Deadlock Behaviour in Split and ST Bisimulation Semantics

by

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

98/03
Deadlock Behaviour in Split and ST Bisimulation Semantics

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.ruu.nl, http://www.wins.uva.nl/research/prog/people/janb/

Abstract

We investigate split and ST bisimulation semantics, in particular the deadlock behaviour of processes in these semantics. We define and axiomatise a variant of ACP, where atomic actions and durational actions coexist, and define split and ST bisimulation semantics on this theory. We exhibit a closed term that has a deadlock in split semantics, but not in ST bisimulation semantics, and vice versa: a closed term that has a deadlock in ST bisimulation semantics but not in split semantics. As an application, we investigate different versions of durational communication.

1 Introduction

From the literature, see e.g. [12], [8], [11], we find that ST bisimulation semantics is a preferred non-interleaving semantics, due to several nice properties of this semantics. The axiomatisation of ST bisimulation semantics in [8] makes this semantics amenable for investigation in an axiomatically presented process algebra as ACP. This axiomatisation starts out by considering split semantics, another non-interleaving semantics.
In this article, we investigate split and ST bisimulation semantics in the setting of the process algebra ACP. We obtain an algebra in which atomic actions (for which the interleaving paradigm holds) and durational actions (that allow concurrent behaviour) coexist. We propose notation and give an axiomatisation based on [8].

We stick to the viewpoint that the basic building blocks of the ACP process algebra are atomic actions. Atomic actions are truly atomic, i.e. (the observation of) the execution of an action is instantaneous. This is the basis of the interleaving paradigm (in a parallel composition, atomic actions from different components are interleaved or merged). Thus, atomic actions have no duration and cannot be refined. Besides these, we may want to give names to certain (sub)processes, and call these also actions, durational actions. Here, durational actions are defined as the sequential composition of two atomic actions, its beginning and its ending. Doing just that gives rise to split semantics, if we also ensure that an ending matches the corresponding beginning, then we obtain ST bisimulation semantics.

In the two theories, we investigate deadlock behaviour. We exhibit a closed term that has a deadlock in split semantics, but not in ST bisimulation semantics, and vice versa: a closed term that has a deadlock in ST bisimulation semantics but not in split semantics. In each case, we use purely equational reasoning, based on our axiomatisations, to verify the results.

We conclude there is a real difference between the two semantics, of fundamental consequence in the specification and verification of processes.

As an application, we investigate data transfer. Usually, we look upon communication as instantaneous, and use atomic actions to model this. Looking closer, communication can be durational, e.g., first a link is established, and then data is transferred. We consider a couple of alternatives modeled as durational actions, and find also here differing deadlock behaviour.

2 ACP with Priorities and No Exit Iteration

We start out from the well-known theory ACP [7, 5]. We apologize to the reader who has seen this many times before; he can skip the material until Definition 1. The signature of ACP contains the following ingredients:

- A given number of core atomic actions \(r, s, c, t, u,...\). Core atomic actions were introduced in [2]. \(CA\) is the set of core atomic actions. All core atoms are atomic actions, \(CA \subseteq A\), where \(A\) is the set of all atomic actions \(a, b, d,...\). Later, we will construct atomic actions that are not core atoms.

- A constant \textit{inaction} denoted \(\delta\). This constant is the neutral element of alternative composition. This process disallows termination, so can be used to denote deadlock behaviour.

- A binary operator \textit{alternative composition} or \textit{choice}, denoted +. Choice is resolved by the execution of an action.
• A binary operator \textit{sequential composition}, denoted \( \cdot \).

• A binary operator \textit{parallel composition} or \textit{merge}, denoted \( \| \). In a parallel composition, one of the components executes an action, or more than one component execute an action together, a communication action. The merge is axiomatised using two auxiliary operators: \textit{left merge}, denoted \( \| \), and \textit{communication merge}, denoted \( \cdot \). Communication on atomic actions is given, subject to the conditions that communication is commutative and associative, and if one of the arguments is \( 15 \), the communication is \( 15 \).

The communication function on atomic actions is considered a parameter of the theory.

• A unary operator \textit{encapsulation}, denoted \( \partial_H \), that blocks the execution of atomic actions from the parameter set \( H \). It is used to encapsulate communicating actions from the environment.

The axioms of ACP are well-known, and given for easy reference in Table 1. In terms, we often omit the sequential composition operator \( \cdot \). Of all operators, sequential composition binds the strongest, and alternative composition the weakest. The other operators are not ranked.

\[
\begin{align*}
x + y &= y + x & (x + y) + z &= x + (y + z) & x + z &= x \\
(x + y) \cdot z &= x \cdot z + y \cdot z & (a \cdot x) \| y &= a \cdot (x \| y) & (x + y) \| z &= x \| z + y \| z \\
(x \cdot y) \cdot z &= x \cdot (y \cdot z) & (a \cdot x) \| b &= (a \| b) \cdot x & (x + y) \| z &= x \| z + y \| z \\
\delta \cdot x &= \delta & (a \cdot x) \| (b \cdot y) &= (a \| b) \cdot (x \| y) & (x + y) \| z &= x \| z + y \| z \\
\partial_H(a) &= a \text{ if } a \notin H & \partial_H(a) &= a \text{ if } a \in H & \partial_H(x + y) &= \partial_H(x) + \partial_H(y) & \partial_H(x \cdot y) &= \partial_H(x) \cdot \partial_H(y) \\
\partial_H(x + y) &= \partial_H(x) + \partial_H(y) & \partial_H(x \cdot y) &= \partial_H(x) \cdot \partial_H(y) & \partial_H(x + y) &= \partial_H(x) + \partial_H(y) & \partial_H(x \cdot y) &= \partial_H(x) \cdot \partial_H(y)
\end{align*}
\]

Table 1: Axioms of ACP \( (a, b, d \in A \cup \{ 15 \}) \).

The definition of an operational semantics by means of SOS deduction rules is also standard. By means of these rules, we define binary relations \( \xrightarrow{a} \) and unary relations \( \xrightarrow{a} \) on closed terms (for \( a \in A \)). Intuitively, they have the following meaning:

\( x \xrightarrow{a} x' \) means that \( x \) evolves into \( x' \) by executing \( a \).
\[ x \xrightarrow{a} \checkmark \] means that \( x \) successfully terminates upon execution of \( a \).

We present the rules in Table 2. The rules for communication in lines 6 and 7 only hold in case \( a \mid b \) is defined to be an atomic action, so not equal to \( \varepsilon \). The rules are in the so-called path format, see e.g. [4], from which we know that the semantics induced by the rules have some nice properties.

We define (strong) bisimulation equivalence in the standard way, based on these rules. Since the rules are in path format, bisimulation is a congruence for all operators, and the set of closed terms modulo bisimulation turns into an algebra for the signature of ACP.

From [4, 5] we quote the result that the axiomatisation of Table 1 is sound and complete the algebra of closed terms modulo bisimulation.

\[
\begin{align*}
\frac{a \xrightarrow{a} \checkmark}{x \xrightarrow{a} x'} & \quad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} x'} & \quad \frac{x \xrightarrow{a} \checkmark}{x + y \xrightarrow{a} \checkmark} & \quad \frac{y \xrightarrow{a} \checkmark}{x + y \xrightarrow{a} \checkmark} \\
\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y} & \quad \frac{y \xrightarrow{a} y'}{x \xrightarrow{a} \checkmark} & \quad \frac{x \xrightarrow{a} \checkmark}{x \cdot y \xrightarrow{a} y} & \quad \frac{x \xrightarrow{a} \checkmark}{x \parallel y \xrightarrow{a} x} & \quad \frac{x \xrightarrow{a} \checkmark}{x \parallel y \xrightarrow{a} x} \\
\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y} & \quad \frac{y \xrightarrow{a} y'}{x \parallel y \xrightarrow{a} x \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark}{x \parallel y \xrightarrow{a} y} & \quad \frac{x \xrightarrow{a} \checkmark}{x \parallel y \xrightarrow{a} x} \\
\frac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} \\
\frac{x \xrightarrow{a} x', y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} & \quad \frac{x \xrightarrow{a} \checkmark, y \xrightarrow{b} y'}{x \parallel y \xrightarrow{a|b} x' \parallel y'} \\
\frac{x \xrightarrow{a} \checkmark, a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')} & \quad \frac{x \xrightarrow{a} \checkmark, a \notin H}{\partial_H(x) \xrightarrow{a} \checkmark} \\
\end{align*}
\]

Table 2: Deduction rules for ACP \((a, b \in A, a \mid b \in A)\).

In the sequel, we talk about the deadlock behaviour of processes. For our purposes, the following definition suffices.
Definition 1 We define when a term has a deadlock. We give an operational definition and an equational definition. We refer to [1] for more information, and a proof that both notions coincide.

- Let \( t \) be a closed term. We say \( t \) has a deadlock if starting from \( t \) we can execute a number of steps to a process \( t' \) that cannot execute a step at all.

- Inductively, we define a set of closed terms \( DL \) as follows:
  1. For each atom \( a \), \( a \cdot \delta \in DL \)
  2. If \( t \in DL \), and \( a \) is an atom and \( s \) any closed term, then \( a \cdot t \in DL \) and \( t + s \in DL \).

Now let \( t \) be a closed term. We say \( t \) has a deadlock if either \( t \) is \( \delta \), or there is a term \( t' \in DL \) that equals \( t \).

To give an example, terms \( \delta \) and \( ab + a\delta \) have a deadlock, and term \( a(b + \delta) \) does not have a deadlock.

We will require two extra ingredients in the sequel: priorities, introduced in the ACP setting in [3], and no exit iteration, introduced in [9] as a special case of the binary iteration operator of [6]. The priority operator \( \theta \) is parametrized by a partial ordering on atomic actions, and in a choice context, summands that start with an atomic action of high priority will block summands starting with an atomic action of lower priority. The axiomatization uses an auxiliary operator \( \vartriangleright \) called unless.

The no exit iteration operator \( \cdot \omega \) will execute its argument an unbounded number of times. There is no exit of the loop. We present the respective axiomatizations in Table 3. The axiom NEI is the defining axiom of no exit iteration, and the conditional axiom RSP\( \omega \) is the Recursive Specification Principle for no exit iteration, stating that restricted recursive equations of the form \( y = x \cdot y \) have a unique solution, viz. \( x^\omega \). Note that our theory does not contain an empty or skip process, which ensures that no process can terminate immediately, and so these equations are all guarded.

Also here, it is straightforward to give operational rules, see Table 4. The deduction rules for priority use so-called negative premises. Nevertheless, the rules are well-defined, and again induce an algebra of closed terms modulo bisimulation congruence. This result uses the fact that the rules are in so-called panth format, see [4]. In [4], we can also find the proof that the axioms for ACP plus priorities form a sound and complete axiomatisation for the model of closed terms modulo bisimulation equivalence.

In the case of no exit iteration, [9] proves the axioms given are sound for the model of closed terms modulo bisimulation equivalence. He also shows completeness for the restricted theory of BPA, that has no parallel composition or encapsulation. We leave as an open question, whether completeness also holds for the full ACP theory.
\begin{align*}
\text{P1} & \quad \theta(a) = a \\
\text{P2} & \quad \theta(x \cdot y) = \theta(x) \cdot \theta(y) \\
\text{P3} & \quad \theta(x + y) = \theta(x) \cdot \theta(y) \cdot x \\
\text{P4} & \quad \theta(x + y) = (x \cdot y) \cdot z \\
\text{P5} & \quad x^\omega = x \cdot (x^\omega) \\
\text{P6} & \quad y = x \cdot y \implies y = x^\omega 
\end{align*}

Table 3: Axioms for priority and no exit iteration \((a, b \in A \cup \{\delta\})\).

\[
\begin{array}{ccc}
\frac{x \xrightarrow{a} x', \forall b > a \ x \not\rightarrow b}{\theta(x) \xrightarrow{a} \theta(x')} & \quad & \frac{x \xrightarrow{a} \sqrt{y}, \forall b > a \ y \not\rightarrow b}{\theta(x) \xrightarrow{a} \sqrt{y}} \\
\frac{x \xrightarrow{a} x', \forall b > a \ y \not\rightarrow b}{x \cdot y \xrightarrow{a} x'} & \quad & \frac{x \xrightarrow{a} \sqrt{y}, \forall b > a \ y \not\rightarrow b}{x \cdot y \xrightarrow{a} x'} \\
\frac{x \xrightarrow{a} x', \forall b > a y \not\rightarrow b}{x^\omega \xrightarrow{a} x' \cdot (x^\omega)} & \quad & \frac{x \xrightarrow{a} \sqrt{y}, \forall b > a \ y \not\rightarrow b}{x^\omega \xrightarrow{a} x^\omega}
\end{array}
\]

Table 4: Deduction rules for priority and no exit iteration \((a, b \in A)\).

### 3 Durational Actions

Next, in order to introduce split and ST bisimulation equivalence, we will assume we also have actions that take time to execute, non-atomic actions. We will call these **durational actions**. To ease notation, we will assume that for each core atom \(t\), we have a durational action \(\hat{t}\). Alternatively, we write \(\text{dur}(t)\).

Having introduced this new signature element, the question is how to extend the axiomatisation of ACP to include durational actions. In particular, we cannot have \(\hat{t} \parallel \hat{u} = \hat{t} \cdot \hat{u} + \hat{u} \cdot \hat{t}\), so the semantics of durational actions will not be purely interleaving.

Here, we use the notion that each durational action is composed of two atomic actions, namely the beginning of the action followed by the end of the action. This is usually notated as follows:

\[
\hat{t} = t^+ \cdot t^-
\]  

(DUR)
In the operational semantics, we add the following extra deduction rule:

\[ \overset{\text{\(\widehat{t} \rightarrow\)}}{+}, t^- \]

where \( t^+, t^- \in A \) are new atomic actions (not core atoms). For \( t^+ \), we also write \( t^0 \) or simply \( t \), and for \( t^- \) we can write \( t^1 \).

Adding either of these equations to \( ACP \) with durational actions gives us split semantics, as is well-known from [12], [8], [11].

Let us define exactly what we mean by the term *split bisimulation semantics*. First, we give the standard definition for the non-interleaving semantics of old \( ACP \)-terms, originally due to [13], next we give a definition on terms that may involve durational actions.

**Definition 2**

1. If \( x \) is a closed \( ACP \)-term over the set of core atomic actions \( CA \), then \( \widehat{x} \) is this term with each core atom \( t \) replaced by \( \widehat{t} \). Now let \( x, y \) be two closed \( ACP \)-terms.
   
   We define \( x \approx_{\text{split}} y \) iff \( \widehat{x} \sim \widehat{y} \), \( \widehat{x} \) is bisimulation equivalent to \( \widehat{y} \) (using the extra rule displayed above).

2. Let \( x, y \) be closed terms over the theory of \( ACP \) with durational actions. We define \( x \approx_{\text{split}} y \), \( x \) is *split bisimulation equivalent* to \( y \), if \( x \sim y \) (using the extra rule displayed above).

In the sequel, we will always use the second definition.

The problem with split semantics that is often cited, is that it can be unknown which end matches a certain beginning. To illustrate the problem, consider the following term:

\[ \overline{r} \parallel \overline{r} \cdot t \parallel \overline{s} \cdot u \parallel \overline{s}. \]

Suppose \( \overline{r} | \overline{s} = \overline{c} \) is a defined communication, which we take to mean that \( r^+ | s^+ = c^+ \) and \( r^- | s^- = c^- \). Further, we have the regular communication \( t \parallel u = v \). Now, in the displayed term, we can execute a \( c^+ \) resulting from a communication between the first and third component and a second \( c^+ \) resulting from a communication between the second and fourth component. We expect that the communication \( v \) can only take place upon completion of both communications. However, we can “criss-cross” the communications, and execute a \( c^- \) resulting from the second and third components, and later another \( c^- \) resulting from the first and fourth components. After the first \( c^- \), already \( v \) becomes possible, contrary to intuition. In the next section, we will see that such “criss-cross” behaviour can alter the deadlock behaviour of a process.

If, on the other hand, we always want to be able to match an ending with the correct beginning, we have to employ additional machinery in order to do so. An axiomatisation doing this can be found in [8]. Here, we present an alternative to [8] that is easier to understand but basically follows the same intuition.
We add to the signature a unary operator \( \text{hps} \), the history pointer shift, with auxiliary operators \( \text{hps}^n \) for each \( n \geq 0 \). Further, for each core atom \( t \), there are new atomic actions \( t^{\text{hp}(n)} \) for \( n \geq 0 \) and durational actions \( \text{dur}(t) \), or, alternatively, if no confusion arises, atomic actions \( t^n \) and durational actions \( t \). In split semantics, the history pointer shift operator will be just the identity.

We replace axioms CM2 and CM3 of ACP by the axioms CM2HP and CM3HP shown in Table 5 below, and add the other axioms in this table.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a \parallel x = a \cdot \text{hps}(x) )</td>
<td>CM2HP</td>
<td>( \text{hps}(x) = \text{hps}_0(x) )</td>
</tr>
<tr>
<td>( (a \cdot x) \parallel y = a \cdot (x \parallel \text{hps}(y)) )</td>
<td>CM3HP</td>
<td>( \text{hps}_n(\delta) = \delta )</td>
</tr>
<tr>
<td>( \hat{t} = t^0 \cdot t^1 )</td>
<td>DUR</td>
<td>( \text{hps}_n(t^m) = t^{m+1} ) if ( n &lt; m )</td>
</tr>
<tr>
<td>( r^n</td>
<td>s^n = c^n ) if ( r \mid s = c )</td>
<td>CDUR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{hps}_n(x + y) = \text{hps}_n(x) + \text{hps}_n(y) )</td>
</tr>
</tbody>
</table>

Table 5: Axioms of durational actions with history pointer \((a \in A \cup \{\delta\}, t, r, s, c \in CA)\).

The SOS rules for the history pointer are straightforward. See Table 6.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{t} \overset{0}{\rightarrow} t^1 )</td>
<td></td>
</tr>
<tr>
<td>( x \overset{t}{\rightarrow} x' )</td>
<td>( \text{hps}<em>n(x) \overset{t}{\rightarrow} \text{hps}</em>{n+1}(x') )</td>
</tr>
<tr>
<td>( x \overset{m}{\rightarrow} x' ), ( m \leq n )</td>
<td>( \text{hps}<em>n(x) \overset{m}{\rightarrow} \text{hps}</em>{n+1}(x') )</td>
</tr>
<tr>
<td>( x \overset{m}{\rightarrow} x' ), ( m &gt; n )</td>
<td>( \text{hps}<em>n(x) \overset{m}{\rightarrow} \text{hps}</em>{n+1}(x') )</td>
</tr>
<tr>
<td>( \text{hps}<em>n(x) \overset{m}{\rightarrow} \text{hps}</em>{n+1}(x') )</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Deduction rules for history pointer \((t \in CA)\).

Now we have the following definition of ST bisimulation semantics, originally defined in [10]. This follows the definition of split bisimulation semantics above.
Definition 3  1. Let \( x, y \) be two closed ACP-terms (not containing durational actions or history pointer). We define \( x \sim_{ST} y \) iff \( \bar{x} \equiv \bar{y} \), \( \bar{x} \) is bisimulation equivalent to \( \bar{y} \) (using the rules of Table 2 and Table 6).

2. Let \( x, y \) be closed terms over the theory of ACP with durational actions and history pointer. We define \( x \approx_{ST} y \), \( x \) is ST bisimulation equivalent to \( y \), if \( x \equiv y \) (using the rules of Table 2 and Table 6).

Again, in the sequel we will only use the second part of this definition. Now we have the following proposition.

**Proposition 4**  1. Let \( x, y \) be closed terms over the theory of ACP with durational actions. Then \( x \approx_{split} y \) iff \( x = y \) is derivable from the axioms of ACP and DUR, CDUR, i.e. the axiomatisation is sound and complete for the operational model.

2. Let \( x, y \) be closed terms over the theory of ACP with durational actions and history pointer. Then \( x \approx_{ST} y \) iff \( x = y \) is derivable from the axioms of ACP, minus CM2, CM3, plus the axioms of Table 5. Again, the axiomatisation is sound and complete for the operational model.

**Proof**  For the second part, the proof of completeness of the axioms for ACP plus durational actions and history pointer shift for the bisimulation model induced by the operational rules of Table 6 follows the recipe of [4]. We outline the key elements of this proof. First, we define the set of basic terms. These are closed terms built with just alternative composition and prefix multiplication. Then, we show that the history pointer operator can be eliminated from all closed terms by means of rewriting to a basic term. By using this elimination theorem, together with a general conservative extension theorem based on the path format of our deduction rules, the proof reduces to the completeness proof of ACP (see [5, 4]).

The remainder of the proof is straightforward, given the many examples in [4].

4  Deadlock in Split but not in ST

We prove that there exists a simple process, that deadlocks in split semantics, but not in ST semantics.

**Proposition 5** There is a closed term in the signature of ACP with durational actions and priority operator that has a deadlock in split semantics but not in ST bisimulation semantics.

**Proof**  We use the communication function that has \( r_i \mid s_i = s_i \mid r_i = c_i \) as the only defined communications on core atoms \( (i = 1, \ldots, 9) \). The set \( H \) contains all \( r_1, s_1 \) actions,
but no $c_i$ actions, and the priority relation gives all core atoms priority over all other atomic actions.

We consider the following term:

$$\theta \circ \partial_H([r_1 \cdot (r_2 r_3 + r_4 r_5) \parallel r_6] \parallel [r_1 \cdot (r_2 r_7 + r_4 r_8) \parallel r_9] \parallel [(s_1 \cdot s_2) \parallel (s_6 s_3 + s_9 s_7)] \parallel [(s_1 \cdot s_4) \parallel (s_6 s_5 + s_9 s_8)])$$

After a couple of steps using the axioms in Table 5, we see that this term evolves into:

$$c^0_1 c_6 (c^0_1 c_2 c_3 \cdot \theta \circ \partial_H([r_1 \cdot (r_2 r_7 + r_4 r_8) \parallel r_9] \parallel [(s_1 \cdot s_4) \parallel (s_6 s_5 + s_9 s_8)]) +$$

$$+ c^0_1 c_9 \cdot \theta \circ \partial_H([r^1_1 \cdot (r_2 r_3 + r_4 r_5)] \parallel [r^1_1 \cdot (r_2 r_7 + r_4 r_8)] \parallel [s^1_1 s_2 \parallel s_3] \parallel [s^1_1 s_4 \parallel s_8]) +$$

$$+ c^0_1 c_6 (c^1_1 c_4 c_5 \cdot \theta \circ \partial_H([r_1 \cdot (r_2 r_7 + r_4 r_8) \parallel r_6] \parallel [(s_1 \cdot s_4) \parallel (s_6 s_5 + s_9 s_8)]) +$$

$$+ c^0_1 c_9 \cdot \theta \circ \partial_H([r^1_1 \cdot (r_2 r_3 + r_4 r_5)] \parallel [r^1_1 \cdot (r_2 r_7 + r_4 r_8)] \parallel [s^1_1 s_2 \parallel s_7] \parallel [s^1_1 s_4 \parallel s_5]) +$$

$$+ c^0_1 c_6 (c^1_1 c_2 c_7 \cdot \theta \circ \partial_H([r_1 \cdot (r_2 r_3 + r_4 r_5) \parallel r_6] \parallel [(s_1 \cdot s_4) \parallel (s_6 s_5 + s_9 s_8)]) +$$

$$+ c^0_1 c_9 \cdot \theta \circ \partial_H([r^1_1 \cdot (r_2 r_3 + r_4 r_5)] \parallel [r^1_1 \cdot (r_2 r_7 + r_4 r_8)] \parallel [s^1_1 s_2 \parallel s_3] \parallel [s^1_1 s_4 \parallel s_8]) +$$

At this point, behaviour in the two semantics will start to diverge. For instance, in the second line, the first argument of the merge can communicate with the third, and the second with the fourth, if we obey the history pointers. In split semantics, also communication between the second and third argument, and between the first and the fourth argument, is possible.

In ST semantics, we obtain the following:

$$c^0_1 c_6 (c^0_1 c_2 c_3 c^0_1 c_6 c^0_1 c_2 c_4 c_8 + c^0_1 c_6 (c^1_1 c_2 c_7 c^0_1 c_2 c_4 c_8 + c^1_1 c_4 c_8 c^1_1 c_2 c_3)) +$$

$$+ c^0_1 c_9 (c^1_1 c_4 c_5 c^1_1 c_9 c^0_1 c_2 c_7 + c^0_1 c_9 (c^1_1 c_4 c_5 c^1_1 c_2 c_7 + c^1_1 c_2 c_7 c^1_1 c_4 c_5)) +$$

$$+ c^0_1 c_9 (c^1_1 c_2 c_7 c^1_1 c_6 c^1_1 c_2 c_4 c_5 + c^0_1 c_6 (c^1_1 c_2 c_7 c^1_1 c_4 c_5 + c^1_1 c_4 c_5 c^1_1 c_2 c_7)) +$$

$$+ c^0_1 c_6 (c^1_1 c_4 c_8 c^1_1 c_6 c^1_1 c_2 c_3 + c^1_1 c_6 (c^1_1 c_4 c_8 c^1_1 c_2 c_3 + c^1_1 c_2 c_3 c^1_1 c_4 c_8)).$$

We see there is no deadlock in ST semantics. In split semantics, we obtain extra summands due to criss-cross behaviour, and expansion gives:
\[ c_1^+c_6(c_1^-c_2c_3c_1^+c_9c_1^-c_4c_8 + c_1^+c_9(c_1^-c_2c_3c_1^-c_4c_8 + c_1^-c_4c_8c_1^-c_2c_3 + \\
+ c_1^+c_4c_1^-c_2\delta + c_1^-c_2c_1^-c_4\delta)) + \\
+ c_1^+c_8(c_1^-c_4c_5c_1^+c_9c_1^-c_2c_7 + c_1^+c_8(c_1^-c_4c_5c_1^-c_2c_7 + c_1^-c_2c_7c_1^-c_4c_5 + \\
+ c_1^-c_4c_1^-c_2\delta + c_1^-c_2c_1^-c_4\delta)) + \\
+ c_1^+c_9(c_1^-c_2c_7c_1^+c_6c_1^-c_4c_3 + c_1^+c_9(c_1^-c_2c_7c_1^-c_4c_3 + c_1^-c_4c_3c_1^-c_2c_7 + \\
+ c_1^-c_4c_1^-c_2\delta + c_1^-c_2c_1^-c_4\delta)) + \\
+ c_1^+c_3(c_1^-c_4c_8c_1^+c_6c_1^-c_2c_3 + c_1^+c_3(c_1^-c_4c_8c_1^-c_2c_3 + c_1^-c_2c_3c_1^-c_4c_8 + \\
+ c_1^-c_4c_1^-c_2\delta + c_1^-c_2c_1^-c_4\delta)). \]

Here, the deadlocks in the first and the last terms represent the sub-process

\[ \theta \circ \partial_H(r_5 \parallel r_7 \parallel s_3 \parallel s_8) \]

and the deadlocks in the middle two terms are due to the remainder

\[ \theta \circ \partial_H(r_3 \parallel r_8 \parallel s_7 \parallel s_8). \]

We leave as an open problem, whether or not such an example can be found in a smaller signature. In our investigations, we found that both left merge and priority operator seem essential.

5 Deadlock in ST but not in Split

We prove that there exists a simple process, that deadlocks in ST semantics, but not in split semantics.

**Proposition 6** There is a closed term in the signature of ACP with durational actions, priority operator and no exit iteration operator, that has a deadlock in ST semantics but not in split semantics.

**Proof** We use the same communication function as in the previous section. The set \( H \) contains all \( r_i, s_i \) actions but in addition the action \( c_3 \). The core atoms have priority over other atomic actions within the scope of the priority operator. The following term shows deadlock behaviour in ST semantics but not in split semantics.
\[ \partial_H(\theta(r_1 \parallel r_2 \cdot r_3^\omega) \parallel s_1^\omega \parallel s_2 \cdot r_1 \parallel s_2 \cdot \theta(s_1 \parallel s_3^\omega)). \]

In ST semantics, we see that this term equals
\[ c_1^0 c_2 c_3 c_1^0 \cdot \partial_H(\theta(r_1^3 \parallel r_3^\omega) \parallel s_1^3 \cdot u^\omega \parallel r_1^3 \parallel \theta(s_1^3 \parallel s_3^\omega)). \]

In the first and fourth argument of the merge, the actions along port 3 take precedence over the actions along port 1. However, communication along port 3 cannot occur as this is blocked by the encapsulation operator. For the second and third argument, the history pointers do not match. Thus, in ST semantics, no further action is possible, so deadlock ensues. In split semantics, communication between the second and third term is still possible, after which an unbounded amount of \( u \)'s is possible, so the result becomes
\[ c_1^+ c_2 c_3 c_1^- \cdot u^\omega. \]

Again, it is an open question whether an example can be found in a smaller signature. In the current example, we note the essential use of left merge, priority and iteration.

6 Durational Actions for Message Passing

As an application of the theory above, we sketch communication between two devices, when communication is not instantaneous but has a duration. In order to describe this, we need the early read and process prefix mechanisms of [2]. We briefly recall this mechanism. For the sake of this discussion, we assume a fixed finite set \( D \) of messages to be communicated.

First of all, regular communication assumes core atoms \( r_i(d), s_i(d), c_i(d) \), for each communication port \( i \in P, \) message \( d \in D \), with the only defined communication actions \( r_i(d) \mid s_i(d) = s_i(d) \mid r_i(d) = c_i(d) \) on core atoms.

Next, we add a new set of atomic actions \( er_i(v); (i \in P, v \) a placeholder ranging over \( B, \) and \( B \subseteq D \), the early read atoms. The working of these can only be appreciated fully together with the process prefix operator ;. We display the axioms that we need, taken from [2], in Table 7. In this table, we use the notation \( x[d/v] \) for the term \( x \) where all occurrences of \( v \) are substituted by \( d \).

When we define deduction rules, we have to realize that the first deduction rule of Table 2 also holds for the new atoms \( er_i^B(v) \). Further, we have the rules in Table 8.

Now, we can imagine that in order for communication to take place, first the connection needs to be established. Then, communication becomes durational, and we can define the following new durational actions (here, \( r_i, s_i, c_i \) are core atoms).
\[
e_{r_i}^B(v): x = \sum_{d \in B} r_i(d) \cdot x[d/v] \quad \text{PRE1}
\]
\[
a; x = a \cdot x \text{ otherwise, if } a \neq e_{r_i}^B(v) \quad \text{PRE2}
\]
\[
(x + y); z = x; z + y; z \quad \text{PRE3}
\]
\[
(x \cdot y); z = x; (y; z) \quad \text{PRE4}
\]

Table 7: Axioms of early read and process prefix \((a \in A \cup \{\delta\})\).

\[
\begin{array}{c|c}
\hline
x \xrightarrow{e_{r_i}^B(v)} x' & x \xrightarrow{a} x' \\
\hline
x; y \xrightarrow{r_i(d)} (x'; y)[d/v] & x; y \xrightarrow{a} (x'; y) \\
x \xrightarrow{e_{r_i}^B(v)} \sqrt{,} & x \xrightarrow{a} \sqrt{,}
\end{array}
\]

Table 8: Deduction rules for early read and process prefix \((a \in A, a \neq e_{r_i}^B(v))\).

\[
\begin{align*}
der_{r_i}^B(v) &= r_i \cdot (e_{r_i}^B(v))^{hp(1)} \\
ds_i(d) &= s_i \cdot (s_i(d))^{hp(1)} \\
dc_i(d) &= c_i \cdot (c_i(d))^{hp(1)}.
\end{align*}
\]

Now, we consider the following process \(X\). Take \(D = \{0, \ldots, 9\}\) to be the set of digits, and \(B = \{0, 2, 4, 6, 8\}\) the set of even digits. We have just one communication port \(i\), and let \(H\) contain the \(r_i, s_i, r_i(d), s_i(d)\) actions. Finally, \(P, Q, R\) are given processes.

\[
X = \partial_H(der_i^B(v); P \parallel ds_i(5) \cdot Q \parallel ds_i(6) \cdot R)
\]

We calculate:

\[
X = \partial_H((r_i \cdot er_i^B(v)^1) ; P \parallel s_i \cdot s_i(5)^1 \cdot Q \parallel s_i \cdot s_i(6)^1 \cdot R) =
\]
\[
X = \partial_H(r_i \cdot (er_i^B(v)^1) ; P \parallel s_i \cdot s_i(5)^1 \cdot Q \parallel s_i \cdot s_i(6)^1 \cdot R) =
\]
\[
= c_i \cdot \partial_H(\sum_{d \in B} r_i(d)^1 \cdot P[d/v] \parallel s_i(5)^1 \cdot Q \parallel ds_i(6) \cdot R) +
\]

\[13\]
We see deadlock is possible. On the other hand, we can consider the actions $dur(s_i(d)) = s_i(d) \cdot s_i(d)$, $dur(c_i(d)) = c_i(d) \cdot c_i(d)$ and $dur(e_f^B(v)) = e_f^B(v) \cdot e_f^B(v)$. This last action is a bit strange, as $v$ will be bound two times. In the following calculation, we cannot start with a communication between the first and the second term, as the durational read cannot accept a message 5.

$$Y = \partial_H(dur(e_f^B(v)); P \parallel dur(s_i(5)) \cdot Q \parallel dur(s_i(6)) \cdot R) =$$

$$= \partial_H(e_f^B(v) \cdot (e_f^B(v)) \cdot P \parallel s_i(5) \cdot s_i(6) \cdot Q \parallel s_i(6) \cdot s_i(6) \cdot R) =$$

$$= \partial_H(\sum_{d \in B} r_i(d) \cdot (e_f^B(v)) \cdot P \parallel s_i(5) \cdot s_i(5) \cdot Q \parallel s_i(6) \cdot s_i(6) \cdot R) =$$

$$= \partial_H(\sum_{d \in B} r_i(d) \cdot (e_f^B(v)) \cdot P \parallel s_i(5) \cdot Q \parallel s_i(6) \cdot R) =$$

$$= \partial_H(c_i(6) \cdot (e_f^B(v)) \cdot P \parallel s_i(5) \cdot Q \parallel s_i(6) \cdot R) =$$

$$= \partial_H(c_i(6)) \cdot (e_f^B(v)) \cdot P \parallel s_i(5) \cdot Q \parallel R).$$

We see again a difference in deadlock behaviour. It will be clear that the behaviour of action $dur(e_f^B(v))$ is peculiar. On the other hand, we see $der_f^B(v)$ as a reasonable programming primitive.

7 Conclusion

We investigated two non-interleaving process semantics, split semantics and ST bisimulation semantics, in the framework of the process algebra ACP. We found that atomic actions and durational actions can coexist. We give an easy-to-understand axiomatisation that is complete in either semantics.

We found that the two semantics are very different: there are closed terms that deadlock in one semantics but not in the other. This has enormous impact for the specification and verification of processes.

We considered as an application the description of message passing by means of durational actions. Here again, we find a difference in deadlock behaviour, depending on the moment of variable binding.
References


Computing Science Reports

In this series appeared:

96/01 M. Voorhoeve and T. Basten
96/02 P. de Bra and A. Aerts
96/03 W.M.P. van der Aalst
96/04 S. Mauw
96/05 T. Basten and W.M.P. v.d. Aalst
96/06 W.M.P. van der Aalst and T. Basten
96/07 M. Voorhoeve
96/08 A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk
96/09 F. Dignum, H. Weigand, E. Verharen
96/10 R. Bloo, H. Geuvers
96/11 T. Laan
96/12 F. Kamareddine and T. Laan
96/13 T. Borghuis
96/14 S.H.J. Bos and M.A. Reniers
96/15 M.A. Reniers and J.J. Vereijken
96/16 E. Boiten and P. Hoogendijk
96/17 P.D.V. van der Stok
96/18 M.A. Reniers
96/20 L. Feijs
96/21 L. Bijlsma and R. Nederpelt
96/22 W.M.P. van der Aalst
96/23 M.C.A. van de Graaf and G.J. Houben
96/24 M. Voorhoeve and W. van der Aalst
96/25 M. Vaccari and R.C. Backhouse
97/01 B. Knaack and R. Gerth
97/02 J. Hooman and O. v. Roosmalen
97/03 J. Blanco and A. v. Deursen
97/04 J.C.M. Baeten and J.A. Bergstra
97/05 J.C.M. Baeten and J.J. Vereijken
97/06 M. Franssen
97/07 J.C.M. Baeten and J.A. Bergstra

Department of Mathematics and Computing Science
Eindhoven University of Technology

Process Algebra with Autonomous Actions, p. 12.
Multi-User Publishing in the Web: DreSS, A Document Repository Service
Station, p. 12.
Example specifications in phi-SDL.
A Process-Algebraic Approach to Life-Cycle Inheritance
Inheritance = Encapsulation + Abstraction, p. 15.
Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
Structural Petri Net Equivalence, p. 16.
OODB Support for WWW Applications: Disclosing the internal structure of
AUTOMATH and Pure Type Systems, p. 30.
A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
Precean Tense Logics in Modal Pure Type Systems, p. 61
The $I^2$ C-bus in Discrete-Time Process Algebra, p. 25.
Completeness in Discrete-Time Process Algebra, p. 139.
Nested collections and polytypism, p. 11.
Real-Time Distributed Concurrency Control Algorithms with mixed time con­
straints, p. 71.
Static Semantics of Message Sequence Charts, p. 71
Algebraic Specification and Simulation of Lazy Functional Programs in a concur­rent Environment, p. 27.
Designing Effective Workflow Management Processes, p. 22.
Structural Characterizations of sound workflow nets, p. 22.
Conservative Adaption of Workflow, p. 22
Deriving a systolic regular language recognizer, p. 28
A Discretisation Method for Asynchronous Timed Systems
Basic Conditional Process Algebra, p. 20.
Discrete Time Process Algebra: Absolute Time, Relative Time and Parametric
Bounded Stacks, Bags and Queues, p. 15.
97/08  P. Hoogendijk and R.C. Backhouse  When do datatypes commute? p. 35.


97/13  W.M.P. van der Aalst  Exploring the Process Dimension of Workflow Management, p. 56.

97/14  J.F. Groote, F. Morin and J. Springintveld  A computer checked algebraic verification of a distributed summation algorithm, p. 28

97/15  M. Franssen  λP: A Pure Type System for First Order Logic with Automated Theorem Proving, p. 35.

97/16  W.M.P. van der Aalst  On the verification of inter-organizational workflows, p. 23

97/17  M. Vaccari and R.C. Backhouse  Calculating a Round-Robin Scheduler, p. 23.


97/19  M. Reniers (artikel volgt)


98/02  M. Voorhoeve  State / Event Net Equivalence.