Mode Transfer in Process Algebra

by

J.C.M. Baeten and J.A. Bergstra

00/01

ISSN 0926-4515

All rights reserved
editors: prof.dr. J.C.M. Baeten
prof.dr. P.A.J. Hilbers

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Reports 00/01
Eindhoven, January 2000
Mode Transfer in Process Algebra

J.C.M. Baeten
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
josb@win.tue.nl, http://www.win.tue.nl/win/cs/fm/josb/

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

Abstract

This paper provides a systematic and full treatment of mode transfer operators in process algebra, including complete axiomatizations, operational rules, analysis of expressive power and extensions with timing features. In particular, we study a disrupt operator and an interrupt operator.

Note: this paper is a revision and extension of [7]

1 Introduction

A useful feature in programming languages and specification languages is the ability to denote mode switches. In particular, most languages have means to describe the disrupt or interrupt the normal execution of a system. Also in process algebra, various disrupt and interrupt operators have received attention, see e.g. [7], [10], [11], [3], [12]. In LOTOS (see [9]) we have the disruption operator, that is denoted \( \triangleright \). Another name is disabling. In this paper, we provide a systematic and full treatment of mode transfer operators in process algebra, including complete axiomatizations, operational rules, analysis of expressive power and extensions with timing features.

We write disrupt as \( \rightarrow \triangleright \). The process \( x \rightarrow \triangleright y \) starts behaving as \( x \) but may, at any moment in time, leave \( x \) and proceed with the execution of \( y \). Further, we consider the interrupt operator \( \rightarrow \triangleright \). The process \( x \rightarrow \triangleright y \) likewise starts behaving as \( x \), may at any moment in time leave \( x \) and proceed with \( y \), but upon completion of \( y \) resumes the execution of \( x \). Disrupt and interrupt both describe a form of mode transfer, and thus can be called mode transfer operators.

Addition of these operators to process algebra follows the usual pattern: we will define them in a couple of axioms, and show they can be eliminated from closed terms. Thus, adding the operators forms a conservative extension of the standard theory. On the other hand, we will show that with recursion, the mode transfer operators do add expressive power.
We provide an operational interpretation by defining structured operational rules, thereby developing the standard model of transition systems modulo bisimulation equivalence. We show that our axiomatisation is sound and complete for this model.

The usefulness of the operators shows in writing intuitively understandable specifications of processes. Also other theories and other process algebras have been endowed with mode transfer operators, thus underlining their intuitive appeal. In our view, any addition of an extra operator to process algebra is warranted as soon as this is useful for the purpose of process specification in a given context, as long as it is done in a neat and consistent way. Thus, a practitioner can select the combination of features that are important in her/his case study, and be provided with a "made to order" process algebra.

2 Basic Process Algebra

We start out from the standard process algebra BPA, Basic Process Algebra with inaction. The signature elements are:

- Constants \( a \) from some given set \( A \) are the atomic processes, the basic building blocks of our process terms. Process \( a \) will execute action \( a \) at some (unspecified) moment in time, and will then terminate immediately and successfully.

- Binary operator \( + \) denotes alternative composition or choice. Process \( x + y \) executes either \( x \) or \( y \), but not both. The choice is resolved upon execution of the first action.

- Binary operator \( \cdot \) denotes sequential composition.

- Constant \( \delta \) denotes inaction, and is the neutral element of alternative composition. Process \( \delta \) cannot execute any action, and cannot terminate. In a setting with time, \( \delta \) does not block advancement of time, so the process can also be called livelock.

The process algebra BPA is axiomatised by axioms A1-7 in Table 1. Omitting constant \( \delta \), just having axioms A1-5, yields the process algebra BPA. More about these process algebras can be found in standard references [6], [5].

Now we add the mode transfer operators to this theory. Both \( x \xrightarrow{\delta} y \) and \( x \xrightarrow{\gamma} y \) are written as the sum of two alternatives, depending whether the first action that is executed is a first action of \( x \) or a first action of \( y \) (axioms DIS and INT). The axioms follow the pattern of the axiomatisation of merge using left merge: the operator is split into two parts, depending on whether the first step comes from the first argument \( \xrightarrow{\delta} \) resp. \( \xrightarrow{\gamma} \) or from the second argument \( \xrightarrow{\gamma} \) resp. \( \xrightarrow{\delta} \).

These auxiliary operators are next axiomatised, left disrupt and left interrupt using the form of the left-hand argument (LDIS1-3 resp. LINT1-3), the definition of right disrupt and right interrupt are even simpler (RDIS resp. RINT) (but will become more complicated in extensions to follow). We call this theory BPAmod, BPA, with mode transfer operators.

A simple consequence of the axioms of BPAmod is given by the following equations:

\[
\delta \xrightarrow{\delta} x = x \quad \text{and} \quad \delta \xrightarrow{\gamma} x = x \cdot \delta.
\]

This differs from the treatment of \( \delta \) in [7], but brings it more in line with our interpretation of \( \delta \) as livelock.

Next, we treat some basic theory. We define the set of basic terms, and show that each closed term can be reduced to a basic term (the so-called elimination theorem). This allows the use of structural induction in proofs like in the proposition to follow (associativity of disrupt).
Table 1: Axioms of BPAMod \( \{ a \in A \cup \{ \delta \} \} \).

**Definition 1** We define a set of closed terms called *basic terms* inductively:

- each \( a \) for \( a \in A \cup \{ \delta \} \) is a basic term;
- for each basic term \( t \) and each \( a \in A \), \( a \cdot t \) is a basic term;
- for each two basic terms \( t, s \), \( t + s \) is a basic term.

We see that basic terms only employ *prefix multiplication*, i.e. the left-hand argument of a sequential composition is always just an atomic action.

**Theorem 2** Let \( t \) be a closed term over the signature of BPAMod. Then there is a basic term \( s \) such that BPAMod \( \vdash t = s \). This is the elimination theorem.

**Proof** Standard term rewrite analysis, see examples in [6] or [5].

Actually, in a proposition in the following section we need the following strengthening of the elimination theorem. Also this strengthening is straightforward to prove.

**Corollary 3** Let \( P \) be a closed term over BPAMod and let \( Q \) be a term over BPAMod possibly involving variables. Then \( P \rightarrow s \) \( Q \) can be written as a term over BPAMod.

**Proposition 4** Disruption is associative, i.e. \( (x \rightarrow y) \rightarrow z = x \rightarrow (y \rightarrow z) \) holds for all closed BPAMod terms \( x, y, z \).

**Proof** Using the elimination theorem, it suffices to prove this for basic terms \( x \). Then, we use structural induction on \( x \), following the definition of basic terms. There are three cases.

1. \( x \equiv a \) \( \{ a \in A \cup \{ \delta \} \} \).

\[
(a \rightarrow y) \rightarrow z = (a + y) \rightarrow z = (a + y) \rightarrow z + (a + y) \rightarrow z = a + y \rightarrow z + z = a + y \rightarrow z = a \rightarrow (y \rightarrow z).
\]
2. \( x \equiv a \cdot x' \ (a \in A) \), the proposition is true for \( x' \).

\[
(a \cdot x' \rightarrow y) \rightarrow z = (a \cdot (x' \rightarrow y) + y) \rightarrow z = a \cdot ((x' \rightarrow y) \rightarrow z) + y \rightarrow z + z =
\]

\[
=a \cdot (x' \rightarrow (y \rightarrow z)) + y \rightarrow z = a \cdot x' \rightarrow (y \rightarrow z) + a \cdot x' \rightarrow (y \rightarrow z) =
\]

\[
=a \cdot x' \rightarrow (y \rightarrow z).
\]

3. \( x \equiv x' + x'' \), the proposition is true for \( x', x'' \).

\[
((x' + x'') \rightarrow y) \rightarrow z = (x' \rightarrow y + x'' \rightarrow y + y) \rightarrow z = (x' \rightarrow y + x'' \rightarrow y) \rightarrow z =
\]

\[
=(x' \rightarrow y) \rightarrow z + (x'' \rightarrow y) \rightarrow z + z = (x' \rightarrow y) \rightarrow z + (x'' \rightarrow y) \rightarrow z =
\]

\[
=x' \rightarrow (y \rightarrow z) + x'' \rightarrow (y \rightarrow z) + y \rightarrow z =
\]

\[
=(x' + x'') \rightarrow (y \rightarrow z) + y \rightarrow z = (x' + x'') \rightarrow (y \rightarrow z).
\]

\[\Box\]

Likewise, we can prove \( x \rightarrow (y \rightarrow z) = (x \rightarrow y) \rightarrow z \) and \( x \rightarrow (y \rightarrow z) = (x \rightarrow y) \rightarrow z \). Similar statements for the interrupt operator do not hold, e.g. we have

\[
a \rightarrow (b \rightarrow c) = a + ba + cba
\]

while

\[
(a \rightarrow b) \rightarrow c = a + b \cdot (a + ca) + c \cdot (a + ba).
\]

Next, we provide a model for BPAnod on the basis of structured operational rules (so-called SOS rules) in the style of Plotkin, see e.g. [13]. The rules in Table 2 define the following relations on closed BPAnod terms: binary relations \( \xrightarrow{a} \) and unary relations \( \xrightarrow{\alpha} \sqrt{\ } \) (for \( a \in A \)). Intuitively, they have the following meaning:

- \( x \xrightarrow{\alpha} x' \) means that \( x \) evolves into \( x' \) by executing atomic action \( \alpha \);
- \( x \xrightarrow{\alpha} \sqrt{\ } \) means that \( x \) successfully terminates upon execution of \( \alpha \)

The rules provide a transition system for each closed term. Next, we define an equivalence relation on these transition systems in the standard way.

**Definition 5** We say two closed terms \( t, s \) are *bimiliar*, notation \( t \bimiliar s \) if there is a binary relation on closed terms \( R \), called a *bisimulation*, such that the following holds:

- \( R(t, s) \)
- whenever \( R(x, y) \) and \( x \xrightarrow{\alpha} x' \) then there is a term \( y' \) such that \( y \xrightarrow{\alpha} y' \) and \( R(x', y') \)
- whenever \( R(x, y) \) and \( y \xrightarrow{\alpha} y' \) then there is a term \( x' \) such that \( x \xrightarrow{\alpha} x' \) and \( R(x', y') \)
- whenever \( R(x, y) \) then \( x \xrightarrow{\alpha} \sqrt{\ } \) iff \( y \xrightarrow{\alpha} \sqrt{\ } \)
Proposition 6 Bisimulation is a congruence relation on closed BPAmod terms.

Proof This is a standard result following from the format of the deduction rules, see e.g. [5]. \(\square\)

Theorem 7 The theory BPAmod is sound and complete for the model of transition systems modulo bisimulation, i.e. for all closed terms \(t, s\) we have

\[
\text{BPAmod} \vdash t = s \iff t \bisim s
\]

Proof This is also a standard result, based on the elimination theorem and the fact that the theory with mode transfer operators is a conservative extension of the theory without these operators. We can follow the recipe of [5]. \(\square\)

The extension of the basic theory treated in this section with parallel composition (or merge, denoted \(\parallel\)), with or without communication, is entirely standard, see e.g. [6] or [5].

3 Recursive Definitions

We use recursion to specify processes with possible infinite behaviour as is standard in process algebra, see e.g. [6] or [5].

Let \(V\) be a set of variables ranging over processes. A recursive specification \(E = E(V)\) is a set of equations \(E = \{ X \rightarrow t_X \mid X \in V \}\) where each \(t_X\) is a term over the signature in question.
(in this case, BPAnod) and variables from $V$. A solution of a recursive specification $E(V)$ in our theory is a set of processes $\{(X | E) \mid X \in V\}$ in some model of the theory such that the equations of $E(V)$ hold, if for all $X \in V$, $X$ stands for $(X | E)$. Mostly, we are interested in one particular variable $X \in V$.

Let $t$ be a term containing a variable $X$. We call an occurrence of $X$ in $t$ guarded if $t$ has a subterm of the form $a \cdot s$ with $a \in A$ and $s$ a term containing this occurrence of $X$. We call a recursive specification guarded if all occurrences of all its variables in the right-hand sides of all its equations are guarded or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification. We can add SOS rules for recursion to the rules in Table 2, such that in the resulting model for BPAnod, all recursive specifications over BPAnod have a solution, and all guarded recursive specifications have a unique solution. We say that the process that is the unique solution of a finite $E$ is defined by $E$.

The SOS rules, presented in Table 3, come down to looking upon $(X | E)$ as the process $(t_X | E)$, which is $t_X$ with, for all $Y \in V$, all occurrences of $Y$ in $t_X$ replaced by $(Y | E)$.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{(t_X</td>
<td>E) \rightarrow y}{(X</td>
</tr>
</tbody>
</table>

Table 3: Deduction rules for recursion ($a \in A$).

We call an equation linear if it has the form

$$X = a_1 \cdot X_1 + \cdots + a_n \cdot X_n + b_1 + \cdots + b_m,$$

for certain $n, m \geq 0, a_i \in A, X_i \in V, b_j \in A \cup \{\delta\}$. We call a process $y$ a state of process $x$ iff $y$ can be reached from $x$ by executing a sequence of steps. We call a process regular if its bisimulation equivalence class contains a transition system with finitely many states. It is a well-known result (see e.g. [8]) that a process is regular iff it can be defined by a finite linear recursive specification.

**Proposition 8** Consider the recursive equation $X = a \cdot X \rightarrow bc$. Obviously, this equation is guarded. The solution of this equation is not regular.

**Proof** It is easy to see that the equation is guarded, so it has a unique solution in the operational model, abusing notation called $X$. Define $T_0 = b \cdot c$, and $T_{n+1} = bc \rightarrow T_n$ for each $n$. Notice that process $T_n$ can execute a series of $m$ $b$-steps followed by $c$ for each $m \leq n + 1$, but not for $m > n + 1$. Thus, all $T_n$ are different processes, and cannot be bisimilar. Next, process $X$ can execute $n$ consecutive $a$-steps, resulting in process $X \rightarrow T_n$. Again, all states $X \rightarrow T_n$ are different, so process $X$ has infinitely many different states. Thus, $X$ is not regular.

**Proposition 9** If $P$ is regular then the solution of $X = P \rightarrow X$ is also regular.

**Proof** On the basis of the deduction rules, we can derive that each state of $X$ is of the form $Q \rightarrow X$, where $Q$ is a state of $P$. Thus, $X$ has no more states than $P$. 


Often, the disrupt operator is used to describe exception handling. This can take the form of a recursion equation \( X = N \mapsto E \cdot X \), where \( N \) denotes the normal behaviour of the system, and \( E \) denotes a process that signals exceptions and performs some exception handling.

Next, we show that recursive definitions over \( \text{BPA}_\text{mod} \) lead outside the domain of processes that are recursively definable over \( \text{BPA}_\delta \).

**Proposition 10** Let \( C \) be a counter process, e.g. defined by the pair of equations
\[
C = a \cdot D \cdot C, \quad D = b + a \cdot D \cdot D.
\]
Then the process \( P = C \mapsto c \) is not recursively definable over \( \text{BPA}_\delta \).

**Proof** This goes exactly like the proof of theorem 3.9 in [2] (only a little simpler). \( \square \)

This proposition hinges on the fact that the process that is disrupting, i.e. \( c \), terminates. For, the next proposition tells us, that if the disrupting process cannot terminate (we say, the process is *perpetual*), then definability is preserved.

**Proposition 11** If \( P, Q \) are recursively definable over \( \text{BPA}_\delta \) and \( Q \) is perpetual, then \( P \mapsto Q \) is recursively definable over \( \text{BPA}_\delta \).

**Proof** Let \( E = \{ X_i = t_i \mid 1 \leq i \leq n \} \) be a recursive specification for \( P \) and let \( F = \{ Y_j = s_j \mid 1 \leq j \leq m \} \) be a recursive specification for \( Q \) (so \( P \) can be substituted for \( X_i \) and \( Q \) for \( Y_j \)). Obtain \( t_i^* \) from \( t_i \) by replacing every \( X_i \) by \( Z \), then \( \{ Z_i = t_i^* + Y_j \mid 1 \leq i \leq n \} \cup F \) is a recursive specification for \( P \mapsto Q \). \( \square \)

Note that the similar statement for \( \text{PA} \) does not hold. This is the content of the following proposition. Let us take another copy of a counter, say
\[
E = d \cdot F \cdot E, \quad F = c + d \cdot F \cdot F,
\]
then \( B = C \parallel E \) denotes a bag over two elements (with inputs \( a, d \) and outputs \( b, e \)). It is well-known (see e.g. [8]) that \( B \) is also the solution of the following recursive equation:
\[
B = a \cdot (b \parallel B) + d \cdot (c \parallel B),
\]
and that \( B \) is not definable over \( \text{BPA}_\delta \).

**Proposition 12** Let \( P = c \cdot P \) and \( B \) be as above (a bag over two elements), then \( B \mapsto P \) is not definable over \( \text{PA} \).

**Proof** Suppose not, so \( B \mapsto P \) is defined by recursive specification \( E = \{ X_i = t_i \mid 1 \leq i \leq n \} \) over \( \text{PA} \). Consider a run of the system in which \( c \) does not occur, so it just consists of actions \( a, b, d, e \). We claim that there is such a trace leading to an expression \( F \) in which a parallel composition of the form \( R \parallel S \) occurs unguarded such that one of \( R, S \) has an initial action different from \( c \). We prove the claim below.

Given the claim, we can finish the proof as follows. Say, without loss of generality, that \( R \) has an initial action \( a \) and \( S \) has an initial action \( c \). Then we can execute also \( a \) after \( c \), contradicting the definition of the disrupt operator.
To prove the claim, consider the system $E'$, obtained from $E$ by replacing all occurrences of $e$ by $\delta$ and then simplifying as much as possible. The system $E'$ defines the bag, so must by \cite{[8]} still essentially involve the merge operator. Take a variable of $E'$ occurring inside such a merge operator; then any trace of $a, b, d, c$ actions leading from the root state to this variable will satisfy the claim. \hfill \Box

\textbf{Proposition 13} In BPA plus mode transfer operators, it is not possible to give a finite recursive definition of the bag over two elements.

\textbf{Proof} Suppose not, so let $E = \{X_i = t_i \mid 1 \leq i \leq n\}$ be a recursive specification over BPAmod defining the bag $B$. By the corollary to the elimination theorem in the previous section, we can suppose that each left-hand side of each $\rightarrow$ or $\mathit{Idis}$ operator in $E$ denotes an infinite process.

Now we derive that there is some run in $E$ that leads to an expression that has a summand of the form $R \rightarrow S$ or $(R \mathit{Idis} S) \cdot T$, with $R$ an infinite process. Now we have to look at all possibilities for initial steps of $R$ and $S$. We note that $R$ can enter different states of the bag by executing actions, but all these states collapse onto one state by executing one initial action of $S$. This is a contradiction, as the bag does not behave this way.

This proof is rather informal, but can be made more precise by formally defining the notions of state and semi-state, see \cite{[8]}. \hfill \Box

\section{Real Time Process Algebra with Relative Timing}

Next, we look at extensions of the theory presented in the previous section. Especially important is the addition of timing features. First, we consider process algebra with real time in relative timing (see \cite{[1]}). Further on, we look at discrete timing. We use the treatment and notation of \cite{[4]}, since it allows a smooth integration of real time and discrete time theories, and also allows separation of actions and timing information. The framework of \cite{[4]} allows for an integrated treatment of all process algebras with timing. Notice that this treatment for dense time contains so-called \textit{urgent} actions, which means that more than one action can occur consecutively at the same moment of time. Of course, this is an abstraction of reality (but a useful one). We can either consider that consecutive execution at the same moment of time denotes execution of independent actions, or consider that closer observation will find a time difference. Be that as it may, it is clear that algebraically, this treatment is superior to one where such behaviour is excluded.

We have the following syntax in addition to the operators $+, \cdot$ of BPA:

- \textit{urgent atomic actions} $\bar{a}$, where $a \in A$. The process $\bar{a}$ denotes immediate and urgent execution of the action $a$, at the current moment of time, followed by immediate and successful termination.

- \textit{time stop} $\delta$. Time cannot progress any more beyond the current moment of time, and no termination can take place. Notice that we do not include the constant $\delta$, for which the current moment of time is already inconsistent. In the absence of this second constant, $\bar{\delta}$ is the neutral element of alternative composition.
• the relative delay operator $\sigma_{rel}$. This operator takes a non-negative real number $p$ and a process $x$, and will delay execution of $x$ for amount of time $p$. We write $\sigma^p_{rel}(x)$, or often $\sigma^p(x)$, instead of $\sigma_{rel}(p,x)$.

Next, we have three auxiliary operators.

• the relative time-out operator $\nu_{rel}$. $\nu^p_{rel}(x)$, or rather $\nu^p(x)$, is that part of process $x$ that starts within $p$ time units ($p \geq 0$).

• the relative initialisation operator $\nu^p_{rel}(x)$, or rather $\nu^p(x)$, is that part of process $x$ that starts after $p$ time units ($p \geq 0$).

• the now operator $\nu_{rel}$, or simply $\nu$, will block all advancement of time, and will only allow urgent actions to take place.

We have the following extra observations on alternative and sequential composition:

• The choice in alternative composition $+$ is resolved by the execution of an action, not by the mere passage of time. This is expressed by the axiom of time factorisation SRT3.

• For sequential composition $\cdot$, relative timing means that time is measured from the execution of the previous action. This is axiom SRT4.

The axiomatisation of BPA"† adds the axioms in Table 4 to the axioms of BPA. Naming of axioms is taken from [4]. Notice we have a different variant of SRI2, due to the absence of the constant $\delta$. We have an elimination theorem: the auxiliary operators can be eliminated.

Using the axioms, we can obtain the following result: for each term $x$, either it is the case that all initial actions start at the current moment of time, and then $x = \nu(x)$, or there is a number $r > 0$ such that $x$ can idle for $r$ time units, and then we can write $x = \nu^r(x) + \sigma^r(y)$ for a certain term $y$. In the second case the first summand denotes that part of $x$ that starts execution within $r$ time units from now, and the second part is the part that starts execution no sooner than $r$ time units from now. For more details, see [4]. We will use these representations in the axiomatisations to come.

The definition of an operational semantics by means of SOS deduction rules is straightforward. To the relations of Table 2, we add binary relations $\cdot \vdash t \cdot \cdot$ on closed terms (for $t > 0$). Intuitively, $x \vdash t \cdot x'$ means that $x$ evolves into $x'$ by waiting for time $t$. We add the rules in Table 5 to the rules of Table 2. Note that $x \vdash t \cdot \cdot$ means that $x$ cannot execute a $\cdot \vdash t$ transition, i.e. $x$ cannot wait for time $t$. Thus, we have here an SOS definition with negative premises, that is well-defined.

Now we add the interrupt and disrupt operators in this framework. We present a few motivating examples for the disrupt operator.

- $\sigma^1(\tilde{a}) \vdash \sigma^1(\tilde{b}) = \sigma^1(\sigma^1(\tilde{a}) + \tilde{b})$. At time 1, the choice is made whether to start the disruption, or not.

- $\sigma^1(\tilde{a}) \vdash \sigma^2(\tilde{b}) = \sigma^1(\tilde{a})$. At time 1, $a$ must be executed, and the disruption comes too late, after the process has finished.

- $\sigma^1(\tilde{a}) \vdash \sigma^1(\tilde{b}) = \sigma^1(\tilde{a} + \tilde{b})$. At time 1, there is a choice whether to execute $a$ or start a disruption.
\[\sigma^0(x) = x\]  
\[\sigma^p(\sigma^q(x)) = \sigma^{p+q}(x)\]  
\[\sigma^p(x) + \sigma^p(y) = \sigma^p(x+y)\]  
\[\sigma^p(x \cdot y) = \sigma^p(x) \cdot y\]  
\[\nu^0(x) = \tilde{\nu}\]  
\[\nu^p(\tilde{\nu}) = \tilde{\nu}\]  
\[\nu^{p+q}(\sigma^p(x)) = \sigma^p(\nu^q(x))\]  
\[\nu^r(x+y) = \nu^r(x) + \nu^r(y)\]  
\[\nu^r(x \cdot y) = \nu^r(x) \cdot y\]

<table>
<thead>
<tr>
<th>SRT1</th>
<th>SRT2</th>
<th>SRT3</th>
<th>SRT4</th>
<th>SRT5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x + \tilde{\nu} = x)</td>
<td>(\tilde{\nu} \cdot x = \tilde{\nu})</td>
<td>(\sigma^p(x) + \sigma^p(y) = \sigma^p(x+y))</td>
<td>(\nu(\tilde{\nu}) = \tilde{\nu})</td>
<td>(\sigma^p(\nu^q(x)) = \sigma^p(\nu^q(x)))</td>
</tr>
</tbody>
</table>

Table 4: Axioms of BPA in relative real time \(a \in A \cup \{\delta\}, p, q \geq 0, r > 0\).

Next, we look at the interrupt operator. Consider the example \(\sigma^{10}(\tilde{\nu}) \rightarrow \sigma^2(\tilde{\nu})\). The process \(\sigma^{10}(\tilde{\nu})\) can be interrupted by the execution of \(b\) after two time units. After the execution of \(b\), control goes back. Since two time units have passed, the remaining process is \(\sigma^8(\tilde{\nu})\).

A second example is \((\sigma^4(\tilde{\nu}) + \sigma^{10}(\tilde{\nu})) \rightarrow (\sigma^2(\tilde{\nu}) \cdot \sigma^5(\tilde{\nu}))\). Upon interruption (after 2 time units), we execute \(c\) followed, after 5 time units, by \(d\). At this point, control goes back, the option to do \(a\) has timed out, and \(\sigma^3(\tilde{\nu})\) remains.

Finally, \(\sigma^4(\tilde{\nu}) \rightarrow (\sigma^2(\tilde{\nu}) \cdot \sigma^5(\tilde{\nu}))\), when interruption occurs after two time units, will execute \(c\) followed by \(\sigma^2(\tilde{\nu})\), as deadlock will occur when the interrupted process can no longer wait.

We show the axiomatisation of the mode transfer operators in Table 6. Again, we have an elimination theorem: all mode transfer operators can be eliminated from closed terms. As a consequence, we can prove that the axiomatisation is sound and complete for the operational model of transition systems modulo bisimulation equivalence.

We add the SOS rules in Table 7.

### 5 Discrete Time

Now we look at what happens when we discretise timing. The signature elements now become:

- Process \(\underline{a}\) for atomic action \(a\) means that \(a\) is executed in the current time slice. This means that \(a\) can be executed at the current moment of time, or at any time between now and the first integer time value (measured on a global clock).

- Likewise, \(\underline{\delta}\) stands for deadlock at the end of the current time slice. Thus, time can progress up to the first integer time value.

- the relative delay operator \(\sigma\), the relative time out operator \(\nu\) and the relative initialisation operator \(\overline{\nu}\) all take only natural numbers in the first argument. \(\sigma^1\) denotes passage to the next time slice.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Rule</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{a} \xrightarrow{a} \sqrt{\bar{a}} )</td>
<td>( \bar{x} \xrightarrow{a} x' )</td>
<td>( \sigma^0(x) \xrightarrow{a} x' )</td>
</tr>
<tr>
<td>( \sigma^{p+r}(x) \xrightarrow{r} \sigma^p(x) )</td>
<td>( \bar{x} \xrightarrow{r} x' )</td>
<td>( \sigma^0(x) \xrightarrow{a} \sqrt{\bar{a}} )</td>
</tr>
<tr>
<td>( x \xrightarrow{r} x' ), ( y \xrightarrow{r} y' )</td>
<td>( x \xrightarrow{r} x' ), ( y \xrightarrow{r} y' )</td>
<td>( \bar{x} \xrightarrow{a} \sqrt{\bar{x}} )</td>
</tr>
<tr>
<td>( x + y \xrightarrow{r} x' + y' )</td>
<td>( x + y \xrightarrow{r} x' )</td>
<td>( \nu(x) \xrightarrow{a} \sqrt{\nu(x)} )</td>
</tr>
<tr>
<td>( x \xrightarrow{r} x' )</td>
<td>( x \xrightarrow{a} x' )</td>
<td>( \nu(x) \xrightarrow{a} \sqrt{\bar{x}} )</td>
</tr>
<tr>
<td>( \bar{v}(x) \xrightarrow{a} x' )</td>
<td>( \nu(x) \xrightarrow{a} \sqrt{\nu(x)} )</td>
<td>( x \xrightarrow{r} x', \nu \leq p \leq r ) ( \nu(x) \xrightarrow{r} x' )</td>
</tr>
<tr>
<td>( \nu(x) \xrightarrow{a} x' )</td>
<td>( x \xrightarrow{r} x' )</td>
<td>( \sigma^0(x) \xrightarrow{a} x' )</td>
</tr>
</tbody>
</table>

Table 5: Deduction rules for BPA with relative real time (\( a \in A, r > 0, p \geq 0 \)).

- The other operators do not change.

In the axiomatisation, all we need to do in Table 4 is to change constants \( \bar{a}, \bar{\delta} \) into \( a, \delta \), replace every \( S \) in an axiom name into a \( D \), and let \( p, q \) range over natural numbers, \( r \) range over positive natural numbers.

Likewise, in the deduction rules, we limit \( r \) to positive natural numbers, \( p \) to natural numbers. In fact, it is enough to consider time steps \( 1 \rightarrow \).

Axiomatization and deduction rules for mode transfer is now straightforward. Elimination, soundness and completeness follows.

6 Conclusion

We have provided a systematic and full treatment of mode transfer operators in process algebra, including complete axiomatizations, operational rules, analysis of expressive power and extensions with timing features. This can be used in system specifications and in semantics of programming and specification languages.
\[ x \mapsto y = x \rightarrow y + x \leftarrow y \]
\[ \alpha \mapsto x = \tilde{\alpha} \]
\[ (\tilde{\alpha} x) \mapsto y = \tilde{\alpha} \cdot (x \rightarrow y) \]
\[ (x + y) \mapsto z = x \rightarrow z + y \rightarrow z \]
\[ \sigma^r(x) \mapsto \nu^r(y) = \sigma^r(x) \]
\[ \sigma^r(x) \mapsto (\nu^r(y) + \sigma^r(z)) = \sigma^r(x \rightarrow z) \]
\[ \tilde{x} \rightarrow \tilde{\alpha} = \tilde{\alpha} \]
\[ x \rightarrow (\tilde{\alpha} y) = \tilde{\alpha} y \]
\[ \nu(x) \rightarrow \sigma^r(y) = \delta \]
\[ (\nu^r(x) + \sigma^r(y)) \mapsto \alpha^r(z) = \sigma^r(x \rightarrow z) \]

<table>
<thead>
<tr>
<th>Mode Transfer in BPA^rt(a \in A \cup {\delta}, r &gt; 0).</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIS</td>
</tr>
<tr>
<td>LDIS1</td>
</tr>
<tr>
<td>LDIS2</td>
</tr>
<tr>
<td>LDIS3</td>
</tr>
<tr>
<td>LDIS4</td>
</tr>
<tr>
<td>LDIS5</td>
</tr>
<tr>
<td>RDIS1</td>
</tr>
<tr>
<td>RDIS2</td>
</tr>
<tr>
<td>RDIS3</td>
</tr>
<tr>
<td>RDIS4</td>
</tr>
</tbody>
</table>

Table 6: Mode transfer in BPA^rt(a \in A \cup \{\delta\}, r > 0).
Table 7: Deduction rules for mode transfer in BPA_{str}(a \in A, r > 0).

References


Computing Science Reports

In this series appeared:

96/01  M. Voorhoeve and T. Basten  Process Algebra with Autonomous Actions, p. 12.
96/02  P. de Bra and A. Aerts  Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12.
96/06  W.M.P. van der Aalst and T. Basten  Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07  M. Voorhoeve  Structural Petri Net Equivalence, p. 16.
96/11  T. Laan  AUTOMATH and Pure Type Systems, p. 30.
96/12  F. Kamareddine and T. Laan  A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
96/13  T. Borghuis  Priorian Tense Logics in Modal Pure Type Systems, p. 61.
96/14  S.H.J. Bos and M.A. Reniers  The $I^1C$-bus in Discrete-Time Process Algebra, p. 25.
96/15  M.A. Reniers and J.J. Vereijken  Completeness in Discrete-Time Process Algebra, p. 139.
96/16  E. Boiten and P. Hoogendijk  Nested collections and polytypism, p. 11.
96/18  P.D.V. van der Stok  Real-Time Distributed Concurrency Control Algorithms with mixed time constraints, p. 71.
96/19  M.A. Reniers  Static Semantics of Message Sequence Charts, p. 71.
96/20  L. Feijs  Algebraic Specification and Simulation of Lazy Functional Programs in a concurrent Environment, p. 27.
96/23  W.M.P. van der Aalst  Structural Characterizations of sound workflow nets, p. 22.
96/24  M. Voorhoeve and W. van der Aalst  Conservative Adaptation of Workflow, p. 22.
96/25  M. Vaccari and R.C. Backhouse  Deriving a systolic regular language recognizer, p. 28.
97/06  M. Franssen  Tools for the Construction of Correct Programs: an Overview, p. 33.
97/07  J.C.M. Baeten and J.A. Bergstra  Bounded Stacks, Bags and Queues, p. 15.
97/08  P. Hoogendijk and R.C. Backhouse  When do datatypes commute? p. 35.
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>99/06</td>
<td>T. Basten and W. v.d. Aalst</td>
<td>Inheritance of Workflows: An Approach to tackling problems related to change, p. 66</td>
</tr>
<tr>
<td>99/10</td>
<td>T.A.C. Willemse</td>
<td>The Analysis of a Conveyor Belt System, a case study in Hybrid Systems and timed ( \mu ) CRL, p. 44</td>
</tr>
<tr>
<td>99/11</td>
<td>J.C.M. Baeten and C.A. Middelburg</td>
<td>Process Algebra with Timing: Real Time and Discrete Time, p. 50</td>
</tr>
<tr>
<td>99/12</td>
<td>S. Andova</td>
<td>Process Algebra with Probabilistic Choice, p. 38</td>
</tr>
<tr>
<td>99/14</td>
<td>A. Engels and S. Mauw</td>
<td>Why men (and octopuses) cannot juggle a four ball cascade, p. 10</td>
</tr>
<tr>
<td>99/16</td>
<td>G.J. Houben, P. Lemmens</td>
<td>A Software Architecture for Generating Hypermedia Applications for Ad-Hoc Database Output, p. 13</td>
</tr>
<tr>
<td>99/18</td>
<td>J.C.M. Baeten and T. Basten</td>
<td>Partial-Order Process Algebra (and its Relation to Petri Nets), p. 79</td>
</tr>
<tr>
<td>99/19</td>
<td>J.C.M. Baeten and C.A. Middelburg</td>
<td>Real Time Process Algebra with Time-dependent Conditions, p. 33</td>
</tr>
</tbody>
</table>