Eindhoven University of Technology

Department of Mathematics and Computing Science

Process Algebra with Feedback

by

J.C.M. Baeten, J.A. Bergstra and
Gh. Ştefănescu

Computing Science Note 94/30
Eindhoven, July 1994
This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

All rights reserved
editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.
We consider process graphs over a set of pins, i.e. with multiple entries and exits. On process graphs modulo bisimulation, we can define all standard process algebra operators plus the feedback operator from flowchart theory. We provide a complete axiomatisation for finite processes. Considering the one-point pin structure, we get back standard process algebra.

1980 Mathematics Subject Classification (1985 revision): 68Q55, 68Q10, 68Q45.
Key words & Phrases: process algebra, feedback, pin.

Note: The research of the first two authors was supported in part by ESPRIT basic research action 7166, CONCUR2.

1. INTRODUCTION.

Semantics of process theory is often given in terms of graphs. The process graphs considered usually have exactly one entry and exactly one exit. In [BES94], this was adapted to allow for multiple entries and multiple exits. The resulting model was used to model key constructs of ACP and of the algebra of flownominals [STE90], especially the feedback construct. Feedback is a looping or iteration construct used in e.g. flowcharts. Another iteration construct is Kleene star, that was extensively studied in the setting of ACP in [BEBP94]. Here, we generalize and extend the results of [BES94].

We consider process graphs that have interior states and so-called pins, connections to the environment (a pin is an external connection of a chip, the name is used for the external connections of a Petri net in [ExS92]). A pin can be an entry, an exit, or both. We obtain a full theory of ACP with feedback operator, which generalises ACP in the sense that it weakens the axioms (e.g., in general, parallel composition is neither commutative nor associative). But if we add structure to the set of pins, we obtain an algebra of which the original ACP is a subalgebra of a reduced model.
In the paper, we first formulate a graph model (graphs that can have several pins) for the full syntax of ACP. Pin names are used to distinguish different entries or exits. Then, we divide out bisimulation, so that we obtain a bisimulation model. For this bisimulation model, we give a complete axiomatisation of finite processes, and we analyse the algebra obtained. Then, we illustrate the expressive power of the general formalism, and sketch the extension of the theory with silent steps and abstraction.

More about feedback and flowchart theories can be found in [BAWM94], [BAR92], [BLE93], [CAS90, 92], [MIL94], [STE87a, 87b], [STA92].

2. PROCESS GRAPHS OVER A SET OF PINS.

We introduce process graphs over a set of edge labels and a set of pin names. Edge labels are, as usual, taken from a finite set of primitive actions $\mathcal{A}$. This set $\mathcal{A}$ is a parameter of the theory. Pin names are taken from a set $\mathcal{V}$, also a parameter of the theory. In order to be able to define parallel composition, we need a pairing construct on $\mathcal{V}$. Therefore, $\mathcal{V}$ must be closed under pairing, i.e. there is a coding $\Psi: \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$. In the general theory, we assume nothing further about the set $\mathcal{V}$. Later on, we consider special cases, by taking specific instances of $\mathcal{V}$. We call a set $\mathcal{V}$ that is closed under pairing and maybe has additional structure a pin structure.

3.1 DEFINITION. We introduce a set of process graphs over $\mathcal{A}$ and $\mathcal{V}$, $\mathcal{G}(\mathcal{A},\mathcal{V})$, as follows. A process graph $\mathcal{g}$ is a quadruple $(S, \rightarrow, I, O)$ where

- $S$ is the set of (interior) states. We abstract from the names of the states, i.e. we consider process graphs modulo isomorphism of states. Always, states are disjoint from pins.
- $I \subseteq \mathcal{V}$ is the set of input or entry pins
- $O \subseteq \mathcal{V}$, the set of output or exit pins
- $\rightarrow \subseteq (I \cup S) \times \mathcal{A} \times (O \cup S)$ is the transition relation

We write $s \xrightarrow{a} t$ for $(s,a,t) \in \rightarrow$. We refer to the four components of a process graph $\mathcal{g}$ by $S(\mathcal{g})$, $\rightarrow(\mathcal{g})$, $I(\mathcal{g})$ and $O(\mathcal{g})$, respectively. $I(\mathcal{g}) \cup O(\mathcal{g})$ is the set of pins of $\mathcal{g}$.

3.2 DEFINITION. We define several operators on process graphs (modulo isomorphism of states). We postpone the treatment of parallel composition to a later section. Here, we define atomic actions, alternative composition, sequential composition and feedback, together with some auxiliary operators.

First, we consider the constants.

1. Atomic actions. For each $a \in \mathcal{A}$ and $p,q \in \mathcal{V}$ we have a constant $p \xrightarrow{a} q$. In [BES94], we find the notation $a_p^q$ for $p \xrightarrow{a} q$. The notation $p \xrightarrow{a} q$ is already used in [BER89]. The interpretation of this process is the graph with no states, one edge, $p \xrightarrow{a} q$, and $I = \{p\}$, $O = \{q\}$. We picture this process on the left hand side of fig. 1. We indicate an entry pin by a small unlabeled incoming arrow, an exit pin by a small outgoing unlabeled arrow. Note that it is possible that $p=q$, but nevertheless, also in this case the intuition is that action $a$ can only be executed once.
2. Empty graph. We have a constant $\emptyset$ that stands for the empty graph, i.e. $\emptyset = (\emptyset, \emptyset, \emptyset, \emptyset)$. Abusing notation, we usually write $\emptyset$ instead of $\emptyset$. The notation $\emptyset$ was also used in [BES94]. This process behaves as the deadlocked process $\delta$ with respect to alternative and sequential composition, but not with respect to parallel composition.

Next, we consider four operators used to manipulate pins.

3. Entry operator. Let a process graph $g$ be given, and let $p \in V$. Now $p \rightarrow g$ has states $S(g)$, each edge $s \rightarrow t$ in $g$ with $s \in I(g)$ is replaced by an edge $p \rightarrow t$, and furthermore has as only entry $p$ and has the same exits as $g$. As an example, we depict graph $p \rightarrow \emptyset$ on the right hand side of fig. 1. This process behaves like ACP's $\delta$ with respect to sequential and parallel composition, but not, in general, with respect to alternative composition.

4. Exit operator. Let a process graph $g$ be given, and let $p \in V$. Now $g \rightarrow p$ has states $S(g)$, each edge $s \rightarrow t$ in $g$ with $t \in O(g)$ is replaced by an edge $s \rightarrow p$, and furthermore has the same entries as $g$ and has as only exit $p$.

5. Initialisation operator. Let a process graph $g$ be given, and let $p \in V$. $p \gg g$ has the same edges and exits as $g$. The set of entries becomes $I(g) \cap \{p\}$. States are all states of $g$, plus in addition other entries than $p$ (if they exist).

6. Exiting operator. Let a process graph $g$ be given, and let $p \in V$. $g \gg p$ has the same edges and entries as $g$. The set of exits becomes $O(g) \cap \{p\}$. States are all states of $g$, plus in addition other exits than $p$ (if they exist).

Now, we consider alternative and sequential composition.

7. Alternative composition. Let process graphs $g, h$ be given. Assume that the set of states of $g$ is disjoint from the set of states of $h$ (since we consider process graphs modulo state isomorphism, we can always ensure that this is the case). $g + h$ is obtained by taking the union of the states, the edges, the entries and the exits. As an example, we show $(p \rightarrow a \rightarrow p) + (q \rightarrow b \rightarrow q) + (p \rightarrow c \rightarrow q)$ in fig. 2. We picture both $p$ and $q$ twice, in order to emphasise the different roles of entries and exits.
8. **Sequential composition.** Let process graphs \( g, h \) be given. Again assume that the set of states of \( g \) is disjoint from the set of states of \( h \). We need to match exits of \( g \) with entries of \( h \), but we need to distinguish these from entries of \( g \) and exits of \( h \). In order to achieve this we take for each \( p \in O(g) \cup I(h) \) a fresh state \( p^* \) (i.e. a name not appearing as the name of any state of \( g \) or \( h \)). The set of interior states of \( g \cdot h \) is \( S(g) \cup S(h) \cup \{ p^* : p \in O(g) \cup I(h) \} \), the set of edges is obtained from the union of the edge sets of \( g \) and \( h \) by replacing each edge in \( g \) of the form \( s \xrightarrow{a} p \) with \( p \in O(h) \) by an edge \( s \xrightarrow{a} p^* \) and replacing each edge in \( h \) of the form \( p \xrightarrow{a} s \) with \( p \in I(h) \) by an edge \( p^* \xrightarrow{a} s \). The set of entries is \( I(g) \), the set of exits \( O(h) \). As an example, we show \( ((p \rightarrow a \rightarrow p) + (p \rightarrow b \rightarrow q)) \cdot (p \rightarrow c \rightarrow q) \) in fig. 3. Interior states are pictured as circles, their names are left out, since they do not matter (we consider graph isomorphism classes). Note that for all \( g \) and all \( p \in V \), \( g \cdot \emptyset = g \cdot \emptyset \), and this graph has no exits.

![Figure 3. Sequential composition.](image)

Finally, we consider feedback. We also define two auxiliary operators, used in the axiomatisation of feedback.

9. **Feedback.** Let process graph \( g \) be given, and let \( p, q \in V \). The set of states of \( g \updownarrow^q_p \) is \( S(g) \cup \{ r \} \), for \( r \) a fresh name, the set of edges is obtained from \( \rightarrow(g) \) by replacing each edge \( q \xrightarrow{a} t \) in \( g \) by an edge \( r \xrightarrow{a} t \), and replacing each edge \( v \xrightarrow{a} p \) by an edge \( v \xrightarrow{a} r \). Further, the entry set is \( I(g) - \{ q \} \), the exit set is \( O(g) - \{ p \} \). As an example, we show \( ((p \rightarrow a \rightarrow s) + (s \rightarrow b \rightarrow s) + (s \rightarrow c \rightarrow q)) \updownarrow^s_q \) in fig. 4.

![Figure 4. Feedback.](image)

10. **Pre-feedback.** Process graph \( g \uparrow^q_p \) is just like \( g \updownarrow^q_p \), except that we keep all edges \( q \xrightarrow{a} t \), and the entry set is \( I(g) \).

11. **Linking composition.** Let process graphs \( g, h \) be given, and let \( p, q \in V \) be given. We can assume that the state set of \( g \) is disjoint from states of \( h \). \( g \uparrow^q_p h \) has as interior state set \( S(g) \cup S(h) \cup \{ r \} \), for \( r \) a fresh name. The set of edges is obtained by replacing each edge \( q \xrightarrow{a} t \) in \( h \) by an edge \( r \xrightarrow{a} t \), and
replacing each edge $\bar{v} \rightarrow p$ in $g$ by an edge $\bar{v} \rightarrow r$. Furthermore, all edges $s \bar{a} \rightarrow t$ in $h$ with $s \in I(h)$, $s \neq q$ are omitted. The graph has as entries $I(g)$ and has as exits $(O(g) - \{p\}) \cup O(h)$.

3. BISIMULATION.
We look at the familiar notion of bisimulation in the present setting. To this end, consider the following definition.

3.1 DEFINITION. We define the familiar notion of bisimulation on process graphs with pins. Let $g,h \in \mathcal{G}(A,V)$ be given with the same entries, i.e. $I(g) = I(h)$. A relation $R$ between states of $g$ and states of $h$ is called a bisimulation if:
1. if $p \in I(g)$ and $p \bar{a} \rightarrow t$ in $g$, then there is $v \in S(h)$ with $p \bar{a} \rightarrow v$ in $h$ and $R(t, v)$;
2. if $p \in I(g)$ and $p \bar{a} \rightarrow q$ in $g$, where $q \in O(g)$, then $q \in O(h)$ and $p \bar{a} \rightarrow q$ in $h$;
3. if $p \in I(h)$ and $p \bar{a} \rightarrow v$ in $h$, where $v \in S(h)$, then there is $t \in S(g)$ with $p \bar{a} \rightarrow t$ in $g$ and $R(t, v)$;
4. if $p \in I(h)$ and $p \bar{a} \rightarrow q$ in $h$, where $q \in O(h)$, then $q \in O(g)$ and $p \bar{a} \rightarrow q$ in $g$;
5. if $R(s, t)$ and $s \bar{a} \rightarrow s'$ in $g$, where $s' \in S(g)$, then there is $t' \in S(h)$ such that $t \bar{a} \rightarrow t'$ in $h$ and $R(s', t')$;
6. if $R(s, t)$ and $s \bar{a} \rightarrow q$ in $g$, where $q \in O(g)$, then $q \in O(h)$ and $t \bar{a} \rightarrow q$ in $h$;
7. if $R(s, t)$ and $t \bar{a} \rightarrow q$ in $h$, where $t' \in S(h)$, then there is $s' \in S(g)$ such that $s \bar{a} \rightarrow s'$ in $g$ and $R(s', t')$;
8. if $R(s, t)$ and $t \bar{a} \rightarrow q$ in $h$, where $q \in O(h)$, then $q \in O(g)$ and $t \bar{a} \rightarrow q$ in $g$.

We say $g,h$ are bisimilar, $g \equiv h$, if $I(g) = I(h)$ and there is an bisimulation between $g$ and $h$. Note that, different from [BES94], bisimulating processes do not need to have the same exits.

As usual, bisimulation is an equivalence relation on process graphs. We can divide out this equivalence, and obtain the algebras $\mathcal{G}(A,V)/\equiv$. We will also find that bisimulation is a congruence for all operators defined in section 2. and so we can define these operators on these algebras.

3.2 STRUCTURED OPERATIONAL SEMANTICS.
We can also give the semantics by means of SOS rules. In the following table we have $a,b,c \in A$, $p,q,r,s \in V$, $x,x',y,y' \in P$. We define a transition relation as a subset of $P \times (V \times A) \times P$ and terminating transitions as a subset of $P \times (V \times A) \times V$. Moreover, entry set membership is a subset of $V \times P$, that holds of $p$ and $x$ whenever $p$ is an entry of $x$, we write $p \in i(x)$. Considering the terminating transitions and entry set membership as predicates, we find that the rules satisfy the path format of [BAV93]. Thus, the standard definition of bisimulation (call this $\equiv_{sos}$) on these transition system yields a congruence. It is not hard to prove the following theorem, for all closed terms $s,t$ (where $[.]$ is the interpretation defined in section 2):

3.3 THEOREM: For all closed terms $s,t$: $s \equiv_{sos} t \iff [s] \equiv [t]$. 

4. ALGEBRA.
In this section, we axiomatise bisimulation equivalence on process graphs over a set of pins.

4.1 SYNTAX.
Sorts:
Process algebra with feedback

\[ P \quad \text{sort of processes} \]
\[ V \quad \text{sort of pins} \]

Constants:
\[ p \rightarrow a \rightarrow q \in P \quad \text{atomic action (for each } a \in A, p,q \in V) \]
\[ \emptyset \in P \quad \text{empty process } (\emptyset \in A) \]

Functions:
\[ \rightarrow: V \times P \rightarrow P \quad \text{entry operator} \]
\[ \leftarrow: P \times V \rightarrow P \quad \text{exit operator} \]
\[ \gg: V \times P \rightarrow P \quad \text{initialisation operator} \]
\[ \gg: P \times V \rightarrow P \quad \text{exiting operator} \]
\[ *: P \times V \times V \times P \rightarrow P \quad \text{linking composition} \]
\[ \uparrow: P \times V \times V \rightarrow P \quad \text{pre-feedback} \]
\[ \uparrow: P \times V \times V \rightarrow P \quad \text{feedback} \]

The signature without the last three operators is the signature of BPA_{\delta}(A,V), Basic Process Algebra over a set of actions A and a pin structure V. The full signature is the signature of BPA_{\delta\uparrow}(A,V), BPA with feedback over actions A and pins V.

4.2 Axioms.

First of all, we present the axioms of BPA_{\delta}, replacing \( \emptyset \) by \( \emptyset \). Always, \( X,Y,Z \in P \). Sequential composition always has strongest binding power, alternative composition always has weakest binding power.

\[
\begin{align*}
X + Y &= Y + X & (A1) \\
(X + Y) + Z &= X + (Y + Z) & (A2) \\
X + X &= X & (A3) \\
(X + Y) \cdot Z &= X \cdot Z + Y \cdot Z & (A4) \\
(X \cdot Y) \cdot Z &= X \cdot (Y \cdot Z) & (A5) \\
X + \emptyset &= X & (A6\emptyset) \\
\emptyset \cdot X &= \emptyset & (A7\emptyset)
\end{align*}
\]

\text{TABLE 2. BPA_{\emptyset}.}

Next, entries and exits. Always, \( a \in A, p,q \in V \).
Next, the initialisation and exiting operators.

<table>
<thead>
<tr>
<th>Table 3. Entries and exits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ p \rightarrow (q \rightarrow a \rightarrow r) = p \rightarrow a \rightarrow r ]</td>
</tr>
<tr>
<td>[ p \rightarrow (q \rightarrow \emptyset) = p \rightarrow \emptyset ]</td>
</tr>
<tr>
<td>[ (p \rightarrow a \rightarrow q) \rightarrow r = p \rightarrow a \rightarrow r ]</td>
</tr>
<tr>
<td>[ \emptyset \rightarrow p = \emptyset ]</td>
</tr>
<tr>
<td>[ (p \rightarrow \emptyset) \rightarrow q = p \rightarrow \emptyset ]</td>
</tr>
<tr>
<td>[ p \rightarrow (X + Y) = p \rightarrow X + p \rightarrow Y ]</td>
</tr>
<tr>
<td>[ (X + Y) \rightarrow p = X \rightarrow p + Y \rightarrow p ]</td>
</tr>
<tr>
<td>[ p \rightarrow X \cdot Y = (p \rightarrow X) \cdot Y ]</td>
</tr>
<tr>
<td>[ X \cdot Y \rightarrow p = X \cdot (Y \rightarrow p) ]</td>
</tr>
<tr>
<td>[ X \cdot (p \rightarrow \emptyset) = X \cdot \emptyset ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4. Initialisation, exiting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ p \triangleright \emptyset = \emptyset ]</td>
</tr>
<tr>
<td>[ p \triangleright (p \rightarrow a \rightarrow q) = p \rightarrow a \rightarrow q ]</td>
</tr>
<tr>
<td>[ p \triangleright (q \rightarrow a \rightarrow r) = \emptyset \text{ if } p \neq q ]</td>
</tr>
<tr>
<td>[ p \triangleright (p \rightarrow \emptyset) = p \rightarrow \emptyset ]</td>
</tr>
<tr>
<td>[ p \triangleright (q \rightarrow \emptyset) = \emptyset \text{ if } p \neq q ]</td>
</tr>
<tr>
<td>[ p \triangleright (X + Y) = p \triangleright X + p \triangleright Y ]</td>
</tr>
<tr>
<td>[ p \triangleright X \cdot Y = (p \triangleright X) \cdot Y ]</td>
</tr>
<tr>
<td>[ (p \rightarrow a \rightarrow q) \cdot X = (p \rightarrow a \rightarrow q) \cdot (q \triangleright X) ]</td>
</tr>
<tr>
<td>[ (p \rightarrow a \rightarrow q) \cdot X = (p \rightarrow a \rightarrow r) \cdot (r \rightarrow (q \triangleright X)) ]</td>
</tr>
<tr>
<td>[ \emptyset \triangleright p = \emptyset ]</td>
</tr>
<tr>
<td>[ (p \rightarrow a \rightarrow q) \triangleright q = p \rightarrow a \rightarrow q ]</td>
</tr>
<tr>
<td>[ (p \rightarrow a \rightarrow q) \triangleright r = (p \rightarrow a \rightarrow q) \cdot \emptyset \text{ if } q \neq r ]</td>
</tr>
<tr>
<td>[ (p \rightarrow \emptyset) \triangleright q = p \rightarrow \emptyset ]</td>
</tr>
<tr>
<td>[ (X + Y) \triangleright p = X \triangleright p + Y \triangleright p ]</td>
</tr>
<tr>
<td>[ X \cdot Y \triangleright p = X \cdot (Y \triangleright p) ]</td>
</tr>
</tbody>
</table>

The axioms of tables 2-4 are the axioms of BPA_{pin}(A,V). The middle two axioms of table 4 show the nature of sequential composition. The first of the two expresses the idea, that a process following a process that exits in p, can only start in p. The second of the two expresses the idea, that the names of the interior states do not matter. On the basis of this second axiom, we allow ourselves to abbreviate \((p \rightarrow a \rightarrow q) \cdot (q \rightarrow b \rightarrow r)\) by \((p \rightarrow a \cdot b \rightarrow r)\), and similarly abbreviate

\[
(p \rightarrow a \rightarrow q) \cdot \left( \sum_{i \in \mathbb{N}} (q \rightarrow b_i \rightarrow r_i) \cdot x_i \right) \quad \text{by} \quad (p \rightarrow a) \cdot \left( \sum_{i \in \mathbb{N}} (b_i \rightarrow r_i) \cdot x_i \right).
\]
Finally, we get to the iteration operators. We use linking and pre-feedback in order to obtain an axiomatisation for feedback.

\[
\begin{align*}
\emptyset \mathcal{R}^q Y &= \emptyset \\
(p \rightarrow a \rightarrow q) \mathcal{R}^q Y &= (p \rightarrow a \rightarrow r) \cdot Y \\
(p \rightarrow a \rightarrow s) \mathcal{R}^q Y &= p \rightarrow a \rightarrow s & \text{if } s \neq q \\
(p \rightarrow \emptyset) \mathcal{R}^q Y &= p \rightarrow \emptyset \\
(X + Y) \mathcal{R}^q Z &= X \mathcal{R}^q Z + Y \mathcal{R}^q Z \\
(X \cdot Y) \mathcal{R}^q Z &= X \cdot (Y \mathcal{R}^q Z) \\
X \mathcal{I}^q_p &= X \mathcal{I}^q_p (X \mathcal{I}^q_p) \\
X \mathcal{I}^q_p &= \sum_{r \in (X \mathcal{I}^q_p)} X \mathcal{I}^q_p
\end{align*}
\]

\textbf{TABLE 5. Linking and feedback.}

By convention, a sum over an empty set is equal to \( \emptyset \). The last axiom uses the auxiliary operator \( i \), that was also used in the SOS rules, determining the entry set of a process. It is axiomatised in the following table 6. The complete system of tables 2 through 6 is called BPA\( ^i\)pin\((A,V)\). When these are clear from the context, we usually omit the parameters \( A,V \), and talk about BPA\( ^i\)pin, BPA\( ^i\)pin.

\[
\begin{align*}
i(\emptyset) &= \emptyset \\
i(p \rightarrow a \rightarrow q) &= \{p\} \\
i(p \rightarrow \emptyset) &= \{p\} \\
i(X + Y) &= i(X) \cup i(Y) \\
i(X \cdot Y) &= i(X)
\end{align*}
\]

\textbf{TABLE 6. Entry set.}

5. Theory.

5.1 Definition. The set of basic (BPA\( ^i\)pin) terms is defined inductively:

i. \( \emptyset \) is a basic term, and for all \( p \in V \) \( p \rightarrow \emptyset \) is a basic term;

ii. for all \( p,q \in V, a \in A \) \( p \rightarrow a \rightarrow q \) is a basic term;

iii. if \( t \) is a basic term with \( i(t) = \{q\} \), and \( p \in V, a \in A \), then \( (p \rightarrow a \rightarrow q) \cdot t \) is a basic term;

iv. if \( t,s \) are basic terms, then \( t + s \) is a basic term.

5.2 Theorem (Elimination Theorem): For all closed BPA\( ^i\)pin terms \( s \) there is a basic term \( t \) such that BPA\( ^i\)pin \( \vdash s = t \).
SKETCH OF PROOF: We use term rewrite analysis. Consider the term rewrite system obtained by orienting the following rules from left to right: A3-A7 of BPA\(8\), all axioms of table 3 and all axioms of table 4 except the middle two. About the middle two axioms, we leave out the second, and have to make sure that the first is only used once for every sequential composition in a term. In order to achieve this, we mark the \(\cdot\) sign on the left hand side, and duplicate all other rules involving sequential composition, by replacing each \(\cdot\) by a marked \(\cdot\). Basically, the same trick was used in [KLU93]. Further, we add the rule \((p \rightarrow \emptyset) \cdot X = p \rightarrow \emptyset\). Then, we prove the term rewriting system is terminating by using the lexicographic path ordering. Then, elimination follows since all normal forms are basic terms. Note that we can also eliminate linking composition.

5.3 THEOREM (Soundness Theorem): \(\mathcal{G}(A,V)/_{\mathcal{E}} \vdash \text{BPA}^{\uparrow\text{pin}}\).
PROOF: Omitted.

5.4 THEOREM (Conservativity Theorem): \(\text{BPA}^{\uparrow\text{pin}}\) over a set of actions \(A\) is a conservative extension of BPA\(8\) (with \(\emptyset\) instead of \(8\)) over the set of actions \(\{p \rightarrow a \rightarrow q : a \in A, p,q \in V\} \cup \{p \rightarrow \emptyset : p \in V\}\).
SKETCH OF PROOF: Since all our SOS rules are in path format, operational conservativity follows immediately (see [VER93]). Then the (equational) conservativity follows by a result of [VER93] since the axiomatisation of BPA\(8\) is sound and complete and the axiomatisation of \(\text{BPA}^{\uparrow\text{pin}}\) is sound (theorem 3.5).

5.5 THEOREM (Completeness Theorem): The axiomatisation of BPAPin (so without iterative constructs!) is complete for the model \(\mathcal{G}(A,V)/_{\mathcal{E}}\).
SKETCH OF PROOF: By the general result of [VER93]. In addition to the ingredients of the previous proof, all we need is the elimination theorem.

5.6 DEFINITION: On \(\text{BPA}^{\uparrow\text{pin}}\), we can define projection operators \(\pi_n\) as follows (\(n \geq 0\)):

\[
\begin{align*}
\pi_0(\emptyset) &= \emptyset \\
\pi_0(p \rightarrow a \rightarrow q) &= p \rightarrow \emptyset \\
\pi_{n+1}(p \rightarrow a \rightarrow q) &= p \rightarrow a \rightarrow q \\
\pi_n(p \rightarrow \emptyset) &= p \rightarrow \emptyset \\
\pi_0((p \rightarrow a \rightarrow q) \cdot X) &= p \rightarrow \emptyset \\
\pi_{n+1}((p \rightarrow a \rightarrow q) \cdot X) &= (p \rightarrow a \rightarrow q) \cdot \pi_n(X) \\
\pi_n(X + Y) &= \pi_n(X) + \pi_n(Y)
\end{align*}
\]

TABLE 7. Projection.

5.7 THEOREM (Projection theorem): Let \(t\) be a closed \(\text{BPA}^{\uparrow\text{pin}}\) term, and let \(n \geq 0\). Then \(\pi_n(t)\) can be written as a basic term.
PROOF: Omitted.
5.8 Theorem (Representation theorem): Let $g$ be a regular process (an element of $\mathcal{G}(A,V)$ with finitely many states). Then there is a closed BPA$^\uparrow$ pin term $t$ such that $[t] \models g$.

Sketch of proof: An example is provided in 5.9 below. The general proof is along the same lines.

5.9 Example: Consider the process graph $g$ in fig. 5, with pins $\{p,q\} \ (p \neq q)$. Then there is no closed BPA$^\uparrow$ pin term $t$ (so without iterative constructs!) such that $[t] \models g$. To write this as a BPA$^\uparrow$ pin term (even up to isomorphism) is easy: take fresh names $r,s$ and consider $((p \rightarrow a \rightarrow r) + (q \rightarrow a \rightarrow s) + (r \rightarrow b \rightarrow s) + (r \rightarrow c \rightarrow p) + (s \rightarrow c \rightarrow q))^{\uparrow} r^{\uparrow} s$.

![Figure 5](image)

6. Parallel Composition.

In this section, we extend the theory of the previous sections by parallel composition operators. Now we have a third parameter of the theory, the communication function. This is a partial, commutative and associative function $\gamma: A \times A \rightarrow A$.

6.1 Graph model.

We add a number of operators.

1. Left global state pairing. Let a process graph $g$ be given, and let $p \in V$. $p \uparrow g$ has states $S(g)$, entries $\{p\} \times I(g)$ and exits $\{p\} \times O(g)$. It has same edges between states as $g$, and the pins of edges starting from an entry or ending in an exit are paired on the left with $p$.

2. Right global state pairing. Let a process graph $g$ be given, and let $p \in V$. $g \downarrow p$ has states $S(g) \times \{p\}$, entries $I(g) \times \{p\}$ and exits $O(g) \times \{p\}$. It has same edges between states as $g$, and the pins of edges starting from an entry or ending in an exit are paired on the right with $p$.

3. Parallel composition. Let process graphs $g,h$ be given. Assume that the state sets of $g$ and $h$ are disjoint. Take a set of fresh state names for each exit, i.e. assume there are sets $O^*(g) = \{p^* : p \in O(g)\}$, $O^*(h) = \{p^* : p \in O(h)\}$ of states not occurring elsewhere in $g$ or $h$ (this is needed in case an exit also occurs as an entry). The set of states of $g \parallel h$ is $((S(g) \cup I(g) \cup O^*(g)) \times (S(h) \cup I(h) \cup O^*(h))) - ((I(g) \times I(h)) \cup (O^*(g) \times O^*(h)))$. The transition relation is given by:

a. for each state $v$ in $g$, and each transition $t \rightarrow t'$ in $h$, there is a transition $v \rightarrow (t \rightarrow v \rightarrow t')$ (here, we take $v \rightarrow (t \rightarrow v \rightarrow t')$ if $t \in O(h)$)

b. for each state $t$ in $h$, and each transition $v \rightarrow v'$ in $g$, there is a transition $v \rightarrow (t \rightarrow v \rightarrow v')$ (here, we take $v \rightarrow (t \rightarrow v \rightarrow v')$ if $v' \in O(g)$).
c. for each pair of transitions \( v \Rightarrow (t \xrightarrow{a} v') \Rightarrow (t' \xrightarrow{b} v') \) such that \( \gamma(a, b) \) is defined, say \( \gamma(a, b) = c \), there is a transition \( v \Rightarrow (t \xrightarrow{c} v') \Rightarrow (t' \xrightarrow{t'} v') \) (again, \( v'' \) and/or \( t'' \) when needed) such that \( y(a, b) \) is defined, say \( y(a, b) = c \), there is a transition \( v \Rightarrow (t \xrightarrow{t''} v') \Rightarrow (t' \xrightarrow{t''} v') \).

The entries are \( I(g) \times I(h) \), the exits are \( O(g) \times O(h) \).

Note that the pairing function need not be commutative or associative. Thus, the parallel composition may not be commutative or associative. As an example, we show \( (p \rightarrow a \rightarrow p) \parallel (q \rightarrow b \rightarrow q) \), where \( \gamma(a, b) = c \), in fig. 6.

\[ \text{FIGURE 6.} \]

4. **Left merge.** The graph of \( g \parallel h \) has the same states as the graph of \( g \parallel h \), the same pins and the same transitions except that the transitions \( v \Rightarrow (t \xrightarrow{a} v') \Rightarrow (t' \text{ with } v, t \text{ entries, and } t' \text{ not an entry, are omitted.} \)

5. **Right merge.** The graph of \( g \parallel h \) has the same states as the graph of \( g \parallel h \), the same pins and the same transitions except that the transitions \( v \Rightarrow (t \xrightarrow{a} v') \Rightarrow (t' \text{ with } v, t \text{ entries, and } v' \text{ not an entry, are omitted.} \)

6. **Communication merge.** The graph of \( g \mid h \) has the same states as the graph of \( g \parallel h \), the same pins and the same transitions except that the transitions \( v \Rightarrow (t \xrightarrow{a} v') \Rightarrow (t' \text{ with } v' \text{ and } t' \text{ an entry are omitted.} \)

7. **Encapsulation.** Let process graph \( g \) be given, and let \( H \subseteq A \). The graph of \( \partial_H(g) \) has the same states as the graph of \( g \), the same pins and the same transitions except that all transitions \( v \xrightarrow{a} t \) with \( a \in H \) are omitted.

### 6.2 Structured Operational Semantics.

We show the SOS rules for the additional operators.

\[
\begin{array}{ccc}
\frac{x \xrightarrow{r,a} x'}{p \xrightarrow{r,a} x' \xrightarrow{p} \xrightarrow{r,a} p \xrightarrow{p} x'} & \frac{x \xrightarrow{r,a} q}{p \xrightarrow{r,a} x \xrightarrow{p} \xrightarrow{r,a} p \xrightarrow{p} q} & \frac{q \in i(x)}{p \xrightarrow{q} q \in i(p) \xrightarrow{p} l(x)} \\
\frac{p \xrightarrow{r, a} x \xrightarrow{p} \xrightarrow{r, a} p \xrightarrow{p} x}{x \xrightarrow{r, a} x \xrightarrow{p} \xrightarrow{r, a} p \xrightarrow{p} x} & \frac{q \in i(x)}{x \xrightarrow{r, a} x \xrightarrow{p} \xrightarrow{r, a} p \xrightarrow{p} q} & \frac{q \in i(x)}{x \xrightarrow{r, a} x \xrightarrow{p} \xrightarrow{r, a} p \xrightarrow{p} q} \\
\end{array}
\]
6.3 ALGEBRA.

We repeat the added signature:

\[
\begin{align*}
\triangleright t &: V \times P \to P & \text{left global state pairing} \\
\triangleleft t &: P \times V \to P & \text{right global state pairing} \\
\triangleright: &: P \times P \to P & \text{standard ACP operator, for } \triangleright \in \{+,-,\ll,\gg,\}
\end{align*}
\]

\[
\begin{align*}
\ll &: P \to P & \text{standard ACP encapsulation, for } H \subseteq A \\
\triangleright \ll &: P \times P \to P & \text{right-merge.}
\end{align*}
\]

The following table shows axioms for the global state pairing operators. We need these in the axiomatisation of parallel composition.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p \rightarrow q ), ( a \in H )</td>
<td>( p \rightarrow q ), ( a \in H )</td>
</tr>
</tbody>
</table>

TABLE 8. Structured operational semantics.
Now we have the ingredients that together allow to axiomatise parallel composition. As parallel composition is not in general commutative, we need both a left-merge and a right-merge operator. Recall that we have a partial, commutative and associative communication function $\gamma$ on $A$. The axioms of $\text{BPA}_{\text{pin}}$ plus the axioms in tables 9 and 10 constitute the theory $\text{ACP}_{\text{pin}}$. Likewise, $\text{BPA}^\uparrow\text{pin}$ plus axioms in tables 9 and 10 yields $\text{ACP}^\uparrow\text{pin}$. On the basis of these axiomatisations, it is easy to obtain axiomatisations for theories without communication, $\text{PA}_{\text{pin}}$ and $\text{PA}^\uparrow\text{pin}$.
For the extensions defined in this section, we can obtain results similar to the results in section 5.

7. ABSTRACTION.
In this section we define abstraction and branching bisimulation for process algebra over a set of pins.

7.1 PROCESS GRAPHS.
Suppose we have a special constant \( \tau \), and we consider process graphs over \( A \cup \{\tau\} \) and \( V \). Again, we consider process graphs modulo state isomorphism. On an element of this domain of process graphs, the abstraction operator \( \tau_i \) is simply defined by relabeling all edges with labels from \( I \subset A \) by \( \tau \). We put \( s \Rightarrow t \), for nodes \( s, t \) of a process graph, if \( t \) can be reached from \( s \) by doing a number of \( \tau \)-steps (0 or more). The notion of branching bisimulation is defined as follows:

<table>
<thead>
<tr>
<th>Table 10. Parallel composition and encapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset</td>
</tr>
<tr>
<td>( X</td>
</tr>
<tr>
<td>( (p \rightarrow \emptyset</td>
</tr>
<tr>
<td>( X</td>
</tr>
<tr>
<td>( (p \rightarrow a \rightarrow q)</td>
</tr>
<tr>
<td>( (p \rightarrow a \rightarrow q)</td>
</tr>
<tr>
<td>( (p \rightarrow a \rightarrow q)</td>
</tr>
<tr>
<td>( (X + Y)</td>
</tr>
<tr>
<td>( X</td>
</tr>
</tbody>
</table>

- \( \partial_H(\emptyset) = \emptyset \)
- \( \partial_H(p \rightarrow \emptyset) = p \rightarrow \emptyset \)
- \( \partial_H(p \rightarrow a \rightarrow q) = p \rightarrow a \rightarrow q \) if \( a \notin H \)
- \( \partial_H(p \rightarrow a \rightarrow q) = p \rightarrow \emptyset \) if \( a \in H \)
- \( \partial_H(X + Y) = \partial_H(X) + \partial_H(Y) \)
- \( \partial_H(X - Y) = \partial_H(X) \cdot \partial_H(Y) \)
5. if $R(s, t)$ and $s \xrightarrow{a} s'$ in $g$, where $s' \in S(g)$, then either $a=\tau$ and $R(s', t)$ or there are $t', t'' \in S(h)$ such that $t \Rightarrow t' \xrightarrow{a} t''$ in $h$ and $R(s, t')$, $R(s', t'')$ or there are $t' \in S(h)$, $q \in O(h)$ such that $t \Rightarrow t' \xrightarrow{a} q$ in $h$ and $R(s, t')$ and $s'$ is a $q$-semi-endpoint;

6. if $R(s, t)$ and $s \xrightarrow{a} q$ in $g$, where $q \in O(g)$, then $q \in O(h)$ and either $a=\tau$ and $t$ is a $q$-semi-endpoint or there are $t', t'' \in S(h)$ such that $t \Rightarrow t' \xrightarrow{a} t''$ in $h$ and $R(s, t')$, $t''$ is a $q$-semi-endpoint or there is $t' \in S(h)$ such that $t \Rightarrow t' \xrightarrow{a} q$ in $h$ and $R(s, t')$;

7. if $R(s, t)$ and $t \xrightarrow{a} t'$ in $h$, where $t' \in S(h)$, then either $a=\tau$ and $R(s, t')$ or there are $s', s'' \in S(g)$ such that $s \Rightarrow s' \xrightarrow{a} s''$ in $g$ and $R(s', t)$, $R(s', t')$ or there are $s' \in S(g)$, $q \in O(g)$ such that $s \Rightarrow s' \xrightarrow{a} q$ in $g$ and $R(s', t)$ and $t'$ is a $q$-semi-endpoint;

8. if $R(s, t)$ and $t \xrightarrow{a} q$ in $h$, where $q \in O(h)$, then $q \in O(g)$ and either $a=\tau$ and $s$ is a $q$-semi-endpoint or there are $s', s'' \in S(g)$ such that $s \Rightarrow s' \xrightarrow{a} s''$ in $g$ and $R(s', t)$, $s''$ is a $q$-semi-endpoint or there is $s' \in S(g)$ such that $s \Rightarrow s' \xrightarrow{a} q$ in $g$ and $R(s', t)$.

Here, we say that an interior node $t$ is a $q$-semi-endpoint ($q \in V$) if $t \Rightarrow q$ and $t \xrightarrow{a} s$ implies $a=\tau$ and $s$ is a $q$-(semi-)endpoint.

We say $g, h$ are rooted branching bisimilar, $g \equiv_{\text{rb}} h$, if $I(g) = I(h)$ and there is a rooted branching bisimulation between $g$ and $h$. Rooted branching bisimulation, as defined here, is an equivalence relation on process graphs. We can divide out this equivalence, and obtain the algebras $G(A\cup\{\tau\}, V)/\equiv_{\text{rb}}$. We will also find that rooted branching bisimulation is a congruence for all operators defined in sections 2 and 6.

### 7.3 Structured Operational Semantics

The SOS rules for the abstraction operator are straightforward, and displayed in table 11. On the basis of the previous definition, it is not hard to define rooted branching bisimulation also on transition systems. We obtain the same equivalence as in 3.3.

<table>
<thead>
<tr>
<th>$x \xrightarrow{p.\alpha} x'$, $a \in I$</th>
<th>$x \xrightarrow{p.\alpha} q$, $a \in I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1(x) \xrightarrow{p.\tau} \tau_1(x')$</td>
<td>$\tau_1(x) \xrightarrow{p.\tau} q$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x \xrightarrow{p.\alpha} x'$, $a \notin I$</th>
<th>$x \xrightarrow{p.\alpha} q$, $a \notin I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1(x) \xrightarrow{p.\alpha} \tau_1(x')$</td>
<td>$\tau_1(x) \xrightarrow{p.\alpha} q$</td>
</tr>
</tbody>
</table>

| $r \in i(x)$ | $r \in i(\tau_1(x))$ |

**TABLE 11.** Structured operational semantics.

### 7.4 Axiomatization

An axiomatisation of silent step and abstraction is given in table 12.
Process algebra with feedback

\[ \tau_1(\emptyset) = \emptyset \]
\[ \tau_1(p \rightarrow \emptyset) = p \rightarrow \emptyset \]
\[ \tau_1(p \rightarrow a \rightarrow q) = p \rightarrow \tau \rightarrow q \quad \text{if } a \in I \]
\[ \tau_1(X + Y) = \tau_1(X) + \tau_1(Y) \]
\[ \tau_1(X \cdot Y) = \tau_1(X) \cdot \tau_1(Y) \]
\[ X \cdot (p \rightarrow \tau \rightarrow q) = (X \Rightarrow p) \rightarrow q \]
\[ X \cdot ((p \rightarrow \tau \rightarrow p) \cdot (Y + Z) + Y) = X \cdot (Y + Z) \]

**Table 12. Abstraction and \( \tau \)-laws.**

### 7.5 Fair Abstraction.

Table 13 shows a law for removing a \( \tau \)-loop. This law corresponds to the law KFAR\( i \) of [BAW90]. The process on the left-hand side of the equation is shown in fig. 7. We have to give an axiomatisation of the set of exits of a process (similar to the set of entries in table 6) in order to turn this into an algebraic law.

\[ ((p \rightarrow \tau \rightarrow q) + (q \rightarrow \tau \rightarrow q) + (q \Rightarrow X)) \tau q = (p \rightarrow \tau \rightarrow q) \cdot X \]

**Table 13. Fair abstraction rule, simplest case.**

![Figure 7](image)

### 8. Examples.

In this section, we consider a few simple examples of the use of process algebra over pins. Given a (finite) data set \( D \), we assume the set of pins \( V \) satisfies \( D \cup \{ \bar{\emptyset} \} \subseteq V \), where \( \bar{\emptyset} \notin D \). We use \( \bar{\emptyset} \) as a default element. We abbreviate \( \bar{\emptyset} \rightarrow a \rightarrow \bar{\emptyset} \) by \( a \), for each \( a \in A \), and we abbreviate \( \bar{\emptyset} \rightarrow \emptyset \) by \( \delta \).

#### 8.1 Buffers.

We specify a two element buffer in BPA*pin(\( A, V \)). An input of \( d \in D \) is denoted by action \( r_1(d) \), an output of \( d \in D \) by \( s_2(d) \). The iteration construct *, binary Kleene star, is defined by the usual equation (see [BEBP94]):
\[ x^*y = x(x^*y) + y \]

**TABLE 14.** Binary Kleene star.

\[ B = \sum_{d \in D} r_1(d) \cdot (d \triangleright (B' \cdot \delta)) \]

\[ B' = \sum_{d \in D} d \rightarrow ((s_2(d) \cdot \sum_{e \in D} r_1(e) \rightarrow e) + (\sum_{e \in D} r_1(e) \cdot s_2(d) \rightarrow e)). \]

A two element buffer over a data set of at most two elements can be specified as a closed term over \( \text{ACP}^* \) (without feedback); if there are 3 distinct data elements, this is not possible any more. The above shows that it is possible over \( \text{BPA}^* \text{pin} \). This shows that generalising to processes over pin structures adds expressive power.

### 8.2 Alternating Bit Protocol

We consider the well-known example of the Alternating Bit Protocol. We give the specification using the iteration operator, taken from [BEBP93]. There, also a verification can be found.

We assume that elements of \( D \) are to be transmitted from sender \( S \) to receiver \( R \) using unreliable channels \( K, L, B = \{0,1\} \). We use the standard communication function given by \( \gamma(r_k(x), s_k(x)) = c_k(x) \) (see [BAW90]). The communication links are as shown in fig. 8. We have the following specifications.

**FIGURE 8.** ABP.

\[ K = \left( \sum_{d \in D, b \in B} r_2(db) \cdot (s_3(db) + i \cdot s_3(\perp)) \right) \cdot \delta \]

\[ L = \left( \sum_{b \in B} r_5(b) \cdot (s_6(b) + i \cdot s_6(\perp)) \right) \cdot \delta \]

\[ S = (S0-S1) \cdot \delta \]

\[ S_b = \sum_{d \in D} r_1(d) \cdot s_2(db) \cdot \left( (r_6(1-b)+r_6(\perp)) \cdot s_2(db) \right)^* \cdot r_6(b) \quad \text{for } b = 0,1. \]

\[ R = (R1-R0) \cdot \delta \]

\[ R_b = \left( \left( \sum_{d \in D} r_3(db) + r_3(\perp) \right) \cdot s_5(b) \right)^* \left( \sum_{d \in D} r_3(d(1-b)) \cdot s_4(d) \right) \cdot s_5(1-b) \quad \text{for } b = 0,1. \]
The protocol is now given by \( \text{ABP} = \partial_{H}(S \parallel K \parallel L \parallel R) \), with

\[ H = \{ r_k(x), s_k(x) : k \in \{2,3,5,6\}, x \in (D \times B) \cup B \cup \{L\} \} \]

Use of pins now allows to give a direct specification of a part of this system. As an example, we consider \( \partial_{H2}(S \parallel K) \), with \( H2 = \{ r_2(x), s_2(x) : x \in D \times B \} \). We claim this expression satisfies the following specification, again using data values as pin names:

\[ X = (X_0 \cdot X_1) \cdot \delta \]
\[ X_b = \sum_{d \in D} r_1(d) \cdot c_2(db) \cdot (i \cdot s_3(db) + i \cdot s_3(\bot)) \cdot (d \gg Y_b) \quad \text{for } b = 0,1 \]
\[ Y_b = \sum_{d \in D} d \rightarrow \left( ((r_6(1-b)+r_6(\bot)) \cdot c_2(db) \cdot (i \cdot s_3(db) + i \cdot s_3(\bot)))^* \cdot r_6(b) \right) \quad \text{for } b = 0,1 \]

9. SUBALGEBRAS OF REDUCED MODELS.

We consider a specific choice for the pin structure \( V \), and consider some subalgebras. With the abbreviations introduced in the beginning of section 8 (\( a \) for \( \sqrt{\Rightarrow} a \rightarrow \sqrt{\Rightarrow} \), and \( \delta \) for \( \sqrt{\Rightarrow} \emptyset \)), we have embedded the signature of ACP in the signature of ACPpin. However, we get unwanted equation, as e.g. \( a \cdot (b \parallel c) = a \cdot \delta \) since pin names do not match. Therefore, we will divide out the equation \( \sqrt{\Rightarrow} \emptyset \cdot \sqrt{\Rightarrow} = \sqrt{\Rightarrow} \).

9.1 STANDARD PROCESS ALGEBRA.

Suppose the set of pins \( V \) contains \( \sqrt{\Rightarrow} \) with \( \sqrt{\Rightarrow} \cdot \sqrt{\Rightarrow} = \sqrt{\Rightarrow} \), and \( A \) is a given finite set. Consider the signature \( \Sigma \) that has constants \( \{ \sqrt{\Rightarrow} a \rightarrow \sqrt{\Rightarrow} : a \in A \} \cup \{ \sqrt{\Rightarrow} \emptyset \} \) and operators \( \{ +, \cdot, \parallel, \|, \} \cup \{ \partial_{H} : H \subseteq A \} \). Now consider the minimal subalgebra of the \( \Sigma \)-reduct of the initial algebra of ACPpin. We claim that standard ACP is an axiomatisation of this algebra, again substituting \( a \) for \( \sqrt{\Rightarrow} a \rightarrow \sqrt{\Rightarrow} \), and \( \delta \) for \( \sqrt{\Rightarrow} \emptyset \). Thus, ACP is a subalgebra of a reduced model specification (an SRM specification) of ACPpin. For more information on SRM specifications, we refer to [BAB94].

9.2 ACP WITH \( \emptyset \).

Now extend the signature \( \Sigma \) above with the extra constant \( \emptyset \). We claim that the axioms in table 15 constitute an SRM specification for this signature \( (a,b \in A \cup \{\delta\}) \).

10. CONCLUSIONS.

We generalised process algebra to processes with multiple entries and exits, so-called pins. On the resulting model of process graphs modulo bisimulation we can define the feedback operator from flowchart theory, besides all usual operators from process algebra. This gives us more expressive power in the specification of processes parametrised by data. Moreover, we get the original ACP back as an SRM specification of the general theory. As a side effect, this introduces a new constant \( \emptyset \) into ACP. As future work, we leave the exact determination of the interaction of all the special constants: \( \emptyset \),
δ (inaction, [BEK84]), nil (CCS termination [Mit89], [ACH92]), ε (termination option [VRA91]), τ (silent step [Mit89]), Δ (divergence [ACH92]), χ (chaos [BRHR84]), 0 (zero process [BAB90]).

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X + Y = Y + X</td>
<td></td>
</tr>
<tr>
<td>(X + Y) + Z = X + (Y + Z)</td>
<td></td>
</tr>
<tr>
<td>X + X = X</td>
<td></td>
</tr>
<tr>
<td>(X + Y) - Z = X - (Y - Z)</td>
<td></td>
</tr>
<tr>
<td>X + Ø = X</td>
<td></td>
</tr>
<tr>
<td>Ø · X = Ø</td>
<td></td>
</tr>
<tr>
<td>X · Ø = X · δ</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>a + δ = a</td>
<td></td>
</tr>
<tr>
<td>a · X + δ = a · X</td>
<td></td>
</tr>
<tr>
<td>δ · X = δ</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b = γ(a, b)</td>
</tr>
<tr>
<td>a</td>
<td>b = δ</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
X | Y &= X | Y + Y | X | Y \\
X | Ø &= Ø \\
Ø | X &= Ø \\
a | (X + δ) &= a · (X + δ) \\
a · X | (Y + δ) &= a · (X | (Y + δ)) \\
(X + Y) | Z &= X | Z + Y | Z \\
Ø | X &= Ø \\
X | Ø &= Ø \\
a · X | b &= (a | b) · X \\
a | b · X &= (a | b) · X \\
a · X | b · Y &= (a | b) · (X | Y) \\
(X + Y) | Z &= X | Z + Y | Z \\
X | (Y + Z) &= X | Y + X | Z
\end{align*}
\]

\[d_H(Ø) = Ø\]
\[d_H(a) = a\quad\text{if } a \notin H\]
\[d_H(δ) = δ\quad\text{if } a \notin H\]
\[d_H(X + Y) = d_H(X) + d_H(Y)\]
\[d_H(X · Y) = d_H(X) · d_H(Y)\]

**Table 15. ACPØ.**

REFERENCES.


In this series appeared:

91/01 D. Alstein

91/02 R.P. Nederpelt
H.C.M. de Swart

91/03 J.P. Katoen
L.A.M. Schoenmakers

91/04 E. v.d. Sluis
A.F. v.d. Stappen

91/05 D. de Reus

91/06 K.M. van Hee

91/07 E. Poll

91/08 H. Schepers

91/09 W.M.P. v.d. Aalst

91/10 R.C. Backhouse
P.J. de Bruin
P. Hoogendijk
G. Malcolm
E. Voermans
J. v.d. Woude

91/11 R.C. Backhouse
P.J. de Bruin
G. Malcolm
E. Voermans
J. van der Woude

91/12 E. van der Sluis

91/13 F. Rietman

91/14 P. Lemmens

91/15 A.T.M. Aerts
K.M. van Hee

91/16 A.J.J.M. Marcelis


Implication. A survey of the different logical analyses "if...,then...", p. 26.

Parallel Programs for the Recognition of $P$-invariant Segments, p. 16.

Performance Analysis of VLSI Programs, p. 31.

An Implementation Model for GOOD, p. 18.

SPECIFICATIEMETHODEN, een overzicht, p. 20.

CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.

Terminology and Paradigms for Fault Tolerance, p. 25.

Interval Timed Petri Nets and their analysis, p. 53.

POLYNOMIAL RELATORS, p. 52.

Relational Catamorphism, p. 31.


A note on Extensionality, p. 21.

The PDB Hypermedia Package. Why and how it was built, p. 63.


An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
| 91/17 | A.T.M. Aerts  
P.M.E. de Bra  
| 91/18 | Rik van Geldrop | Transformational Query Solving, p. 35. |
| 91/19 | Erik Poll | Some categorical properties for a model for second order lambda calculus with subtyping, p. 21. |
| 91/20 | A.E. Eiben  
| 91/21 | J. Coenen  
W.-P. de Roever  
J. Zwiers | Assertional Data Reification Proofs: Survey and Perspective, p. 18. |
| 91/23 | K.M. van Hee  
L.J. Somers  
M. Voorhoeve | Z and high level Petri nets, p. 16. |
| 91/24 | A.T.M. Aerts  
D. de Reus | Formal semantics for BRM with examples, p. 25. |
| 91/25 | P. Zhou  
J. Hooman  
R. Kuiper | A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52. |
| 91/26 | P. de Bra  
G.J. Houben  
| 91/27 | F. de Boer  
C. Palamidessi | Embedding as a tool for language comparison: On the CSP hierarchy, p. 17. |
| 91/28 | F. de Boer | A compositional proof system for dynamic process creation, p. 24. |
| 91/29 | H. Ten Eikelder  
R. van Geldrop | Correctness of Acceptor Schemes for Regular Languages, p. 31. |
| 91/30 | J.C.M. Baeten  
F.W. Vaandrager | An Algebra for Process Creation, p. 29. |
<p>| 91/31 | H. ten Eikelder | Some algorithms to decide the equivalence of recursive types, p. 26. |
| 91/33 | W. v.d. Aalst | The modelling and analysis of queueing systems with QNM-ExSpect, p. 23. |
| 91/34 | J. Coenen | Specifying fault tolerant programs in deontic logic, p. 15. |</p>
<table>
<thead>
<tr>
<th>No.</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18.</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra, p. 45.</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
<td>Set theory and nominalisation, Part II, p. 22.</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
<td>The total order assumption, p. 10.</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
<td>A system at the cross-roads of functional and logic programming, p. 36.</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljée</td>
<td>Integrity checking in deductive databases; an exposition, p. 32.</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis, p. 20.</td>
</tr>
<tr>
<td>92/18</td>
<td>R.Nederpelt, F. Kamareddine</td>
<td>A unified approach to Type Theory through a refined lambda-calculus, p. 30.</td>
</tr>
<tr>
<td>92/20</td>
<td>F.Kamareddine</td>
<td>Are Types for Natural Language? P. 32.</td>
</tr>
</tbody>
</table>
Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

A useful lambda notation, p. 17.

Nominalization, Predication and Type Containment, p. 40.

Bottum-up Abstract Interpretation of Logic Programs, p. 33.

A Programming Logic for Fo, p. 15.

A modelling method using MOVIE and SimCon/ExSpect, p. 15.

A taxonomy of keyword pattern matching algorithms, p. 50.

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

A continuous version of the Prisoner's Dilemma, p. 17.

Quicksort for linked lists, p. 8.

Deterministic and randomized local search, p. 78.

A congruence theorem for structured operational semantics with predicates, p. 18.

On the unavoidability of metastable behaviour, p. 29.

Exercises in Multiprogramming, p. 97.

A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.


<table>
<thead>
<tr>
<th>Year</th>
<th>Code</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/16</td>
<td>H. Schepers J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
<td></td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
<td></td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
<td></td>
</tr>
<tr>
<td>93/19</td>
<td>G-J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p.21.</td>
<td></td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
<td></td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
<td></td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions, p. 38.</td>
<td></td>
</tr>
<tr>
<td>93/26</td>
<td>W.M.P. van der Aalst</td>
<td>Multi-dimensional Petri nets, p. 25.</td>
<td></td>
</tr>
<tr>
<td>93/27</td>
<td>T. Kloks and D. Kratsch</td>
<td>Finding all minimal separators of a graph, p. 11.</td>
<td></td>
</tr>
<tr>
<td>93/28</td>
<td>F. Kamareddine and R. Nederpelt</td>
<td>A Semantics for a fine $\lambda$-calculus with de Bruijn indices, p. 49.</td>
<td></td>
</tr>
<tr>
<td>93/29</td>
<td>R. Post and P. De Bra</td>
<td>GOLD, a Graph Oriented Language for Databases, p. 42.</td>
<td></td>
</tr>
<tr>
<td>93/30</td>
<td>J. Deogun T. Kloks D. Kratsch H. Müller</td>
<td>On Vertex Ranking for Permutation and Other Graphs, p. 11.</td>
<td></td>
</tr>
<tr>
<td>93/31</td>
<td>W. Körver</td>
<td>Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.</td>
<td></td>
</tr>
</tbody>
</table>
93/33 L. Loyens and J. Moonen
ILIAS, a sequential language for parallel matrix computations, p. 20.

93/34 J.C.M. Baeten and J.A. Bergstra
Real Time Process Algebra with Infinitesimals, p. 39.

93/35 W. Ferrer and P. Severi
Abstract Reduction and Topology, p. 28.

93/36 J.C.M. Baeten and J.A. Bergstra
Non Interleaving Process Algebra, p. 17.

93/37 J. Brunekreef J-P. Katoen R. Koymans S. Mauw
Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

93/38 C. Verhoef
A general conservative extension theorem in process algebra, p. 17.

93/39 W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee
Job Shop Scheduling by Constraint Satisfaction, p. 22.

93/40 P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein

93/41 A. Bijlsma
Temporal operators viewed as predicate transformers, p. 11.

93/42 P.M.P. Rambags
Automatic Verification of Regular Protocols in P/T Nets, p. 23.

93/43 B.W. Watson
A taxonomy of finite automata construction algorithms, p. 87.

93/44 B.W. Watson
A taxonomy of finite automata minimization algorithms, p. 23.

93/45 E.J. Luit J.M.M. Martin
A precise clock synchronization protocol, p.

93/46 T. Kloks D. Kratsch J. Spinrad

93/47 W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky

93/48 R. Gerth
Verifying Sequentially Consistently Consistent Memory using Interface Refinement, p. 20.
<table>
<thead>
<tr>
<th>Paper</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>94/01</td>
<td>P. America, M. van der Kammen, R.P. Nederpelt, O.S. van Roosmalen, H.C.M. de Swart</td>
<td>The object-oriented paradigm, p. 28.</td>
</tr>
<tr>
<td>94/02</td>
<td>F. Kamareddine, R.P. Nederpelt</td>
<td>Canonical typing and ( \Pi )-conversion, p. 51.</td>
</tr>
<tr>
<td>94/04</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.</td>
</tr>
<tr>
<td>94/06</td>
<td>T. Basten, T. Kunz, J. Black, M. Coffin, D. Taylor</td>
<td>Time and the Order of Abstract Events in Distributed Computations, p. 29.</td>
</tr>
<tr>
<td>94/08</td>
<td>O.S. van Roosmalen</td>
<td>A Hierarchical Diagrammatic Representation of Class Structure, p. 22.</td>
</tr>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Process Algebra with Partial Choice, p. 16.</td>
</tr>
<tr>
<td>94/10</td>
<td>T. van Doornbos</td>
<td>The testing Paradigm Applied to Network Structure, p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljée</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peremans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doornbos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>1994</td>
<td>R. Bloo, F. Kamareddine, and R. Nederpelt</td>
<td>Beyond β-Reduction in Church's ( \lambda \rightarrow ), p. 22.</td>
</tr>
<tr>
<td>1994</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.</td>
</tr>
<tr>
<td>1994</td>
<td>S. Mauw and M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>1994</td>
<td>D. Dams, O. Grumberg, and R. Gerth</td>
<td>Abstract Interpretation of Reactive Systems: Abstractions Preserving ( \forall )CTL*, ( \exists )CTL* and CTL*, p. 28.</td>
</tr>
<tr>
<td>1994</td>
<td>T. Kloks</td>
<td>( K_{1,3} )-free and ( W_4 )-free graphs, p. 10.</td>
</tr>
</tbody>
</table>