Some simple calculations in relative discrete time process algebra

Some simple calculations in relative discrete time process algebra

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

Abstract

We do some simple calculations involving buffers in discrete time process algebra with relative timing.

Note: Dedicated to prof.dr. F.E.J. Kruseman Aretz, on the occasion of his 'afscheidscollege'.

1 Introduction

Any formal language should enable formal analysis. As Kruseman Aretz already emphasised in [6], it is essential to do precise calculations, perform mathematics, with a formal language, and a well-designed language should facilitate such calculations. In this note, we do some simple calculations in the formal language of relative discrete time process algebra.

The process algebra ACP [3] describes the main features of imperative concurrent programming without explicit mention of time. Implicitly, time is present in the interpretation of sequential composition: in $p \cdot q$ the process p must be executed before q. A quantative view on the relation between process execution and progress of time is absent in ACP, however. In recent years, process algebras have been developed that provide standardised features to incorporate a quantative view on time. On the one hand, we can represent time by means of non-negative reals, and have time stamps on actions. On the other hand, we can divide time in slices indexed by natural numbers, have an implicit or explicit time stamping mechanism that provides each action with the time slice in which it occurs and have a time order within each time slice only. We use the phrase *discrete time process algebra* if an enumeration of time slices is used.

In [1] ACP was extended to a discrete time process algebra. Here, we use discrete time process algebra with *relative timing*, where timing refers to the execution of the previous action. We use the so-called *two-phase version*, where the passage of time and the execution of actions is separated. Another version of discrete time process algebra uses *absolute timing*, where all timing refers to an absolute clock. Finally, we have discrete process algebra with *parametric timing*, where absolute and relative timing are integrated (see [1]).

2 Discrete Time Process Algebra with Relative Timing

We present axioms for discrete time process algebra with relative timing. We base ourselves on [1], but use the slightly optimised presentation of [2].

2.1 Basic Process Algebra

We assume we have given a (finite) set of atomic actions A. This set is a parameter of the theory to be presented.

With $\underline{a}$ we denote the process that will execute atomic action a in the current time slice, the time slice in which it is initialised. So, if $\underline{a}$ is enabled during slice 7, then a will be performed in the course of time slice 7. With the operator σ_{rel} processes can be delayed one time slice. So if the process $\sigma_{\mathrm{rel}}(\sigma_{\mathrm{rel}}(\underline{a}))$ is initialised in slice 5, then a will be executed in slice 7.

The signature of $\mathrm{BPA}_{\mathrm{drt}}$ has constants $\underline{a}$ (for $a \in A$), denoting a in the current time slice, and $\underline{\delta}$, denoting a deadlock at the end of the current time slice. Furthermore, we have the immediate deadlock constant $\dot{\delta}$, denoting immediate and catastrophic deadlock. The operators are alternative ($+$) and sequential composition ($\cdot$), and the *relative discrete time unit delay* σ_{rel}. The process $\sigma_{\mathrm{rel}}(x)$ will start after one clock tick, i.e. in the next time slice. Note that the operator σ_{rel} is a constructor: the term $\sigma_{\mathrm{rel}}(\underline{a})$ cannot be reduced. Furthermore, we have the auxiliary operator *current time slice time out* ν_{rel}. The current time slice time out operator disallows an initial time step, it gives the part of a process that starts with an action in the current time slice. This operator will prove useful when we extend $\mathrm{BPA}_{\mathrm{drt}}$ with parallel composition in the next subsection.

Within a time slice, there is no explicit mention of the passage of time, we can see the passage to the next time slice as a clock tick. Thus, the $\underline{a}$ can be called *non-delayable* actions: the action must occur before the next clock tick. In order to add delayable actions to the theory, we have the operator $\lfloor\ \rfloor^{\omega}$, the *unbounded start delay* operator: $\lfloor x \rfloor^{\omega}$ can start the execution of x in the current time slice, or delay unchanged to the next time slice. The defining axiom for this operator takes the form of a recursive equation. The unbounded start delay operator is defined easily in terms of the current time slice time out operator. A delayable action a (atomic action a in any time slice) is then defined as the unbounded start delay of the non-delayable action $\underline{a}$ (axiom ATS). In order to be able to work with recursive equations, e.g. if we want to prove identities concerning the unbounded start delay operator, we need to extend the algebra with a proof principle. We will look at such a principle next.

The axioms of $\mathrm{BPA}_{\mathrm{drt}}$ are given in Table 1. The axiom DRT1 is the *time factorization* axiom: it says that the passage of time by itself cannot determine a choice. However, it is possible that passage of time disables the execution of an action a. This is illustrated by the example $\underline{a} + \sigma_{\mathrm{rel}}(\underline{b})$. If a clock tick occurs the process evolves into $\underline{b}$. Thereby, it has

become impossible to execute the action a: it is disabled by the occurrence of the clock tick. The axiom DCS3 for the current time slice time out operator clearly expresses that all alternatives which cannot perform an action in the current time slice are replaced by $\underline{\delta}$.

$x+y=y+x$	A1	$\sigma_{\mathrm{rel}}(x)+\sigma_{\mathrm{rel}}(y)=\sigma_{\mathrm{rel}}(x+y)$	DRT1
$(x+y)+z=x+(y+z)$	A2	$\sigma_{\mathrm{rel}}(x)\cdot y=\sigma_{\mathrm{rel}}(x\cdot y)$	DRT2
$x+x=x$	A3	$\sigma_{\mathrm{rel}}(\dot{\delta})=\underline{\delta}$	DRT3
$(x+y)\cdot z=x\cdot z+y\cdot z$	A4	$\underline{a}+\underline{\delta}=\underline{a}$	DRT4
$(x\cdot y)\cdot z=x\cdot(y\cdot z)$	A5		
$x+\dot{\delta}=x$	A6ID	$\nu_{\mathrm{rel}}(\dot{\delta})=\dot{\delta}$	DCS1
$\dot{\delta}\cdot x=\dot{\delta}$	A7ID	$\nu_{\mathrm{rel}}(\underline{a})=\underline{a}$	DCS2
		$\nu_{\mathrm{rel}}(\sigma_{\mathrm{rel}}(x))=\underline{\delta}$	DCS3
$\lfloor x\rfloor^{\omega}=\nu_{\mathrm{rel}}(x)+\sigma_{\mathrm{rel}}(\lfloor x\rfloor^{\omega})$	USD	$\nu_{\mathrm{rel}}(x+y)=\nu_{\mathrm{rel}}(x)+\nu_{\mathrm{rel}}(y)$	DCS4
$a=\lfloor\underline{a}\rfloor^{\omega}$	ATS	$\nu_{\mathrm{rel}}(x\cdot y)=\nu_{\mathrm{rel}}(x)\cdot y$	DCS5

Table 1: Axioms of $\mathrm{BPA}_{\mathrm{drt}}$ ($a\in A\cup\{\delta\}$).

2.2 Recursion

A *recursive specification* is a set of equations $E=\{X_i=t_i(X_1,\ldots,X_n):1\leq i\leq n\}$ where $n\geq 1$, the X_i are (formal) variables and the t_i are terms (over $\mathrm{BPA}_{\mathrm{drt}}$), possibly containing variables X_i. A sequence of processes in a certain model of $\mathrm{BPA}_{\mathrm{drt}}$ is called a *solution* of E if all equations of E are valid in the model, if we interpret the variables by the corresponding process.

We call a recursive specification E *guarded* if all occurrences of variables X_i in all right-hand sides t_j are guarded, i.e. preceded by an atomic action or a delay. To be precise, an occurrence of a variable X in term t is guarded if t has a subterm s containing this occurrence of X, and s has one of the forms $\underline{a}\cdot s'$, $a\cdot s'$ or $\sigma_{\mathrm{rel}}(s')$ for some $a\in A$.

The *Recursive Specification Principle RSP* says that a guarded recursive specification has at most one solution. This principle was introduced in [5] and has proven to be very useful in system verifications. Specialised to the recursive equation defining the unbounded start delay, RSP instantiates as shown in Table 2. This special form of RSP will be called RSP(USD) in the sequel.

2.3 Structured Operational Semantics

We refer to [1] for an operational semantics for $\mathrm{BPA}_{\mathrm{drt}}$. The set of bisimulation equivalence classes of closed terms then gives a model for $\mathrm{BPA}_{\mathrm{drt}}$, thereby showing consistency of the

$y = \nu_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(y) \Rightarrow y = \lfloor x \rfloor^\omega$	RSP(USD)

Table 2: RSP for Unbounded Start Delay.

theory. We can also provide action rules in order to deal with recursion. Then, we obtain a model in which every recursive specification has a solution, and in which every guarded recursive specification has a unique solution (thus, satisfying RSP). For further details, see [7].

Proposition 2.1 The following identities are derivable from $\mathrm{BPA}_{\mathrm{drt}}$ + RSP(USD):

1. $\sigma_{\mathrm{rel}}(x) = \sigma_{\mathrm{rel}}(x) + \underline{\delta}$
2. $\lfloor \dot{\delta} \rfloor^\omega = \delta$
3. $\lfloor x + y \rfloor^\omega = \lfloor x \rfloor^\omega + \lfloor y \rfloor^\omega$
4. $\lfloor x \cdot y \rfloor^\omega = \lfloor x \rfloor^\omega \cdot y$
5. $\lfloor \sigma_{\mathrm{rel}}(x) \rfloor^\omega = \delta$
6. $a = \underline{a} + \sigma_{\mathrm{rel}}(a)$
7. $\nu_{\mathrm{rel}}(a) = \underline{a}$
8. $\lfloor a \rfloor^\omega = a$
9. $a + \delta = a$
10. $\delta \cdot x = \delta$.

Proof 1. $\sigma_{\mathrm{rel}}(x) = \sigma_{\mathrm{rel}}(x + \dot{\delta}) = \sigma_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(\dot{\delta}) = \sigma_{\mathrm{rel}}(x) + \underline{\delta}$; 2. Use 1 and RSP(USD); 3. Use DCS4 and RSP(USD); 4,5. Likewise, with DCS5,3; 6. By definition; 7. Use 6; 8. Use 7; 9. $a + \delta = \lfloor \underline{a} \rfloor^\omega + \lfloor \underline{\delta} \rfloor^\omega = \lfloor \underline{a} + \underline{\delta} \rfloor^\omega = \lfloor \underline{a} \rfloor^\omega = a$; 10. Like 9.

2.4 Parallel Composition

We extend the theory $\mathrm{BPA}_{\mathrm{drt}}$ with parallel composition, with or without communication, as in [2]. The additional syntax has binary operators $\|$ (merge), $\lfloor\!\lfloor$ (left merge) and $|$ (communication merge), and unary operators ∂_H (encapsulation operator, for $H \subseteq A$). The process $x \| y$ interleaves the behaviours of x and y (with possible synchronisation on communication actions), but it is forced to synchronise on time steps, i.e. a clock tick is executed by both processes at the same time, or if this is not possible, no clock tick can occur. We assume given a partial, commutative and associative communication function $\gamma : A \times A \to A$. The additional axioms are given in Table 3. We obtain a model satsifying RSP for the extended theory $\mathrm{ACP}_{\mathrm{drt}}$ along the same lines as before.

In the sequel, we will only use the so-called *standard communication function.* Suppose we have given two finite sets, the set of messages or data D, and the set of ports P. For each $d \in D$ and $i \in P$, we have atomic actions $ri(d)$, $si(d)$, $ci(d)$ (denoting *receive, send* and *communicate* d along i) and the only defined communications are $\gamma(ri(d), si(d)) = \gamma(si(d), ri(d)) = ci(d)$.

Proposition 2.2 We derive the following identities for time free atoms in $\mathrm{ACP}_{\mathrm{drt}}$+ RSP(USD):

$x \parallel y = x \between y + y \between x + x \mid y$	CM1	$\underline{a} \mid \underline{b} = \underline{\gamma(a,b)}$ if γ defined	DRTCF1
$\underline{a} \between (x + \underline{\delta}) = \underline{a} \cdot (x + \underline{\delta})$	DRTCM2	$\underline{a} \mid \underline{b} = \underline{\delta}$ otherwise	DRTCF2
$\underline{a} \cdot x \between (y + \underline{\delta}) = \underline{a} \cdot (x \parallel (y + \underline{\delta}))$	DRTCM3	$\dot{\delta} \between x = \dot{\delta}$	LMID1
$(x + y) \between z = x \between z + y \between z$	CM4	$x \between \dot{\delta} = \dot{\delta}$	LMID2
$\underline{a} \cdot x \mid \underline{b} = (\underline{a} \mid \underline{b}) \cdot x$	DRTCM5	$\dot{\delta} \mid x = \dot{\delta}$	CMID1
$\underline{a} \mid \underline{b} \cdot x = (\underline{a} \mid \underline{b}) \cdot x$	DRTCM6	$x \mid \dot{\delta} = \dot{\delta}$	CMID2
$\underline{a} \cdot x \mid \underline{b} \cdot y = (\underline{a} \mid \underline{b}) \cdot (x \parallel y)$	DRTCM7	$\partial_H(\dot{\delta}) = \dot{\delta}$	DID
$(x + y) \mid z = x \mid z + y \mid z$	CM8	$\partial_H(\underline{a}) = \underline{a}$ if $a \notin H$	DRTD1
$x \mid (y + z) = x \mid y + x \mid z$	CM9	$\partial_H(\underline{a}) = \underline{\delta}$ if $a \in H$	DRTD2
$\sigma_{\mathrm{rel}}(x) \between (\sigma_{\mathrm{rel}}(y) + \nu_{\mathrm{rel}}(z)) = \sigma_{\mathrm{rel}}(x \between y)$	DRT5	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$\sigma_{\mathrm{rel}}(x) \mid (\sigma_{\mathrm{rel}}(y) + \nu_{\mathrm{rel}}(z)) = \sigma_{\mathrm{rel}}(x \mid z)$	DRT6	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	D4
$(\sigma_{\mathrm{rel}}(x) + \nu_{\mathrm{rel}}(y)) \mid \sigma_{\mathrm{rel}}(z) = \sigma_{\mathrm{rel}}(x \mid y)$	DRT7	$\partial_H(\sigma_{\mathrm{rel}}(x)) = \sigma_{\mathrm{rel}}(\partial_H(x))$	DRT8

Table 3: Additional axioms for $\mathrm{ACP}_{\mathrm{drt}}$ ($a \in A \cup \{\delta\}$).

1. $a \between \lfloor x \rfloor^\omega = a \cdot \lfloor x \rfloor^\omega$
2. $a \cdot x \between \lfloor y \rfloor^\omega = a \cdot (x \parallel \lfloor y \rfloor^\omega)$
3. $a \cdot x \mid b = (a \mid b) \cdot x$
4. $a \mid b \cdot x = (a \mid b) \cdot x$
5. $a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$.
6. $\partial_H(a) = a$ if $a \notin H$
7. $\partial_H(a) = \delta$ if $a \in H$
8. $a \mid b = c$ if $\gamma(a,b) = c$
9. $a \mid b = \delta$ otherwise

Moreover, the following identities are useful in the calculations to come:

10. $\underline{a} \cdot x \between \sigma_{\mathrm{rel}}(y) = \underline{a} \cdot (x \parallel \sigma_{\mathrm{rel}}(y))$
11. $\underline{a} \cdot x \mid \sigma_{\mathrm{rel}}(y) = \underline{\delta}$
12. $\sigma_{\mathrm{rel}}(x) \between \underline{a} \cdot y = \underline{\delta}$
13. $a \cdot x \between \underline{b} \cdot y = \underline{a} \cdot (x \parallel \underline{by})$
14. $a \cdot x \mid \underline{b} \cdot y = (\underline{a} \mid \underline{b}) \cdot (x \parallel y)$
15. $\sigma_{\mathrm{rel}}(x) \parallel \sigma_{\mathrm{rel}}(y) = \sigma_{\mathrm{rel}}(x \parallel y)$.

Proof 1. $a \between \lfloor x \rfloor^\omega = (\underline{a} + \sigma_{\mathrm{rel}}(a)) \between \lfloor x \rfloor^\omega = \underline{a} \between \lfloor x \rfloor^\omega + \sigma_{\mathrm{rel}}(a) \between \lfloor x \rfloor^\omega = \underline{a} \cdot \lfloor x \rfloor^\omega + \sigma_{\mathrm{rel}}(a) \between (\nu_{\mathrm{rel}}(x) + \sigma_{\mathrm{rel}}(\lfloor x \rfloor^\omega)) = \nu_{\mathrm{rel}}(\underline{a} \cdot \lfloor x \rfloor^\omega) + \sigma_{\mathrm{rel}}(a \between \lfloor x \rfloor^\omega)$ so $a \between \lfloor x \rfloor^\omega = \lfloor \underline{a} \cdot \lfloor x \rfloor^\omega \rfloor^\omega = \lfloor \underline{a} \rfloor^\omega \cdot \lfloor x \rfloor^\omega = a \cdot \lfloor x \rfloor^\omega$. The other identities are equally straightforward, we just display 12: $\sigma_{\mathrm{rel}}(x) \between \underline{a} \cdot y = \sigma_{\mathrm{rel}}(x) \between (\underline{a} \cdot y + \underline{\delta}) = \sigma_{\mathrm{rel}}(x) \between (\nu_{\mathrm{rel}}(a \cdot y) + \sigma_{\mathrm{rel}}(\dot{\delta})) = \sigma_{\mathrm{rel}}(x \between \dot{\delta}) = \sigma_{\mathrm{rel}}(\dot{\delta}) = \underline{\delta}$.

3 Some Simple Calculations

In time free process algebra, there is the following standard specification of a one-item buffer with input port i and output port j:

$$B^{ij} = \sum_{d \in D} ri(d) \cdot sj(d) \cdot B^{ij}$$

A straightforward calculation (see e.g. [3], page 106) shows that the composition of two such buffers in sequence gives a two-item buffer. In the following, we consider three timed versions of this buffer. In each case, only one input per time slice is possible.

3.1 No Delay

We can define a channel that allows one input in every time slice, and outputs with no delay, with input port i and output port j by the following recursive equation:

$$C^{ij} = \sum_{d\in D} ri(d) \cdot \underline{\underline{sj(d)}} \cdot \sigma_{\rm rel}(C^{ij})$$

Using parts 2,3,5 of Proposition 2.2 we see $C^{ij} = \lfloor C^{ij} \rfloor^\omega$. With $H = \{r2(d), s2(d) : d \in D\}$ we can derive:

$$\partial_H(C^{12} \parallel C^{23}) = \partial_H(C^{12} \between C^{23}) + \partial_H(C^{23} \between C^{12}) + \partial_H(C^{12} \mid C^{23}) =$$

$$= \sum_{d\in D} \partial_H\left(r1(d) \cdot \underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \between \left\lfloor C^{23} \right\rfloor^\omega\right) + \sum_{d\in D} \partial_H\left(r2(d) \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\rm rel}(C^{23}) \between \left\lfloor C^{12} \right\rfloor^\omega\right) +$$

$$+ \sum_{d,e\in D} \partial_H\left(r1(d) \cdot \underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \mid r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\rm rel}(C^{23})\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \parallel C^{23}\right) + \sum_{d\in D} \delta \cdot \partial_H\left(\underline{\underline{s3(d)}} \cdot \sigma_{\rm rel}(C^{23}) \parallel C^{12}\right) +$$

$$+ \sum_{d,e\in D} \delta \cdot \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \parallel \underline{\underline{s3(e)}} \cdot \sigma_{\rm rel}(C^{23})\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \between C^{23}\right) + \partial_H\left(C^{23} \between \underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12})\right) +\right.$$

$$\left. + \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \mid C^{23}\right)\right) + \delta =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\underline{\delta} + \sum_{e\in D} \partial_H\left(r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\rm rel}(C^{23})\right) \between \underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) +\right.$$

$$\left. + \sum_{e\in D} \partial_H\left(\underline{\underline{s2(d)}} \cdot \sigma_{\rm rel}(C^{12}) \mid r2(e) \cdot \underline{\underline{s3(e)}} \cdot \sigma_{\rm rel}(C^{23})\right)\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \left(\underline{\delta} + \underline{\delta} + \underline{\underline{c2(d)}} \cdot \partial_H\left(\sigma_{\rm rel}(C^{12}) \parallel \underline{\underline{s3(d)}} \cdot \sigma_{\rm rel}(C^{23})\right)\right) =$$

$$= \sum_{d\in D} r1(d) \cdot \underline{\underline{c2(d)}} \cdot \underline{\underline{s3(d)}} \cdot \partial_H\left(\sigma_{\rm rel}(C^{12}) \parallel \sigma_{\rm rel}(C^{23})\right) =$$

$$= \sum_{d \in D} r1(d) \cdot \underline{\underline{c2(d)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}\Big(\partial_H\left(C^{12} \parallel C^{23}\right)\Big).$$

Using a theory of abstraction as outlined in [2], we can derive that after hiding the $\underline{\underline{c2(d)}}$ actions, the composition behaves again as a no-delay channel, with input port 1 and output port 3. This fact was also stated in [4], where this no-delay channel (called minimal stream delayer) acts as the identity in a data flow algebra.

3.2 Delay 1, Capacity 1

It is interesting to see what happens if we change the previous specification slightly. Consider

$$D^{ij} = \sum_{d \in D} ri(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{sj(d)}} \cdot D^{ij}\right).$$

This specification describes a buffer with capacity one and delay between input and output of one time unit. The composition $\partial_H(D^{12} \parallel D^{23})$ (with H as above) satisfies the following recursive specification:

$$X = \sum_{d \in D} r1(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{c2(d)}}\right) \cdot X_d$$

$$X_d = \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}}\right) \cdot \underline{\underline{c2(e)}} \cdot X_e + \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}} \cdot X + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{c2(e)}}\right) \cdot X_e\right)$$

(for each $d \in D$). The composition denotes a buffer with capacity two and delay of two time units. This is similar to the result obtained if all timing is left out (see [3], page 106).

3.3 Delay 1, More Capacity

Finally, we drop the restriction in the previous specification that output must occur before the next input. We obtain the following specification:

$$E^{ij} = \sum_{d \in D} ri(d) \cdot \sigma_{\mathrm{rel}}\left(\underline{\underline{sj(d)}} \parallel E^{ij}\right).$$

Now, the composition satisfies the following recursive specification:

$$Y^0 = \sum_{d \in D} r1(d) \cdot \sigma_{\mathrm{rel}}(Y^1_d)$$

$$Y_d^1 = \underline{\underline{c2(d)}} \cdot Y_d^2 + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{c2(d)}} \cdot \sigma_{\mathrm{rel}}(Y_{de}^3) \quad \text{for} \quad d \in D$$

$$Y_d^2 = \sigma_{\mathrm{rel}}\left(\underline{\underline{s3(d)}} \cdot Y^0 + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(Y_e^1)\right) + \sum_{e \in D} \underline{\underline{r1(e)}} \cdot \sigma_{\mathrm{rel}}(Y_{de}^3) \quad \text{for} \quad d \in D$$

$$Y_{de}^3 = \underline{\underline{s3(d)}} \cdot Y_e^1 + \underline{\underline{c2(e)}} \cdot \left(\underline{\underline{s3(d)}} \cdot Y_d^2 + \sum_{f \in D} \underline{\underline{r1(f)}} \cdot \underline{\underline{s3(d)}} \cdot \sigma_{\mathrm{rel}}(Y_{ef}^3)\right) +$$

$$+ \sum_{f \in D} \underline{\underline{r1(f)}} \cdot \left(\underline{\underline{c2(e)}} \parallel \underline{\underline{s3(d)}}\right) \cdot \sigma_{\mathrm{rel}}(Y_{ef}^3) \quad \text{for} \quad d, e \in D$$

Again, the delay between input and output is two time units, but now, it is possible that three data elements are present in the system at the same time.

References

[1] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. Technical Report CSR 95-09, Eindhoven University of Technology, Computing Science Department, 1995. To appear in Formal Aspects of Computing.

[2] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra with abstraction. Technical Report CSR 95-17, Eindhoven University of Technology, Computing Science Department, 1995. To appear in Proceedings FCT'95, Dresden.

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

[4] J.A. Bergstra and Gh. Ştefănescu. Network algebra for synchronous and asynchronous dataflow. Technical Report LGPS 122, Utrecht University, Department of Philosophy, 1994.

[5] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85*, number 215 in LNCS, pages 9–23. Springer-Verlag, 1986.

[6] F.E.J. Kruseman Aretz. *Vallen en opstaan (de computer in de wiskunde).* Inaugural Address. Scheltema en Holkema, Amsterdam, 1967.

[7] M.A. Reniers and J.J. Vereijken. Completeness in discrete time process algebra. draft, 1995.