We give a number of alternatives. First a stack without signals.

\[
S_1^e = \sum_{d \in D} \text{push}(d) \cdot S_1^d
\]

\[
S_{1d} = \text{pop} \cdot S_o + \text{top} \cdot S_{o} + \sum_{e \in D} \text{push}(e) \cdot S_{ode}.
\]

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

\[
S_2^e = \sum_{d \in D} \text{push}(d) \cdot S_2^d
\]

\[
S_{2d} = \text{pop} \cdot S_o + \text{top} \cdot (\text{show}(d) \wedge S_{2d}) + \sum_{e \in D} \text{push}(e) \cdot S_{2de}.
\]

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

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

\[
S_{3d} = \text{pop} \cdot S_o + \text{top} \cdot (\text{show}(d) \wedge S_{3d}) + \neg \text{empty} \wedge \sum_{e \in D} \text{push}(e) \cdot S_{3de}.
\]

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

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

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

\[
S_{4d} = \text{pop} \cdot S_o + \text{top} \cdot (\text{show}(d) \wedge S_{4d}) + \neg \text{empty} \wedge \sum_{e \in D} \text{push}(e) \cdot S_{4de}.
\]
The fifth stack keeps functioning, when a pop or top is executed on an empty stack.

\[
S_5^e = \text{empty} \lor \sum_{d \in D} \text{push}(d) \cdot S_5^d + \text{top}(\text{show}(\perp)) \lor S_5^e \\
S_5^{\alpha_d} = \text{pop} \cdot S_5^\alpha + \text{top}(\text{show}(d)) \lor S_5^{\alpha_d} + \text{empty} \lor \sum_{e \in D} \text{push}(e) \cdot S_5^{\alpha_d}. 
\]

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

\[
S_6^e = \text{empty} \lor \sum_{d \in D} \text{push}(d) \cdot S_6^d + (\text{top} + \text{pop})(\text{error}(\text{top} + \text{pop} + \sum_{d \in D} \text{push}(d)) \cdot \delta) \\
S_6^{\alpha_d} = \text{pop} \cdot S_6^\alpha + \text{top}(\text{show}(d)) \lor S_6^{\alpha_d} + \text{empty} \lor \sum_{e \in D} \text{push}(e) \cdot S_6^{\alpha_d}. 
\]

6.4 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. 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.

\[
B_{ij} = \text{show}_j(\emptyset) \lor \sum_{d \in D} \text{read}_i(d) \cdot B_{ij} \\
B_{ij}^d = \text{show}_j(d) \lor \text{send}_j(d) \cdot B_{ij}. 
\]

\[X = \varnothing H (B_{12} \land B_{23}) \]

where \(\text{send}_2(d) \lor \text{read}_2(d) = \text{comm}_2(d)\) (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:

\[
X = (\text{show}_2(\emptyset) \land \text{show}_3(\emptyset)) \lor \sum_{d \in D} \text{read}_1(d) \cdot X^d_1 \\
X^d_1 = (\text{show}_2(d) \land \text{show}_3(\emptyset)) \lor \text{comm}_2(d) \cdot X^d_2 \\
X^d_2 = (\text{show}_2(\emptyset) \land \text{show}_3(d)) \lor \text{send}_3(d) \cdot X + \sum_{e \in D} \text{read}_1(d) \cdot X^{de}_3 \\
X^{de}_3 = (\text{show}_2(e) \land \text{show}_3(d)) \lor \text{send}_3(d) \cdot X^d_3 
\]

Hiding all signals gives back the usual specification of two coupled one-element buffers (as in [BAW90], page 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

\[
\Phi_1 = (\text{show}_1(\emptyset) \Rightarrow \bigwedge_{d \in D} \neg \text{show}_1(d)) \land \\
\bigwedge_{d \in D} (\text{show}_1(d) \Rightarrow \neg \text{show}_1(\emptyset) \land \bigwedge_{e \neq d} \neg \text{show}_1(e)) \\
\Phi_2 = \text{show}_1(\emptyset) \lor \sum_{d \in D} \text{show}_1(d) \Rightarrow \text{read}_1 \cdot C_{ij}^d \\
\Phi_3 = \text{show}_j(\emptyset) \lor \text{send}_j \cdot C_{ij}. 
\]

\[Y = (\Phi_1 \land \Phi_2 \land \Phi_3) \land \varnothing H(C_{12} \land C_{23}) \]

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

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

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

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

$Y^d_2 = (\text{show}_2(\emptyset) \land \text{show}_3(d)) \rightarrow \text{send}_3 \cdot (Y + \sum_{e \in D} \text{show}_1(e) : \rightarrow \text{read}_1 \cdot Y^d_3$

$Y^d_3 = (\text{show}_2(e) \land \text{show}_3(d)) \rightarrow \text{send}_3 \cdot Y^d_1$

Let us now abstract from actions and signals at port 2. Put $I = \{\text{comm}_2\}$ and $\text{show}_2 = \{\text{show}_2(d) : d \in D\} \cup \{\text{show}_2(\emptyset)\}$, and derive the following specification for $Z = \text{show}_2 \Delta \tau_1(Y)$:

$Z = \text{show}_2 \Delta \tau_1(Y) = \text{show}_3(\emptyset) \sum_{d \in D} \text{show}_1(d) : \rightarrow \text{read}_1 \cdot Z^d_1$

$Z^d_1 = \text{show}_2 \Delta \tau_1(Y^d_1) = \text{show}_3(\emptyset) \rightarrow Z^d_2$

$Z^d_2 = \text{show}_2 \Delta \tau_1(Y^d_2) = \text{show}_3(d) \rightarrow \text{send}_3 \cdot Z + \sum_{e \in D} \text{show}_1(e) : \rightarrow \text{read}_1 \cdot Z^d_3$

$Z^d_3 = \text{show}_2 \Delta \tau_1(Y^d_3) = \text{show}_3(d) \rightarrow \text{send}_3 \cdot Z^d_1$

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:

$E^i = (\text{show}(d) \land \neg \text{flag}_i) \sum_{d \in D} (\text{show}(d) \land \neg \text{flag}_j) : \rightarrow \text{read}_1 \cdot E^d_1$

$E^d_1 = (\text{show}(d) \land \text{flag}_i) \sum_{d \in D} (\text{show}(d) \land \neg \text{flag}_j) : \rightarrow \text{s}_j \cdot C^i_d$

$W = (\Phi_1 \land \Phi_2 \land \Phi_3) \rightarrow E_1 \parallel E_2$, where this is the free merge, i.e. this is a specification in PAs.

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:

$F^i = (\text{read}_1 \land \text{show}_1(\emptyset) \land \neg \text{flag}_j) \sum_{d \in D} (\text{flag}_i \land \text{show}(d)) : \rightarrow \text{read}_1 \cdot F^d_3$

$F^d_3 = (-\text{ready}_1 \land \text{show}(d) \land \neg \text{flag}_j) \sum_{d \in D} (-\text{ready}_1 \land \neg \text{flag}_1) : \rightarrow \text{send}_j \cdot G^i_d$

$G^i_d = (-\text{ready}_1 \land \text{show}(d) \land \text{flag}_j) \sum_{d \in D} (-\text{ready}_1 \land \neg \text{reset}_j) : \rightarrow \text{reset}_j \cdot G^i_d$

$V = (\Phi_1 \land \Phi_2 \land \Phi_3) \rightarrow F_1 \parallel F_2$, where this is the free merge, i.e. this is a specification in PAs.

We can derive the following specification for $V$:

$V = (\text{read}_1 \land \text{read}_2 \land \text{show}_2(\emptyset) \land \text{show}_3(\emptyset) \land \neg \text{flag}_2 \land \neg \text{flag}_3) \sum_{d \in D} (\text{flag}_1 \land \text{show}(d)) : \rightarrow \text{read}_1 \cdot V_1,d$

$V_1,d = (-\text{ready}_1 \land \text{read}_2 \land \text{show}_2(d) \land \text{show}_3(\emptyset) \land \neg \text{flag}_2 \land \neg \text{flag}_3) \rightarrow \neg \text{flag}_1 \rightarrow \text{send}_2 \cdot V_2,d$

$V_2,d = (-\text{ready}_1 \land \text{read}_2 \land \text{show}_2(d) \land \text{show}_3(\emptyset) \land \text{flag}_2 \land \neg \text{flag}_3) \rightarrow \text{read}_2 \cdot V_3,d$

$V_3,d = (-\text{ready}_1 \land \neg \text{read}_2 \land \text{show}_2(d) \land \text{show}_3(\emptyset) \land \text{flag}_2 \land \neg \text{flag}_3) \rightarrow \text{reset}_2 \cdot V_4,d$

$V_4,d = (\text{ready}_1 \land \neg \text{read}_2 \land \text{show}_2(\emptyset) \land \text{show}_3(d) \land \neg \text{flag}_2 \land \neg \text{flag}_3)$
ready₃ := send₃ \cdot V₅,d + \sum_{e \in D} (\text{flag}₁ \land \text{show}(e)) \rightarrow \text{read}₁ \cdot V₆,de

V₅,d = (\text{ready}₁ \land \neg \text{ready}₂ \land \text{show}₂(∅) \land \text{show}₃(d) \land \neg \text{flag}₂ \land \text{flag}₃) \quad \neg \text{ready}₃ \rightarrow \text{reset}₃ \cdot V + \sum_{e \in D} (\text{flag}₁ \land \text{show}(e)) \rightarrow \text{read}₁ \cdot V₇,de

V₆,de = (\neg \text{ready}₁ \land \neg \text{ready}₂ \land \text{show}₃(e) \land \text{show}₃(d) \land \neg \text{flag}₂ \land \neg \text{flag}₃) \quad \neg \text{ready}₃ \rightarrow \text{send}₃ \cdot V₇,de

V₇,de = (\neg \text{ready}₁ \land \neg \text{ready}₂ \land \text{show}₃(e) \land \text{show}₃(d) \land \neg \text{flag}₂ \land \text{flag}₃) \quad \neg \text{ready}₃ \rightarrow \text{reset}₃ \cdot V₁,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

\( P = \{\text{show}₂(d) : d \in D\} \cup \{\text{show}₂(∅), \text{ready}₁, \text{ready}₂, \text{ready}₃, \text{flag}₁, \text{flag}₂, \text{flag}₃\} \)

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

\[
\begin{align*}
P \Delta V &= \text{show}₃(∅) \quad \sum_{d \in D} \text{show}(d) \rightarrow \text{read}₁ \cdot (P \Delta V₁,d) \\
P \Delta V₁,d &= \text{show}₃(∅) \quad \text{send}₂ \cdot (P \Delta V₂,d) \\
P \Delta V₂,d &= \text{show}₃(∅) \quad \text{read}₂ \cdot (P \Delta V₃,d) \\
P \Delta V₃,d &= \text{show}₃(d) \quad \text{reset}₂ \cdot (P \Delta V₄,d) \\
P \Delta V₄,d &= \text{show}₃(d) \quad \text{send}₃ \cdot (P \Delta V₅,d) + \sum_{e \in D} \text{show}(e) \rightarrow \text{read}₁ \cdot (P \Delta V₆,de) \\
P \Delta V₅,d &= \text{show}₃(d) \quad \text{reset}₃ \cdot (P \Delta V₇,de) + \sum_{e \in D} \text{show}(e) \rightarrow \text{read}₁ \cdot (P \Delta V₇,de) \\
P \Delta V₆,de &= \text{show}₃(d) \quad \text{send}₃ \cdot (P \Delta V₇,de) \\
P \Delta V₇,de &= \text{show}₃(d) \quad \text{reset}₃ \cdot (P \Delta V₁,e) \\
\end{align*}
\]

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:

\[
\begin{align*}
\tau_I(P \Delta V) &= \text{show}₃(∅) \quad \sum_{d \in D} \text{show}(d) \rightarrow \text{read}₁ \cdot \tau_I(P \Delta V₁,d) \\
\tau_I(P \Delta V₁,d) &= \text{show}₃(∅) \quad \tau \cdot \tau_I(P \Delta V₂,d) \\
\tau_I(P \Delta V₂,d) &= \text{show}₃(∅) \quad \tau \cdot \tau_I(P \Delta V₃,d) \\
\tau_I(P \Delta V₃,d) &= \text{show}₃(d) \quad \tau \cdot \tau_I(P \Delta V₄,d) \\
\tau_I(P \Delta V₄,d) &= \text{show}₃(d) \quad \text{send}₃ \cdot \tau_I(P \Delta V₅,d) + \sum_{e \in D} \text{show}(e) \rightarrow \text{read}₁ \cdot \tau_I(P \Delta V₆,de) \\
\tau_I(P \Delta V₅,d) &= \text{show}₃(d) \quad \tau \cdot \tau_I(P \Delta V) + \sum_{e \in D} \text{show}(e) \rightarrow \text{read}₁ \cdot \tau_I(P \Delta V₇,de) \\
\tau_I(P \Delta V₆,de) &= \text{show}₃(d) \quad \text{send}₃ \cdot \tau_I(P \Delta V₇,de) \\
\tau_I(P \Delta V₇,de) &= \text{show}₃(d) \quad \tau \cdot \tau_I(P \Delta V₁,e) \\
\end{align*}
\]

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

\[
\begin{align*}
\tau_I(P \Delta V) &= \text{show}₃(∅) \quad \sum_{d \in D} \text{show}(d) \rightarrow \text{read}₁ \cdot \tau_I(P \Delta V₂,d) \\
\tau_I(P \Delta V₂,d) &= \text{show}₃(∅) \quad \tau \cdot \tau_I(P \Delta V₄,d) \\
\tau_I(P \Delta V₄,d) &= \text{show}₃(d) \quad \text{send}₃ \cdot \tau_I(P \Delta V) + \sum_{e \in D} \text{show}(e) \rightarrow \text{read}₁ \cdot \tau_I(P \Delta V₆,de) \\
\tau_I(P \Delta V₆,de) &= \text{show}₃(d) \quad \text{send}₃ \cdot \tau_I(P \Delta V₁,e) \\
\end{align*}
\]

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

REFERENCES.