Frame-Based Process Logics

Jan A. Bergstra[1,2] Alban Ponse[1]

[1] *University of Amsterdam, Programming Research Group Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*

[2] *Utrecht University, Department of Philosophy Heidelberglaan 8, 3584 CS Utrecht, The Netherlands*

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

ABSTRACT. We consider frames that can be defined by polynomials over states and transitions. States are elements from the natural numbers with pairing, and (state-)transitions are labeled with actions from some finite set. We study a connection with process algebra: pointing a frame by distinguishing either two states that function as a root and a tail, respectively, implements a terminating process. Only selecting a root can be used to model a non-terminating process. This implies that frame properties can be investigated with process algebraic means.
Providing further structure on frames by inserting propositions at the states, gives a basic "framework" for Floyd Hoare logic, or a modal logic with modalities $\Box_a$ for all actions a considered.

Note: This paper will be published in: *Modal Logic and Process algebra,* Alban Ponse, Yde Venema and Maarten de Rijke (eds.), CSLI publications, to appear.

1 Introduction

This paper focusses on *frames* in the setting of process algebra, Floyd Hoare logic, and modal logic. Compared with frames as used in Kripke semantics and modal logic, the most important difference is that we take the accessibility relation ternary: we consider transitions $s \xrightarrow{a} s'$ between states s and s' with a label a taken from a finite set of actions. (In the next section the basics of algebraic frame representation are introduced.)

In a simultaneous paper (Bergstra and Ponse 1994) we develop a direct, algebraic notation for frames, and give some theoretical information and examples. In this paper we study frames at a more abstract level. More precisely, we consider three such levels, or "frame-based process logics" as stated in the title. First we approach a simple class of frames with *process algebra*, or more specifically with axiom systems defined in the setting of ACP (Algebra of Communicating Processes, see Bergstra and Klop 1984, 1985). In Section 3 we show that process algebra can be seen as a "frame-based logic", or rather as an abstract version of a frame-based algebra. Though, due to the axiomatic approach, different semantical settings have emerged for process algebra, its correspondence with frames is quite natural. One easily depicts "frames" in reasoning about "processes". More precisely, selecting a root state in some frame (and possibly a tail representing successful termination), can be used to extract a process algebra expression. To this purpose, we introduce a process extraction operation. As a consequence, pointed frames can be subject to process algebraic reasoning. In Section 4 we illustrate this by analyzing two counters. Next, in Section 5, we also regard frames with "signal insertion", bringing us into the semantical setting of propositional dynamic logic. In particular, states are equipped with propositions. This provides a semantical basis for process algebra with conditions, which is shortly recalled in Section 6. (In Baeten and Bergstra 1994, a similar connection is studied.) For signal inserted frames, we introduce in Section 7 a simple *Floyd Hoare logic* over conditional processes. We think its virtue is straightforwardness: each program construct induces its own, simple correctness rule. In Section 8 we present a *modal proposition logic* with modalities $\Box_a$ for all actions a, and give some connections to process algebra with conditions.

Another motivation to study frames in the setting of process algebra is bisimulation geared. We give a short explanation. The nesting operation $^\sharp$, introduced in Bergstra et al. 1994, and defined by

$$x^\sharp y = x(x^\sharp y)x + y$$

is crucial for this motivation. Ongoing research pointed out that each recursively enumerable frame can be expressed up to weak bisimulation in process algebra with $^\sharp$ and *. Here * is the binary Kleene star, discussed in Section 3. Currently, the "process-part" of this result is clear-cut (based on the process algebraic representation of Turing machines). The "frame-part" of this result is based on notions that are not widely spread. We hope that our present papers on frames (this one, and Bergstra and Ponse 1994) will help to make these notions common property. In Section 9 we return to this issue. (For other expressivity results, see e.g. Baeten et al. 1987, Vaandrager 1993, Bergstra et al. 1994.)

2 Frames and Frame Polynomials

Let the symbol $\mathbb{N}$ represent the naturals given by constant 0 and successor function S (and equipped with equality predicate $=$). As usual, we represent the elements of $\mathbb{N}$ as numerals 0, 1, 2, 3, ..., and we use meta-variables k, l, m, n for these.

Let $\mathbb{S}$ be a set of states, obtained by an embedding $i_\mathbb{N}$ of $\mathbb{N}$ in $\mathbb{S}$ and a pairing function $\rangle\!\!-\!\!\langle : \mathbb{S}^2 \to \mathbb{S}$. We further abbreviate $i_\mathbb{N}(n)$ by n.

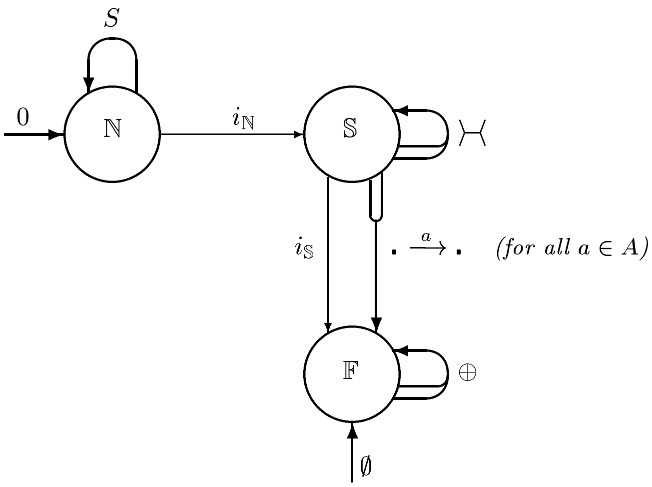
Let A be a finite set of *action symbols* or *labels*. We define the signature of the set $\mathbb{F}$ of *(simple) frames* by

1. An embedding $i_\mathbb{S}$ of $\mathbb{S}$ in $\mathbb{F}$;
2. Binary operations $\cdot \xrightarrow{a} \cdot : \mathbb{S}^2 \to \mathbb{F}$ for all $a \in A$, so called *transitions* having a *label* in A;
3. The empty frame $\emptyset \in \mathbb{F}$;
4. The binary frame operation $\oplus$, *alternative frame composition* or *frame union*.

We further write n instead of $i_\mathbb{S}(n)$ or $i_\mathbb{N}(i_\mathbb{S}(n))$, so e.g., $0\rangle\!\!-\!\!\langle 1$ is considered a frame. The signature introduced above can be depicted as follows:

TABLE 1 Axioms for frames.

(FA1)	$X \oplus Y$	$=$	$Y \oplus X$
(FA2)	$X \oplus (Y \oplus Z)$	$=$	$(X \oplus Y) \oplus Z$
(FA3)	$X \oplus X$	$=$	X
(FA4)	$X \oplus \emptyset$	$=$	X
(FA5)	$s \oplus (s \xrightarrow{a} s')$	$=$	$s \xrightarrow{a} s'$
(FA6)	$s' \oplus (s \xrightarrow{a} s')$	$=$	$s \xrightarrow{a} s'$



The constant $\emptyset$ denotes the empty frame, containing neither states nor transitions. The operation $\oplus$ on frames gives the union of the states and transitions of both its arguments, and is defined in Table 1 (see also Bergstra and Ponse 1994). For $s, s' \in \mathbb{S}$, frames of the form s or $s \xrightarrow{a} s'$ are called *atomic*. The set of states of frame F is denoted by $|F| \subseteq \mathbb{S}$. The purpose of the pairing function $\rangle\!\!-\!\!\langle$ is twofold:

1. It provides a simple means to define "fresh" states to be used for the construction of frames, e.g., $\{0 \rangle\!\!-\!\!\langle s \mid s \in |F|\}$ is a fresh copy of $|F|$.
2. In Bergstra and Ponse 1994 it is used to define "concurrent frame composition".

We define *iterated* alternative composition by

$$\bigoplus_{i=n}^{k} F_i = \begin{cases} \emptyset & \text{if } k < n, \\ F_n \oplus \bigoplus_{i=n+1}^{k} F_i & \text{otherwise.} \end{cases}$$

Now each frame has a representation of the form $\bigoplus_{i=1}^{m} F_i$, where the F_i are atomic. Below we prove this property for a generalized syntax for frame representation.

Let V be a countable infinity of variables $x, y, \ldots$ (possibly primed or sub-

TABLE 2 Additional axioms for frame polynomials, $x, y \in V$.

(FP1)	$\bigoplus_x F$	$=$	F	provided x does not occur in F
(FP2)	$\bigoplus_y F$	$=$	$\bigoplus_x F[x/y]$	provided x does not occur in F
(FP3)	$\bigoplus_x \bigoplus_y F$	$=$	$\bigoplus_y \bigoplus_x F$	
(FPA1)	$\bigoplus_x (F \oplus F')$	$=$	$\bigoplus_x (F) \oplus \bigoplus_x (F')$	
(FPA2)	$\bigoplus_x F(x)$	$=$	$F[0/x] \oplus \bigoplus_x F[S(x)/x]$	

scripted) over $\mathbb{N}$. In the following we define *frame polynomials* over $V, \mathbb{S}$ and A of which a simple example is x for some $x \in V$. Let $\mathbb{S}[V]$ be defined as the set obtained by closure under the function $\rangle\!\!-\!\!\langle$ on $(\mathbb{S} \cup V)^2$, and let $i_{\mathbb{S}}$ be appropriately extended. Quantification over states can be expressed with the *generalized frame sum* $\bigoplus_x$, defined by

$$\bigoplus_x F(x) = F[0/x] \oplus F[1/x] \oplus F[2/x] \oplus \cdots$$

where $F[n/x]$ denotes the substitution of n for x in F. *Frame polynomials* over $V, \mathbb{S}$ and A are formally defined by the following BNF grammar with $x \in V$, $s, s' \in \mathbb{S}[V]$ and $a \in A$.

$$F \ ::= \ \emptyset \ | \ s \ | \ (s \xrightarrow{a} s') \ | \ F \oplus F \ | \ \bigoplus_x (F)$$

Let $\mathrm{FV}(F)$ be the set of free variables in polynomial F (variables not bound by a $\bigoplus_x$ application). A frame polynomial is *closed* if $\mathrm{FV}(F) = \emptyset$. A closed frame polynomial will be further referred to as a *frame*. So from now on we consider frames that may have an infinite number of states and/or transitions, and $\mathbb{F}$ now denotes this set.

Frame polynomials can also be equated. An example of an obvious identity is

$$\bigoplus_x (x \xrightarrow{a} S(x)) = (0 \xrightarrow{a} 1) \oplus \bigoplus_x (S(x) \xrightarrow{a} S^2(x)).$$

In order to formalize reasoning about equality between (closed) frame polynomials, we give the axioms in Table 1 with the s, s' now ranging over $\mathbb{S}[V]$, and the axioms in Table 2. In this latter table, the proviso

"x does not occur in F".

means that variable x occurs neither free, nor bounded in F. Note that bounded variables may always be renamed (FP2).

For convenience, we shall often write $\bigoplus_{x_1,\ldots,x_n}$ instead of $\bigoplus_{x_1} \circ \cdots \circ \bigoplus_{x_n}$.

Definition 2.1. A frame polynomial is in *normal form* if it is of the form

$$\bigoplus_{i=1}^k \bigoplus_{x_1,\ldots,x_l} F_i$$

with all F_i *atomic*, i.e. each F_i is either of the form s, or $s \xrightarrow{a} s'$ for some $s, s' \in \mathbb{S}[V]$ and $a \in A$.

The following representation property of frames follows easily by structural induction.

Lemma 2.2. *Each (infinite) frame F can be represented by a polynomial in normal form.* □

TABLE 3 The axiom system $\mathrm{BPA}_\delta(A)$.

(A1)	$x+y$	$=$	$y+x$
(A2)	$(x+y)+z$	$=$	$x+(y+z)$
(A3)	$x+x$	$=$	x
(A4)	$(x+y)z$	$=$	$xz+yz$
(A5)	$(xy)z$	$=$	$x(yz)$
(A6)	$x+\delta$	$=$	x
(A7)	δx	$=$	δ

3 Process Algebra over Frames

We shortly introduce "process algebra", or more precisely, that part of ACP that plays a role in this section. For an overview, in which many more references can be found, we refer to Baeten and Weijland 1990. We consider processes with syntax

$$P \ ::= \ P+P \ \mid \ P\cdot P \ \mid \ P^*P \ \mid \ \delta \ \mid \ a\in A.$$

Let $\mathcal{P}$ be the set of processes. We adopt the conventions that $+$ (sum or choice) binds weakest, and $\cdot$ (product or sequential composition) binds strongest of all operations, and to leave out the symbol $\cdot$ in a product. The operation *, the *Binary Kleene Star* (Kleene 1956, Bergstra et al. 1994), has as its defining axiom

$$\text{(BKS1)} \qquad x^*y = x(x^*y)+y$$

and is relatively new in process algebra (but see Milner 1984, where a unary * is discussed in the setting of regular behaviours). The constant δ is not an element of A (so no frame has a transition labeled with δ) and is called *deadlock* or *inaction*. For example, $a\cdot\delta$ is a process that has no (successful) termination and is stuck after executing a.

Besides the axiom for *, other process algebra axioms that shall be used in this section are those of $\mathrm{BPA}_\delta(A)$ (Basic Process Algebra with δ), displayed in Table 3. We write $\mathrm{BPA}^*_\delta(A)$ if BKS1 is included. A further rule on *-processes is

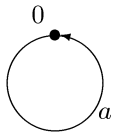
$$(\mathrm{RSP}^*) \qquad x = y\cdot x+z \quad\Longrightarrow\quad x=y^*z.$$

As an example, we derive

$$\begin{aligned}(a+c\cdot\delta)^*b &= (a+c\cdot\delta)\cdot((a+c\cdot\delta)^*b)+b\\ &= a\cdot((a+c\cdot\delta)^*b)+c\cdot\delta\cdot((a+c\cdot\delta)^*b)+b\\ &= a\cdot((a+c\cdot\delta)^*b)+(b+c\cdot\delta).\end{aligned}$$

Hence, with $\mathrm{BPA}^*_\delta(A)$ and RSP^* it follows that $(a+c\cdot\delta)^*b = a^*(b+c\cdot\delta)$.

In the rest of this section we apply process algebraic techniques to prove particular properties of frames. Our method is to "point" a frame, i.e. to distinguish one of its states as a *root* modelling the start of a process, and one as a *tail*, representing (successful) termination of the process. The possible paths that can be traversed, starting from the root, then may determine a certain process term. Various root-tail interpretations are possible. For example, the frame



or, less pictorial $0 \xrightarrow{a} 0$, with (its only) state 0 as root can either be associated with the single process a with tail 0, or with the process a^ω (after each execution of a re-entering the root). This latter process can also be denoted by $a^*\delta$. For each view, we introduce different operations: $r^\frown t$ where r is the root and t the tail in some frame; and operations $s^{\circlearrowleft}$ with s the root and no tail, that capture non-terminating behaviour. Typical identities are:

$$0^\frown 0(0 \xrightarrow{a} 0) = a \quad \text{and} \quad 0^{\circlearrowleft}(0 \xrightarrow{a} 0) = a^*\delta.$$

A simple way to express that a transition $s \xrightarrow{a} s'$ is part of a frame F is

$$F = F \oplus (s \xrightarrow{a} s') \quad \text{or shortly} \quad s \xrightarrow{a} s' \in_{\mathbb{F}} F.$$

Because equality is decidable for the class of frames under consideration (proved in Bergstra and Ponse 1994), this notion is well-defined.

We first define the *process extraction* operation

$${}^\frown : \mathbb{S}^2 \to (\mathbb{F} \to \mathcal{P})$$

for the root-tail case as follows:

$$s^\frown tF = \sum_{a \in A} \left(\begin{array}{c} (s \xrightarrow{a} t \in_{\mathbb{F}} F) :\rightarrow a \,+ \\ \sum_{r \in |F| \setminus \{t\}} ((s \xrightarrow{a} r \in_{\mathbb{F}} F) :\rightarrow a \cdot (r^\frown tF)) \end{array} \right)$$

where $s \xrightarrow{a} t \in_{\mathbb{F}} F$ takes the value t or f ("true" or "false"), and the expression $:\rightarrow$ denotes the "guarded command" (see Dijkstra 1976):

$$\mathsf{t} :\rightarrow x = x$$

$$\mathsf{f} :\rightarrow x = \delta.$$

Furthermore, empty sums equal δ and we take $\sum_{i \in I} \delta = \delta$ for I infinite. Note that the process extraction operation never returns processes of the form $(x + y)z$ (cf. axiom A4). Some examples:

$$\begin{array}{rcl} 0^\frown 1(0 \xrightarrow{a} 0) & = & a \cdot 0^\frown 1(0 \xrightarrow{a} 0) \\ & = & a \cdot a \cdot 0^\frown 1(0 \xrightarrow{a} 0) \\ & = & \ldots.. \end{array}$$

and thus undefined so far, and

$$\begin{array}{rcl} 0^\frown 0(0 \xrightarrow{a} 1) & = & a \cdot 1^\frown 0(0 \xrightarrow{a} 1) \\ & = & a\delta. \end{array}$$

TABLE 4 Rules for reachability in a frame F.

$$s \xrightarrow{a} s' \in_{\mathbb{F}} F \quad \Longrightarrow \quad F \models s \longrightarrow s'$$

$$\left.\begin{array}{l} F \models s \longrightarrow s'' \\ F \models s'' \longrightarrow s' \end{array}\right\} \Longrightarrow \quad F \models s \longrightarrow s'$$

The partiality of $^\frown$ with respect to $\mathrm{BPA}_\delta(A)$, illustrated in the first example above, can also be caused by the fact that $\sum_{i\in I} x_i$ only is a process term if I is finite, namely for $I = \{i_0, i_1, ..., i_n\}$ a term equivalent to

$$x_{i_0} + x_{i_1} + \cdots + x_{i_n}.$$

So frames having a state with an infinite out-degree may not be "pointable" to a process term. However, process extraction can be extended uniquely to a total operation if we replace $\mathcal{P}$ by a process algebra with infinite sums and unique solutions for guarded, linear equations.

Next we extract * (Binary Kleene Star) from pointed frames. We provide two ways of doing so, one for processes that can terminate, and a second one for non-terminating processes. To this end, we define a notion of *reachability* that extends connectedness of two states by a single transition: $F \models s \longrightarrow s'$ if there is a path in F from s to s'. This notion is defined in Table 4. We write $F \not\models s \longrightarrow s'$ if s' is *not* reachable from s in F. In this way one can express that a state s is not part of *some* loop: $F \not\models s \longrightarrow s$. For extraction of *-processes, we have the following two rules:

$$\text{(FBKS1)} \qquad \frac{|X| \cap |Y| = \{s\} \qquad t \notin |X| \qquad Y \not\models s \longrightarrow s}{s^\frown t(X \oplus Y) = (s^\frown sX)^*(s^\frown tY)}$$

$$\text{(FBKS2)} \qquad s^{\circlearrowleft} X = (s^\frown sX)^*\delta.$$

In some cases, the rule FBKS1 can be applied in different ways. For example, if

$$F = (0 \xrightarrow{a} 0) \oplus (0 \xrightarrow{b} 1) \oplus (0 \xrightarrow{c} 2),$$

then decomposing F in the union of $(0 \xrightarrow{a} 0) \oplus (0 \xrightarrow{c} 2)$ and $(0 \xrightarrow{b} 1)$ yields

$$\begin{aligned} 0^\frown 1F &= 0^\frown 0((0 \xrightarrow{a} 0) \oplus (0 \xrightarrow{c} 2))^* 0^\frown 1(0 \xrightarrow{b} 1) \\ &= (a + c \cdot 2^\frown 0((0 \xrightarrow{a} 0) \oplus (0 \xrightarrow{c} 2)))^* b \\ &= (a + c \cdot \delta)^* b. \end{aligned}$$

Splitting up F into the union of $(0 \xrightarrow{a} 0)$ and $(0 \xrightarrow{b} 1) \oplus (0 \xrightarrow{c} 2)$ gives with FBKS1 that

$$\begin{aligned} 0^\frown 1F &= 0^\frown 0(0 \xrightarrow{a} 0)^* 0^\frown 1((0 \xrightarrow{b} 1) \oplus (0 \xrightarrow{c} 2)) \\ &= a^*(b + c \cdot 2^\frown 1((0 \xrightarrow{b} 1) \oplus (0 \xrightarrow{c} 2)) \\ &= a^*(b + c \cdot \delta). \end{aligned}$$

However, as showed above, both these *-processes are equal.

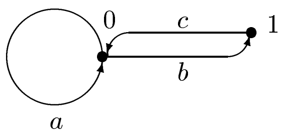
The first of the examples above can now be computed with rule FBKS1:

$$\begin{aligned}
0^\frown 1(0 \xrightarrow{a} 0) &= 0^\frown 1((0 \xrightarrow{a} 0) \oplus 0)\\
&= (0^\frown 0(0 \xrightarrow{a} 0))^* (0^\frown 1(0))\\
&= a^* \delta.
\end{aligned}$$

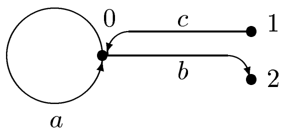
One can prove $0^{\circlearrowleft}(0 \xrightarrow{a} 1) = a\delta$ with the second identity FBKS2:

$$\begin{aligned}
0^{\circlearrowleft}(0 \xrightarrow{a} 1) &= (0^\frown 0(0 \xrightarrow{a} 1))^* \delta\\
&= (a \cdot (1^\frown 0(0 \xrightarrow{a} 1)))^* \delta\\
&= (a\delta)^* \delta\\
&= (a\delta) \cdot ((a\delta)^* \delta) + \delta\\
&= a\delta.
\end{aligned}$$

We further define two auxiliary operations on frames. The purpose of these operations is to unfold a pointed frame in such a way that FBKS1 and FBKS2 become applicable (one can imagine more such operations and rules, especially for extracting $^\sharp$ from pointed frames). The first operation is *tail-unwinding*. The idea is that a frame can be provided with a fresh tail. A (perhaps too simple) motivation for this operator is the following. Consider the frame F defined by:



Clearly, $0^\frown 1F = a \cdot (0^\frown 1F) + b$. Hence, by RSP* we find $0^\frown 1F = a^* b$. However, we cannot apply FBKS1 on this frame because of the c-transition. On infinite frames, RSP* need not so easily be applicable, if at all. Therefore we apply a tail-unwinding on F, making FBKS1 applicable, by adding a fresh tail 2 for the b-transition:



Call the resulting frame F'. Then both $0^\frown 2F' = a \cdot (0^\frown 2F') + b$ and $0^\frown 2F' \stackrel{\text{FBKS1}}{=} a^* b$.

We define tail-unwinding by the operation

$tu : (\mathbb{S} \times \mathbb{S} \times \mathbb{F}) \to \mathbb{F}$, where $tu(s,t,F)$ stands for "replace t by s in F", in an inductive manner:

$$tu(s,t,\emptyset) = \emptyset$$

$$tu(s,t,s') = s'$$

$$tu(s,t,s' \xrightarrow{a} t') = \begin{cases} (s' \xrightarrow{a} s) \oplus t' & \text{if } t = t' \\ s' \xrightarrow{a} t' & \text{otherwise} \end{cases}$$

$$tu(s,t,X \oplus Y) = tu(s,t,X) \oplus tu(s,t,Y)$$

(the summand $.. \oplus t'$ in the definition of $tu(s,t,s' \xrightarrow{a} t')$ originates from the axiom $s' \xrightarrow{a} t' = (s' \xrightarrow{a} t') \oplus t'$, see Table 1).
As to frames defined by (closed) polynomials, the definition of tu is little more involved. By Lemma 2.2 it is sufficient to define tu for closed polynomials in normal form (see Definition 2.1).
For $F = \bigoplus_{x_1,\dots,x_l} s'$ we simply define

$$tu(s,t,F) = F.$$

For $F = \bigoplus_{x_1,\dots,x_l} s' \xrightarrow{a} t'$ we distinguish two cases:

1. $t \notin |\bigoplus_{x_1,\dots,x_l} t'|$. In this case $tu(s,t,F) = F$.
2. $t \in |\bigoplus_{x_1,\dots,x_l} t'|$. We decompose F into a number of instantiations, and a rest-polynomial in which t does not occur. Assume t contains k occurrences of the successor function S. With the axioms in Table 2 we can decompose F into the union of the $(k+1)^l$ instantiations of $s' \xrightarrow{a} t'$ in which each variable x_i has one of the instantiations $0, \dots, k$, and the rest-polynomial
$$\bigoplus_{x_1,\dots,x_l} s'[S^{k+1}(x_i)/x_i] \xrightarrow{a} t'[S^{k+1}(x_i)/x_i]$$
where $[S^{k+1}(x_i)/x_i]$ abbreviates application of the l substitutions
$$[S^{k+1}(x_0)/x_0] \dots [S^{k+1}(x_l)/x_l].$$
Clearly, t cannot be an element of this rest-polynomial, so that $tu(s,t,.)$ is defined on it by 1 above. As the $(k+1)^l$ instantiations are non-polynomial, $tu(s,t,.)$ also is defined on these.

An example:

$$\begin{aligned} & tu(0\rangle\!\!-\!\!\langle 0, 0, \textstyle\bigoplus_z (S(z) \xrightarrow{a} z)) \\ = \; & tu(0\rangle\!\!-\!\!\langle 0, 0, 1 \xrightarrow{a} 0) \oplus tu(0\rangle\!\!-\!\!\langle 0, 0, \textstyle\bigoplus_z (S(S(z)) \xrightarrow{a} S(z))) \\ = \; & (1 \xrightarrow{a} 0)\rangle\!\!-\!\!\langle(0) \oplus 0 \oplus \textstyle\bigoplus_z (S(S(z)) \xrightarrow{a} S(z)). \end{aligned}$$

The second operation that we shall use for frame transformation is the extension of the successor function S to frame polynomials ($s,t \in \mathbb{S}[V]$):

$$S(\emptyset) = \emptyset$$

$$S(s\rangle\!\!-\!\!\langle t) = S(s)\rangle\!\!-\!\!\langle S(t)$$

$$S(s \xrightarrow{a} t) = S(s) \xrightarrow{a} S(t)$$

$$S(X \oplus Y) = S(X) \oplus S(Y)$$

$$S(\textstyle\bigoplus_x (X)) = \textstyle\bigoplus_x (S(X))$$

This operation gives rise to the following law (for $s, t \in \mathbb{S}$):

$$s^\frown tX = S(s)^\frown S(t)\ S(X).$$

Validity is trivial: this law only concerns a "global state renaming".

4 Analyzing Counters

Employing tail-unwinding and the successor function, we can prove some properties of pointed frames with process algebraic techniques. Consider the frames F and G defined by

$$F = \bigoplus_x ((x \xrightarrow{a} S(x)) \oplus (S(x) \xrightarrow{b} x))$$

$$G = F \oplus (0 \xrightarrow{stop} 0)\!\!-\!\!\!\!-\!\!(0).$$

These frames can be depicted as in Figure 1.

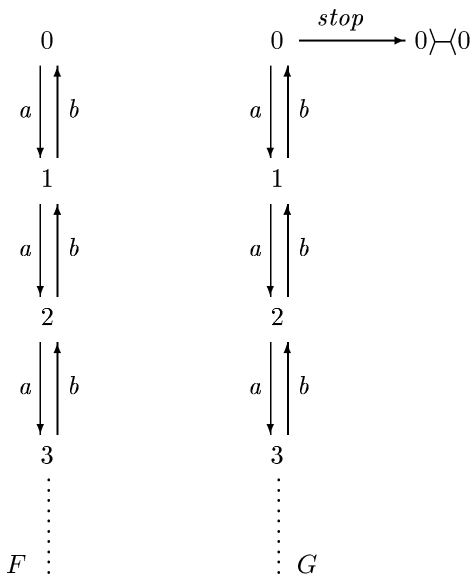


FIGURE 1 Two frames.

Two general types of *counters* that are often considered in the setting of ACP are pointed versions of the frames F and G. The counter $0^{\circlearrowleft}F$ is a perpetual one with root state 0, while the counter $0^\frown(0)\!\!-\!\!\!\!-\!\!(0)G$ can terminate from its root 0 by an action *stop* to its tail $0)\!\!-\!\!\!\!-\!\!(0$. We shall prove that both these counters indeed satisfy standard recursive specifications in process algebra. To this end, we first show

$$0^\frown 0F = a \cdot ((0^\frown 0F)^* b)$$

(an auxiliary statement used in both proofs).

$$
\begin{array}{ll}
\multicolumn{2}{l}{0^\frown 0F} \\
= & 0^\frown (0)\succ\!\prec(0)\,tu(0)\succ\!\prec(0,0,F) \\
= & 0^\frown (0)\succ\!\prec(0)\,tu(0)\succ\!\prec(0,0,S(F) \oplus (0 \xrightarrow{a} 1) \oplus (1 \xrightarrow{b} 0)) \\
= & 0^\frown (0)\succ\!\prec(0)(S(F) \oplus [(0 \xrightarrow{a} 1) \oplus (1 \xrightarrow{b} 0)\succ\!\prec(0) \oplus 0]) \\
= & a \cdot (1^\frown (0)\succ\!\prec(0)(S(F) \oplus [(0 \xrightarrow{a} 1) \oplus (1 \xrightarrow{b} 0)\succ\!\prec(0)])) \\
\stackrel{(*)}{=} & a \cdot ((1^\frown 1S(F))^*(1^\frown (0)\succ\!\prec(0)((0 \xrightarrow{a} 1) \oplus (1 \xrightarrow{b} 0)\succ\!\prec(0)))) \\
\stackrel{(**)}{=} & a \cdot ((0^\frown 0F)^* b)
\end{array}
$$

(where $\stackrel{(*)}{=}$ follows by FBKS1, and $\stackrel{(**)}{=}$ by the successor-law mentioned in the previous section).

The following recursive specification defines a *non-terminating* counter NC in process algebra:

$$
\begin{array}{rcl}
NC & = & X^*\delta \\
X & = & aY \\
Y & = & aYY + b.
\end{array}
$$

Theorem 4.1. *Let $F = \bigoplus_x((x \xrightarrow{a} S(x)) \oplus (S(x) \xrightarrow{b} x))$. Then $0^\circlearrowleft F = NC$, where NC is the non-terminating counter as defined above.*

Proof. This follows by the Recursive Specification Principle RSP (see e.g., Baeten and Weijland 1990). The rule RSP implies that each (recursively, properly) defined process that satisfies the equation for NC, is equal to NC.[1] So $0^\circlearrowleft F = NC$ if $X = 0^\frown 0F$. Above we showed $0^\frown 0F = a \cdot ((0^\frown 0F)^* b)$. So proving that $(0^\frown 0F)^* b$ solves the equation for Y finishes the proof. This follows straightforward:

$$
\begin{array}{rcl}
(0^\frown 0F)^* b & = & (0^\frown 0F) \cdot ((0^\frown 0F)^* b) + b \\
& = & a \cdot ((0^\frown 0F)^* b) \cdot ((0^\frown 0F)^* b) + b.
\end{array}
$$

□

Secondly, a standard specification of a *terminating* counter TC in process algebra is

$$
TC \;=\; X^*\mathit{stop}
$$

with X (and Y) specified as above.

Theorem 4.2. *Let $G = F \oplus (0 \xrightarrow{stop} 0)\succ\!\prec(0)$ and $F = \bigoplus_x((x \xrightarrow{a} S(x)) \oplus (S(x) \xrightarrow{b} x))$. Then $0^\frown (0)\succ\!\prec(0)G = TC$, where TC is the terminating counter as defined above.*

[1] This is the case because the specification of NC is "guarded"; the unguarded specification $X = X$ certainly does not determine a process.

TABLE 5 A proof system for propositional logic.

(P1)	$\phi \to (\psi \to \phi)$
(P2)	$((\phi \to (\psi \to \xi)) \to ((\phi \to \psi) \to (\phi \to \xi)))$
(P3)	$((\neg\phi \to \neg\psi) \to (\psi \to \phi))$
(P4)	$\mathsf{t} \leftrightarrow (p \to p)$
(P5)	$\mathsf{f} \leftrightarrow \neg\mathsf{t}$
(MP)	$\dfrac{\phi \qquad (\phi \to \psi)}{\psi}$

Proof. We derive

$$\begin{array}{rcl} 0^\frown(0)\!\succ\!\!\prec\!(0)G & \overset{\text{FBKS1}}{=} & (0^\frown 0F)^*(0^\frown(0)\!\succ\!\!\prec\!(0)(0 \xrightarrow{stop} 0)\!\succ\!\!\prec\!(0)) \\ & = & (0^\frown 0F)^*\mathit{stop}. \end{array}$$

According to Theorem 4.1, $X = (0^\frown 0F)$, and hence $0^\frown(0)\!\succ\!\!\prec\!(0)G = TC$. □

Different counters can be algebraically analyzed in this way, eg., versions that have at their root (or at all states) also the possibility to perform non-counting, looping actions $n \xrightarrow{loop} n$. The general idea is to unfold a pointed frame using tail-unwinding and successor function into subframes on which rules such as FBKS1 can be applied (one can define more such rules, e.g., for extracting $^\sharp$ (see Introduction) from a pointed frame).

5 Frames with Signal Insertion

In the sequel of the paper, we consider frames over an extended signature, so called *signal inserted* frames. A similar notion occurred earlier in Baeten and Bergstra 1992. In Baeten and Bergstra 1994, an additional requirement is made: signals are propositions. The starting point of view in that paper can be paraphrased by "the visible part (signal) of a process is a proposition". In this way, the conditions in conditional expressions (discussed in the next section) such as the ϕ in 'if—ϕ—then—P—fi' are complementary to signals, and the latter provide whether conditions can be evaluated, and to what result. In this paper, we also stick to propositions as signals, but—as remarked in Baeten and Bergstra 1994—one can imagine many modifications, such as 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.

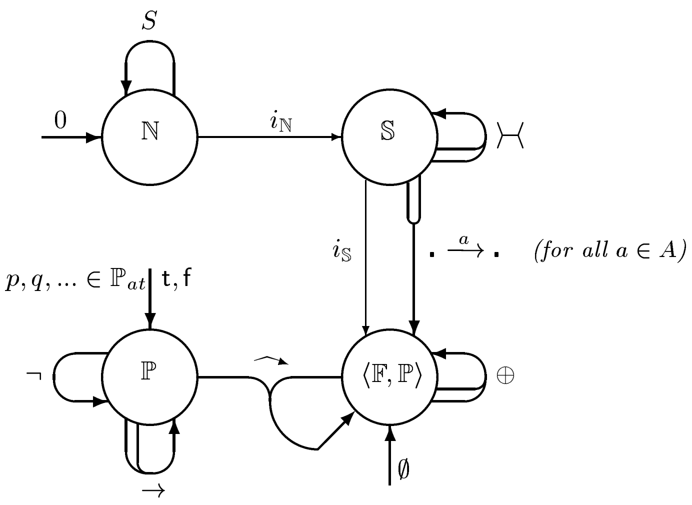
Let the symbol $\mathbb{P}$ represent propositions, constructed from *atomic propositions* $\mathbb{P}_{at}$, denoted by $p, q, r, ...$, the connectives $\neg, \rightarrow$, and t ("true") and f ("false"). Let $\phi, \psi, \ldots$ range over propositions. As usual, we write $\phi \vee \psi$ for $\neg\phi \to \psi$, $\phi \wedge \psi$ for $\neg(\neg\phi \vee \neg\psi)$, and $\phi \leftrightarrow \psi$ for $(P \to \psi) \wedge (\psi \to \phi)$. In Table 5 we give a common propositional calculus that is complete.

We define the signature of the set $\langle\mathbb{F}, \mathbb{P}\rangle$ of *signal inserted frames* by

1. An embedding $i_{\mathbb{S}}$ of $\mathbb{S}$ in $\langle\mathbb{F}, \mathbb{P}\rangle$;

2. Binary operations $\cdot \xrightarrow{a} \cdot : \mathbb{S}^2 \to \langle \mathbb{F}, \mathbb{P} \rangle$ for all $a \in A$, *transitions* with a *label* in A;
3. The *signal insertion operation* $\curvearrowright : \mathbb{P} \times \langle \mathbb{F}, \mathbb{P} \rangle \to \langle \mathbb{F}, \mathbb{P} \rangle$;
4. The empty frame $\emptyset \in \langle \mathbb{F}, \mathbb{P} \rangle$;
5. The binary frame operation $\oplus$, *alternative frame composition* or *frame union*.

Recall that A is a finite set of *action symbols* or *labels*. The signature of signal inserted frames can be depicted as follows:



The operation $\curvearrowright$ is defined in Table 6, where Ins4–8 characterize specific combinations with $\oplus$. This latter operation still gives the union of the states of both arguments, but not in each case that of the transitions. Signal inserted states equivalent to $\mathsf{f} \curvearrowright s$ absorb all in-going and out-going transitions from s (see Ins7–8). The remaining axioms for signal inserted frames are those given in Table 1 (see Section 2). Recall the notation $s \xrightarrow{a} s' \in_{\mathbb{F}} F$ (defined by $(s \xrightarrow{a} s') \oplus F = F$) and consider

$$F = (s \xrightarrow{a} s') \oplus \mathsf{f} \curvearrowright s \oplus \mathsf{f} \curvearrowright s'.$$

Then the transition $s \xrightarrow{a} s'$ actually is not a part of this frame. So this notation does *not* apply to signal inserted frames.

Frame polynomials may now include signals. Recall that $\mathbb{S}[V]$ is defined as the set obtained by closure under the function $\succ\!\prec$ on $(\mathbb{S} \cup V)^2$, and $i_{\mathbb{S}}$ is appropriately extended. Frame polynomials over $V, \mathbb{S}, A$ and $\mathbb{P}$ take the following syntax for $x \in V,\ s, s' \in \mathbb{S}[V],\ a \in A$ and $\phi \in \mathbb{P}$:

$$F \ ::= \ \emptyset \ \mid \ s \ \mid \ (s \xrightarrow{a} s') \ \mid \ \bigoplus_x(F) \ \mid \ \phi \curvearrowright F \ \mid \ F \oplus F.$$

Frame polynomials with signal insertion satisfy the axioms in Tables 1, 2 and 6 with the s, s' now ranging over $\mathbb{S}[V]$. The following representation property follows easily by structural induction (cf. Lemma 2.2).

TABLE 6 Axioms for signal insertion.

(Ins1)	$\phi \curvearrowright \emptyset$	$=$	$\emptyset$
(Ins2)	$\mathsf{t} \curvearrowright X$	$=$	X
(Ins3)	$\phi \curvearrowright (\psi \curvearrowright X)$	$=$	$(\phi \wedge \psi) \curvearrowright X$
(Ins4)	$(\phi \curvearrowright X) \oplus (\psi \curvearrowright X)$	$=$	$(\phi \wedge \psi) \curvearrowright X$
(Ins5)	$\phi \curvearrowright (X \oplus Y)$	$=$	$(\phi \curvearrowright X) \oplus (\phi \curvearrowright Y)$
(Ins6)	$\phi \curvearrowright (s \xrightarrow{a} s')$	$=$	$(\phi \curvearrowright s) \oplus (s \xrightarrow{a} s') \oplus (\phi \curvearrowright s')$
(Ins7)	$(\mathsf{f} \curvearrowright s) \oplus (s \xrightarrow{a} s')$	$=$	$(\mathsf{f} \curvearrowright s) \oplus s'$
(Ins8)	$(s \xrightarrow{a} s') \oplus (\mathsf{f} \curvearrowright s')$	$=$	$s \oplus (\mathsf{f} \curvearrowright s')$

Lemma 5.1. *Each (infinite) frame F can be represented by a polynomial of the form*

$$\bigoplus_{i=1}^{k} \bigoplus_{x_1,\ldots,x_l} F_i$$

with all F_i either of the form $\phi \curvearrowright s$, or $s \xrightarrow{a} s'$ for some $s, s' \in \mathbb{S}[V]$ and $a \in A$. □

It is useful to have an operation that extracts the propositions inserted on a frame F relative to a state s: the operation $Ext(F, s)$ defined in Table 7 does the job. With this operation we can define whenever a signal inserted frame F models a proposition ϕ, in symbols $F \models \phi$ (recall that $|F| \subseteq \mathbb{S}$ denote the set of states of a (signal inserted) frame F):

$$F \models \phi \iff \forall s \in |F| (F, s \models \phi)$$

$$F, s \models \phi \iff Ext(F, s) \models \phi$$

where $\psi \models \phi$ is defined as usual.

TABLE 7 Axioms for proposition extraction.

(PE1)	$Ext(\emptyset, s)$	$=$	t
(PE2)	$Ext(s', s)$	$=$	t
(PE3)	$Ext(s' \xrightarrow{a} s'', s)$	$=$	t
(PE4)	$Ext(X \oplus Y, s)$	$=$	$Ext(X, s) \wedge Ext(Y, s)$
(PE5)	$Ext(\phi \curvearrowright s', s)$	$=$	$\begin{cases} \phi & \text{if } s' = s \\ \mathsf{t} & \text{otherwise} \end{cases}$

TABLE 8 Axioms for conditional processes.

(Con1)	$x \lhd \phi \rhd y$	$=$	$\phi :\rightarrow x + \neg\phi :\rightarrow y$
(Con2)	$\mathsf{t} :\rightarrow x$	$=$	x
(Con3)	$\mathsf{f} :\rightarrow x$	$=$	δ
(Con4)	$\phi :\rightarrow \delta$	$=$	δ
(Con5)	$\phi :\rightarrow (x+y)$	$=$	$\phi :\rightarrow x + \phi :\rightarrow y$
(Con6)	$(\phi :\rightarrow x)y$	$=$	$\phi :\rightarrow xy$
(Con7)	$(\phi \vee \psi) :\rightarrow x$	$=$	$\phi :\rightarrow x + \psi :\rightarrow x$
(Con8)	$\phi :\rightarrow (\psi :\rightarrow x)$	$=$	$(\phi \wedge \psi) :\rightarrow x$

6 Process Algebra with Conditions

In this section we introduce "conditional processes", parameterized over $\mathbb{P}$. We fist give syntax for such processes ($\phi \in \mathbb{P}$):

$$\begin{aligned} P \quad ::= \quad & P + P \mid P \cdot P \mid P^* P \mid \\ & P \lhd \phi \rhd P \mid \phi :\rightarrow P \mid \delta \mid a \in A. \end{aligned}$$

We adopt the syntactical conventions introduced in Section 3: in particular $+$ binds weakest, and $\cdot$ binds strongest of all operations considered. The operation $:\rightarrow$ denotes the guarded command or "if—then—fi" (see Dijkstra 1976, in Section 3 we already used this operation). The operation $. \lhd . \rhd .$ represents "then—if—else—neht" and is introduced by Hoare et al. 1987. Both operations are axiomatized in Table 8 (here we do not stick to conventional axiom names). Further references to process algebra with conditions are Baeten and Bergstra 1992, Groote and Ponse 1994.

Next we define *reachability of processes* in frames: $F \models s \xrightarrow{P} s'$ if process P has a path in F from s to s'. In Table 9 we define $F \models s \xrightarrow{P} s'$ in an inductive manner, where the auxiliary notation $x^n \cdot y$ for $n \in \mathbb{N}$ stands for:

$$x^0 \cdot y \stackrel{\text{def}}{=} y, \quad x^{n+1} \cdot y \stackrel{\text{def}}{=} x \cdot (x^n \cdot y).$$

The following soundness result follows straightforward.

Theorem 6.1. *If* $\mathrm{BPA}^*_\delta(A) + \mathrm{RSP}^* + \mathrm{Con1\text{-}8} \vdash P = Q$, *then*

$$F \models s \xrightarrow{P} s' \iff F \models s \xrightarrow{Q} s'.$$

We do not define process extraction for conditional processes and signal inserted frames. This topic is quite involved, and raises several questions. It is simpler to follow a reverse approach, and to investigate whether a conditional process has (at least) a path in a certain frame. This brings us to the setting of Floyd Hoare logic, discussed in the next section.

TABLE 9 Reachability of conditional processes in signal inserted frames.

$$\left.\begin{array}{r} F = F \oplus s \xrightarrow{a} s' \\ F, s \not\models \mathsf{f} \text{ and } F, s' \not\models \mathsf{f} \end{array}\right\} \Longrightarrow \quad F \models s \xrightarrow{a} s'$$

$$F \models s \xrightarrow{x} s' \quad \Longrightarrow \quad \left\{\begin{array}{l} F \models s \xrightarrow{x+y} s' \\ F \models s \xrightarrow{y+x} s' \end{array}\right.$$

$$\left.\begin{array}{r} F \models s \xrightarrow{x} s'' \\ F \models s'' \xrightarrow{y} s' \end{array}\right\} \Longrightarrow \quad F \models s \xrightarrow{x \cdot y} s'$$

$$\exists n. F \models s \xrightarrow{x^n \cdot y} s' \quad \Longrightarrow \quad F \models s \xrightarrow{x^* y} s'$$

$$\left.\begin{array}{l} F \models s \xrightarrow{x} s' \\ F, s \models \phi \end{array}\right\} \Longrightarrow \quad F \models s \xrightarrow{x \lhd \phi \rhd y} s'$$

$$\left.\begin{array}{l} F \models s \xrightarrow{y} s' \\ F, s \models \neg\phi \end{array}\right\} \Longrightarrow \quad F \models s \xrightarrow{x \lhd \phi \rhd y} s'$$

$$\left.\begin{array}{l} F \models s \xrightarrow{x} s' \\ F, s \models \phi \end{array}\right\} \Longrightarrow \quad F \models s \xrightarrow{\phi :\rightarrow x} s'.$$

7 Floyd Hoare Logic over Frames

A partial correctness assertion has syntax

$$\{\phi\}P\{\psi\}$$

where ϕ, ψ are *assertions*, and P is a (conditional) process term. A general overview of correctness assertions and their logic is given in Apt 1981. See Ponse 1991, and Groote and Ponse 1994 for a process algebraic approach.

The interpretation of correctness assertions is defined as usual:

$$F \models \{\phi\}P\{\psi\} \text{ if } \forall s \in |F|,\ F, s \models \{\phi\}P\{\psi\}$$

$$F, s \models \{\phi\}P\{\psi\} \text{ if } \forall s' \in |F|$$

$$\left.\begin{array}{l} F, s \models \phi, \text{ and} \\ F \models s \xrightarrow{P} s' \end{array}\right\} \quad \Longrightarrow \quad F, s' \models \psi.$$

Given F, an assertion ξ is a *strongest postcondition* of ϕ and conditional process P if

$$F \models \{\phi\}P\{\xi\}, \text{ and}$$

$$F \models \{\phi\}P\{\psi\} \quad \Longrightarrow \quad F \models \xi \rightarrow \psi.$$

Table 10 Axioms and rules for Floyd-Hoare Logic.

(i)	$\{\phi\}a\{sp(\phi, a)\}$	(ii)	$\{\phi\}\delta\{\psi\}$
(iii)	$\dfrac{\{\phi\}x\{\psi\} \quad \{\phi\}y\{\psi\}}{\{\phi\}x + y\{\psi\}}$	(iv)	$\dfrac{\{\phi\}x\{\psi\} \quad \{\psi\}y\{\xi\}}{\{\phi\}x \cdot y\{\xi\}}$
(v)	$\dfrac{\{\phi\}x\{\phi\} \quad \{\phi\}y\{\psi\}}{\{\phi\}x^*y\{\psi\}}$		
(vi)	$\dfrac{\{\phi \wedge \xi\}x\{\psi\} \quad \{\phi \wedge \neg\xi\}y\{\psi\}}{\{\phi\}x \lhd \xi \rhd y\{\psi\}}$	(vii)	$\dfrac{\{\phi \wedge \xi\}x\{\psi\}}{\{\phi\}\xi :\rightarrow x\{\psi\}}$
$(viii)$	$\dfrac{\phi \rightarrow \phi' \quad \{\phi'\}x\{\psi'\} \quad \psi' \rightarrow \psi}{\{\phi\}x\{\psi\}}$		

An equivalent characterization is:

$$\forall s\ (F, s \models \xi \iff \exists s' \in |F|\ (F, s' \models \phi\ \&\ F \models s' \xrightarrow{P} s)).$$

We say that F is *strongly expressive* if all strongest postconditions can be expressed in $\mathbb{P}$ (cf. Cook and Oppen 1975). In this case the strongest postcondition of ϕ and P is denoted as

$$sp(\phi, P).$$

In Table 10 we give a simple proof system for deriving partial correctness assertions, assuming expressibility of strongest postconditions of actions. Soundness follows straightforward. Furthermore, in case a frame F is strongly expressive, this proof system is complete relative to all implications over $\mathbb{P}$ that are valid in F, say $Th(F)$:

Theorem 7.1. $F \models \{\phi\}P\{\psi\} \iff Th(F) \vdash \{\phi\}P\{\psi\}$.

Proof. Soundness follows by inspection of the rules. Completeness follows easily by induction on the process terms involved. In fact, the only non-trivial inductive step is the one for *.

For a proof, let

$$F \models \{\phi\}P^*Q\{\psi\}.$$

and let proposition ξ be equivalent with the disjunction of ϕ and the strongest postcondition of ϕ and P^*P, i.e.,

$$\xi \leftrightarrow (\phi \vee sp(\phi, P^*P)).$$

We claim that

1. $F \models \{\xi\}P\{\xi\}$,
2. $F \models \{\xi\}Q\{\psi\}$,
3. $F \models \phi \rightarrow \xi$.

With these claims, we establish derivability of $\{\phi\}P^*Q\{\psi\}$: by 1, 2 and induction $\{\xi\}P\{\xi\}$ and $\{\xi\}Q\{\psi\}$ can be derived; with rule (v) one derives $\{\xi\}P^*Q\{\psi\}$; and $\{\phi\}P^*Q\{\psi\}$ follows from 3 and rule $(viii)$.

Proof of claim 1: assume the contrary. Then $\exists s, s' \in |F|$ such that

$$F, s \models \xi,$$

$$F \models s \xrightarrow{P} s'$$

$$F, s' \not\models \xi$$

Now $F, s \models \xi$ holds because one of the following cases:

(a) $F, s \models \phi$, or
(b) $F, s \models sp(\phi, P^*P)$.

We show that both cases contradict $F, s' \not\models \xi$: in case (a) conclude by $F \models s \xrightarrow{P} s'$ that $F \models s \xrightarrow{P^*P} s'$, hence $F, s' \models sp(\phi, P^*P)$ and thus $F, s' \models \xi$. *Contradiction.*

As for case (b), there necessarily exists s'' such that $F, s'' \models \phi$ and $F \models s'' \xrightarrow{P^*P} s$, i.e., for some n, $F \models s'' \xrightarrow{P^{n+1}} s$. So $F \models s'' \xrightarrow{P^{n+2}} s'$, so $F, s' \models sp(\phi, P^*P)$ and thus $F, s' \models \xi$. *Contradiction.*

Claims 2 and 3 can be proved in a similar way. □

8 Modal Proposition Logic

In this section we introduce a simple, modal proposition logic. This version is in between "minimal modal proposition logic", also called K (for an overview of modal logics see Bull and Segerberg 1984) and "propositional dynamic logic", also known as PDL (see e.g. Harel 1984). This is because we consider modalities labeled with elements of A. Our logic strongly resembles the modal logic defined by Hennessy and Milner in Hennessy and Milner 1985 (see also Milner 1989); the difference is that it does not contain infinite conjunctions. We define the language $\mathcal{L}$ of labeled modal proposition logic as follows:

$$\Phi \ ::= \ \Box_a\Phi \ \mid \ \Phi \to \Phi \ \mid \ \neg\Phi \ \mid \ p \in \mathbb{P}_{at}$$

(where $\mathbb{P}_{at}$ was the set of atomic propositions defining $\mathbb{P}$).

The extra clause needed for defining the interpretation of $\mathcal{L}$ is:

$$F, s \models \Box_a\Phi \ \text{ if } \forall s' \in |F| \ (F \models s \xrightarrow{a} s' \quad \Longrightarrow \quad F, s' \models \Phi).$$

We give the following proof system for $\mathcal{L}$, reflecting that the theory of $\mathcal{L}$ just is a labeled version of so called *normal* propositional modal logic.

1. The axioms and rule MP (Modus Ponens) for propositional logic displayed in Table 5;
2. The axiom $(\Box_a(\Phi \to \Psi)) \to (\Box_a\Phi \to \Box_a\Psi)$ ("modal distribution");
3. The rule $\dfrac{\Phi}{\Box_a\Phi}$ ("generalization").

Moreover, this proof system also is equivalent with the restriction of the (complete) proof system for PDL to the syntax of $\mathcal{L}$. Soundness follows immediately by inspection of the axioms and rules. Completeness can be inferred from the completeness result on PDL. The semantical setting of PDL is however not entirely the same as the one provided by signal inserted frames. Whereas in a PDL-frame F the transitions are defined as in our case (roughly speaking), the interpretation of propositions in F is given by a function $Val : \mathbb{P}_{at} \to 2^{|F|}$ (that extends to compound propositions as expected). So a PDL-frame just is a certain signal inserted frame but not vice versa because we allow partial interpretation of propositions. Completeness follows by contraposition in the standard way: starting from an *un*derivable formula Φ one can construct a finite (PDL-)frame that satisfies $\neg\Phi$ (the states of which correspond to conjunctions of subformulas of $\neg\Phi$). Hence Φ is not universally valid. Moreover, universal validity in $\mathcal{L}$ is decidable (given Φ, it is sufficient to select a finite class of frames: those with a number of states exponential in Φ. See further Harel 1984).

One can define extended modalities $\Box_P$ with P a process in the following way:

$$\begin{aligned}
\Box_\delta\Phi &\stackrel{\text{def}}{=} \mathsf{t}\\
\Box_{(x+y)}\Phi &\stackrel{\text{def}}{=} \Box_x\Phi \wedge \Box_y\Phi\\
\Box_{(x\cdot y)}\Phi &\stackrel{\text{def}}{=} \Box_x\Box_y\Phi\\
\Box_{(x\lhd\phi\rhd y)}\Phi &\stackrel{\text{def}}{=} (\phi \to \Box_x\Phi) \wedge (\neg\phi \to \Box_y\Phi)\\
\Box_{(\phi:\to x)}\Phi &\stackrel{\text{def}}{=} \phi \to \Box_x\Phi
\end{aligned}$$

Furthermore, introducing infinite conjunctions (with their obvious interpretation), and A^+ as the set of finite products over A, one can define:

$$\begin{aligned}
\Box_{(x^*y)}\Phi &\stackrel{\text{def}}{=} \textstyle\bigwedge_n \Box_{(x^n\cdot y)}\Phi\\
\Box\Phi &\stackrel{\text{def}}{=} \Phi \wedge \textstyle\bigwedge_{\sigma\in A^+}\Box_\sigma\Phi.
\end{aligned}$$

The left conjunct in the definition of the $\Box$-operation can be left out. The reason for including it is that—as remarked in Groote and van Vlijmen 1995—once we accept the *silent step* τ as a label, reflexivity becomes an irrelevant notion. Note that the last two modalities are not definable in $\mathcal{L}$; adding these (or countable conjunctions instead) gives the modal logic of Hennessy and Milner referred to above, except for the presence of conditionals in our language. Furthermore, observe that a partial correctness assertion $\{\phi\}P\{\psi\}$ is satisfied in F (or in state s of F) iff $F \models \phi \to \Box_P\psi$ ($F, s \models \phi \to \Box_P\psi$, respectively).

In order to establish a connection with process algebra, we let the guards in guarded commands range over $\mathcal{L}$. The extra clause necessary for defining reachability is:

$$\left.\begin{array}{l} F \models s \xrightarrow{x} s' \\ F, s \models \Phi \end{array}\right\} \Longrightarrow F \models s \xrightarrow{\Phi:\to x} s'.$$

So, for example $F \models s \xrightarrow{\Box\Phi:\to P} s'$ states that

- Φ holds in s and in all states reachable from s,
- there is a path $F \models s \xrightarrow{P} s'$, i.e. P has a trace from s to s' in F.

An interesting question concerns the equality

$$\Phi :\to (x \cdot y) = (\Phi :\to x)(\Phi :\to y). \tag{1}$$

This equality certainly is not valid in the world of signal inserted frames. A simple frame refuting $(p :\rightarrow a)(p :\rightarrow b)$ with p an atomic proposition and a an action, is:

$$p \curvearrowright 0 \xrightarrow{a} \neg p \curvearrowright 1 \xrightarrow{b} 2$$

or, more formally, $(0 \xrightarrow{a} 1) \oplus (1 \xrightarrow{b} 2) \oplus p \curvearrowright 0 \oplus \neg p \curvearrowright 1$. It is easily seen that $\forall s, s' \in |F|\ (F \not\models s \xrightarrow{(p:\rightarrow a)(p:\rightarrow b)} s')$. On the other hand we find $F \models 0 \xrightarrow{p:\rightarrow(a\cdot b)} 2$.

Now the following much weaker version of (1) is universally valid:

$$\Box\Phi :\rightarrow (x \cdot y) = (\Box\Phi :\rightarrow x)(\Box\Phi :\rightarrow y).$$

This can be seen as follows: given a signal inserted frame F and $s \in |F|$, define

$$|F, s| \stackrel{\text{def}}{=} \{s\} \ \cup\ \{s' \mid \exists P\ (F \models s \xrightarrow{P} s')\}.$$

Observe that $|F, s|$ is closed under reachability: $s' \in |F, s| \implies |F, s'| \subseteq |F, s|$. If $F \models s \xrightarrow{\Box\Phi:\rightarrow(P\cdot Q)} s'$, then $F \models s \xrightarrow{PQ} s'$ and Φ holds in all of $|F, s|$, so in particular in those states connecting $\xrightarrow{P}$ and $\xrightarrow{Q}$. Conversely, if $F \models s \xrightarrow{(\Box\Phi:\rightarrow P)(\Box\Phi:\rightarrow Q)} s'$, then there is a t with $F \models s \xrightarrow{\Box\Phi:\rightarrow P} t$ and $F \models t \xrightarrow{\Box\Phi:\rightarrow Q} s'$. Now use $|F, t| \subseteq |F, s|$.

This observation illustrates a phenomenon that is not present in the specification languages LOTOS (ISO 1987), PSF (Mauw and Veltink 1990, 1993) and μCRL (Groote and Ponse 1991, 1995), where only the Booleans t and f occur in conditional programming constructs. (Note that (1) is valid for those values of Φ.) In Bergstra et al. 1994, it is argued that *backtracking* can be modelled in process algebra using propositions as conditions (going with a non-trivial modification of bisimulation semantics).

9 Expressivity

Ongoing research led to the following result:

Theorem 9.1. *Each pointed, recursively enumerable frame can be expressed up to weak bisimulation in ACP with abstraction,* $^\sharp$ *and* *.

(The process algebra operation $^\sharp$ is shortly discussed in the latter part of Section 1.) Here we do not give a full proof of this result, but remark the following: stacks can be specified with * and $^\sharp$, and regular processes with * only (see Bergstra et al. 1994. As to the "frame-part" of this result: a pointed frame can be transformed into a computable one that is weakly bisimilar and of which all states have out-degree 2. And such frames can be expressed in the system mentioned above, which suffices to represent finite control and stacks, and thereby arbitrary Turing Machines.

References

Apt, K.R. 1981. Ten Years of Hoare's Logic, a Survey, Part I. *ACM Transactions on Programming Languages and Systems* 3(4):431–483.

Baeten, J.C.M., and J.A. Bergstra. 1992. Process Algebra with Signals and Conditions. In *Programming and Mathematical Methods, Proceedings Summer School Marktoberdorf 1991*, ed. M. Broy, 273–323. Springer-Verlag. NATO ASI Series F88.

Baeten, J.C.M., and J.A. Bergstra. 1994. Process Algebra with Propositional Signals. Logic Group Preprint Series 123. Utrecht: CIF, State University of Utrecht.

Baeten, J.C.M., J.A. Bergstra, and J.W. Klop. 1987. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science* 51(1/2):129–176.

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

Bergstra, J.A., I. Bethke, and A. Ponse. 1994. Process algebra with iteration and nesting. *Computer Journal* 37(4):243–258.

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

Bergstra, J.A., and J.W. Klop. 1985. Algebra of communicating processes with abstraction. *Theoretical Computer Science* 37(1):77–121.

Bergstra, J.A., and A. Ponse. 1994. Frame algebra with synchronous communication. In *Working Papers of the International Workshop on Information Systems - Correctness and Reusability IS-CORE'94*, ed. R.J. Wieringa and R.B. Feenstra. Free University, Amsterdam. To appear.

Bergstra, J.A., A. Ponse, and J.J. van Wamel. 1994. Process Algebra with Backtracking. In *Proceedings of the REX Symposium "A Decade of Concurrency: Reflections and Perspectives"*, ed. J.W. de Bakker, W.P. de Roever, and G. Rozenberg, Lecture Notes in Computer Science, Vol. 803, 46–91. Springer-Verlag.

Bull, R.A., and K. Segerberg. 1984. Basic Modal Logic. In *Handbook of Philosophical Logic, Vol. II*, ed. D. Gabbay and F. Guenthner. 1–88. Reidel.

Cook, S.A., and D.C. Oppen. 1975. An Assertion Language for Data Structures. In *Conference Record of the* 2^{nd} *ACM Symposium on Principles of Programming Languages*, 160–166.

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

Groote, J.F., and A. Ponse. 1991. μCRL: A Base for Analysing Processes with Data. In *Proceedings* 3^{rd} *Workshop on Concurrency and Compositionality, Goslar, GMD-Studien Nr. 191*, ed. E. Best and G. Rozenberg, 125–130. Universität Hildesheim, May.

Groote, J.F., and A. Ponse. 1994. Process Algebra with Guards. Combining Hoare Logic and Process Algebra. Formal Aspects of Computing 6:115–164. An extended abstract appeared in *Proceedings CONCUR 91,* Amsterdam, ed. J.C.M. Baeten and J.F. Groote, Lecture Notes in Computer Science, Vol. 527, 235–249. Springer-Verlag, 1991.

Groote, J.F., and A. Ponse. 1995. The Syntax and semantics of μ-CRL. In *Algebra of Communicating Processes, Utrecht 1994*, ed. A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, 26–62. Workshops in Computing. Springer-Verlag.

Harel, D. 1984. Dynamic Logic. In *Handbook of Philosophical Logic, Vol. II*, ed. D. Gabbay and F. Guenthner. 497–604. Reidel.

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

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

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

Kleene, S.C. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, 3–41. Princeton University Press.

Mauw, S., and G.J. Veltink. 1990. A process specification formalism. *Fundamenta Informaticae* XIII:85–139.

Mauw, S., and G.J. Veltink (ed.). 1993. *Algebraic Specification of Communication Protocols.* Cambridge Tracts in Theoretical Computer Science 36. Cambridge University Press.

Milner, R. 1984. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences* 28:439–466.

Milner, R. 1989. *Communication and Concurrency.* Englewood Cliffs: Prentice-Hall International.

Ponse, A. 1991. Process Expressions and Hoare's Logic. *Information and Computation* 95(2):192–217.

Vaandrager, F.W. 1993. Expressiveness Results for Process Algebras. In *Proceedings REX Workshop on Semantics: Foundations and Applications,* Beekbergen, The Netherlands, June 1992, ed. J.W. de Bakker, W.P. de Roever, and G. Rozenberg, Lecture Notes in Computer Science, Vol. 666, 609–638. Springer-Verlag.