Real space process algebra

Formal Aspects of Computing (1993) 5: 481–529

Formal Aspects of Computing

Real Space Process Algebra

J. C. M. Baeten[1] and J. A. Bergstra[2]

[1] Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands; and [2] Programming Research Group, University of Amsterdam, Amsterdam, and Department of Philosophy, Utrecht University, Utrecht, The Netherlands

Keywords: Process algebra; Real time process algebra; Real space process algebra; Asynchronous communication; Broadcasting; State operator; Priorities

Abstract. The real time process algebra of Baeten and Bergstra [*Formal Aspects of Computing*, **3**, 142–188 (1991)] is extended to real space by requiring the presence of spatial coordinates for each atomic action, in addition to the required temporal attribute. It is found that asynchronous communication cannot easily be avoided. Based on the state operators of Baeten and Bergstra [*Information and Computation*, **78**, 205–245 (1988)] and following Bergstra et al. [*Proc. Seminar on Concurrency*, LNCS 197, Springer, 1985, pp. 76–95], asychronous communication mechanisms are introduced as an additional feature of real space process algebra. The overall emphasis is on the introductory explanation of the features of real space process algebra, and characteristic examples are given for each of these.

1. Introduction

This paper requires familiarity with our paper [BaB91a] that introduces the real time version of the algebra of communicating processes ACP from [BeK84]. The paper collects material from three reports that found their way into various Springer Lecture Notes [BaB91b, BaB91c, BaB92]. We have not included the Lorentz invariant formulation of real space process algebra of [BaB91b] because its relevance for computing is at present a matter of speculation. The work below is entirely driven by the goal to specify various example systems. Of course these are

Correspondence and offprint requests to: J. C. M. Baeten, Department of Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

toy examples, but the authors believe that these examples (in particular Section 5.16) indeed exemplify important computational mechanisms.

Real space process algebra is a modification of real time process algebra. Its purpose is to allow the description of distributed systems with a known spatial distribution. An important class of such systems are communication protocols that connect stations which are engaged in continuous motion through space. We model real space as $\mathbb{R}^3$ and each action is provided with an additional attribute $x \in \mathbb{R}^3$. Thus, the atomic actions of real space process algebra will be of the form $a(x,t)$ with $x \in \mathbb{R}^3$ and $t \in \mathbb{R}_{\geqslant 0}$.

Once we have defined our real space process algebra M as a bisimulation model over certain transition systems, we can consider for each $p \in \mathbb{R}^3$ a subalgebra $M(p)$ consisting of processes that have all their actions located at p only. $M(p)$ can be viewed as a real time process algebra by forgetting the attribute p for each action. Indeed this subalgebra $M(p)$ is isomorphic to the real time process algebra of [BaB91a]. (Only here do we use the slightly more abstract semantics of [Klu91]) via the mapping induced by $a(t) \mapsto a(p,t)$. This provides a satisfactory connection between the developments in this paper and [BaB91a].)

Technically, the point of departure is a finite set A of atomic actions not containing δ. Then $A_\delta = A \cup \{\delta\}$ and $|: A_\delta \times A_\delta \to A_\delta$ is a so-called communication function. $a|b$ is the action that takes place if a and b happen at the same time and place. This rule is captured in the equation

$$a(x,t)|b(x,t) = (a|b)(x,t)$$

What has to be decided when designing a real space process algebra is what happens when simultaneous actions occur at different places. Technically: what is $a(x,t)|b(y,t)$ if $x \neq y$? Our proposal is to have multi-actions $(a(x) \,\&\, b(y))(t)$. These actions entail the simultaneous execution of a set of atomic actions (different from δ) at a set of pairwise different locations. Thus the basic ingredient of real space process algebra is the *atomic multi-action* $(a_1(x_1) \,\&\, \ldots a_n(x_n))(t)$ with $x_i \neq x_j$ for $i \neq j$ and all $a_i \in A$. Taking the atomic multi-actions as atomic actions, real space process algebra becomes an instance of real time process algebra with a structured (and uncountable) collection of atomic actions.

We will now survey the contents of the paper and give a few comments.

1. The algebra of actions: atomic actions, locations, located atomic actions, located atomic multi-actions, timed actions and multi-actions. This is taken from [BaB92].
2. The systems BPA$\sigma\rho$ and ACP$\sigma\rho$, real space versions of BPAρ and ACPρ of [BaB91a], as well as an extension with integration. This is taken from [BaB91b] and [BaB91c].
3. Examples for ACP$\sigma\rho$: we discuss parallel clocks, connections of channels, and other cases where closed expressions are available for the time and place of output, given time and place of input.
4. Static state operators: a static state operator will change its state depending on actions, only irrespective of the precise timing of these actions. Static state operators are introduced and four examples are given. This section is based on [BaB92].
5. Asynchronous communication modelled via state operators: based on the asynchronous communication mechanism of [BKT85] (see also [BKP92]) and the priority operator of [BBK86], asynchronous communication in real space

is developed as an additional feature. Two extensive examples are given. This is based on [BaB91c].

6. Expressing communication channels by means of process creation: in this section it is shown that process creation in the style of [Ber90] provides a reasonable technique to describe a wide variety of communication channels. Seven different implementations of such a communication channel are specified.
7. Next, a real space/time version of the mode transfer mechanism of [Ber89] is defined. Its use is exemplified by the description of some faulty communication channels.
8. The introduction of asynchronous communication in Section 5 requires the use of a priority operator. This in turn leads to additional insights concerning the nature of time stops. In this section we review different alternatives in dealing with time stops.
9. Finally, we consider relative time notation. In this section we adapt the relative time notation of [BaB91a] to the real space setting. We introduce the initial abstraction operator that allows us to have absolute and relative time notation in a single framework. It turns out that finite space process algebra is the case where relative time notation is most useful. An example is provided that illustrates the use of initial abstraction.

Technically, the central issue of this paper is the asynchronous communication mechanism of Section 5. For this reason, we provide some additional introductory remarks on this topic.

Although a rigorous proof is absent, it seems impossible to express communication between processes moving in three-dimensional space with the primitives of [BaB91a] and [BaB91b]. The difficulty arises because both papers use synchronous communication whereas motion of processes seems to call for asynchronous communication: if process P at place/time (x, t) sends d to Q, then Q may receive d at place/time (y, r) provided $|x-y| = v \cdot (r-t)$. Here $|x-y|$ is the distance between x and y and v is the message transmission velocity in the medium connecting P and Q. If P and Q are at rest we find $r = |x-y|/v+t$ and this formula can be used in process expressions in the style of [BaB91a] and [BaB91b].

However, if y depends on time, the equation $|x(t)-y(r)| = v \cdot (r-t)$ has to be solved in order to determine the time r at which message d will be received. In general, no closed form can be found for r. Even worse, the motion of Q (and so y) may depend on actions of the system after t (the time at which d is sent).

In order to study communication between moving processes we assume that messages travel with speed v through space in all directions, i.e. after d is sent at (x, t) it travels through space as an expanding spherical wave. For simplicity we assume that the message can be detected at any distance. It is probably not difficult to describe a decrease in loudness of the signal together with a threshold mechanism for receivers.

According to [BKT85], asynchronous communication in process algebra can be modelled using auxiliary operators. Subsequently, these operators have been studied in detail in [BaB88] where they are called state operators. In [BaB91c], it is indicated how any state operator definition of untimed process algebra can be extended to real time process algebra over finitely many locations, provided a partial ordering on locations is given. In Section 5, we will consider special state operators (as in [BKT85]) for which the action and effect functions depend on

place and time, but that satisfy a commutativity requirement for actions that happen at the same instant of time.

The state operators λ_V^c (state operator λ of asynchronous channel c, e.g. a radio frequency, in state V) are parametrised by finite collections V of triples $\langle d, x, t\rangle$, d a message in D, x a point in space ($x \in \mathbb{R}^3$), t a time (a non-negative real, $t \in \mathbb{R}_{\geqslant 0}$). If $\langle d, x, t\rangle \in V$, this indicates that $\lambda_V^c(X)$ provides an environment for X in which along channel c a datum d was sent at place x and time t. The asynchronous send actions $c \uparrow d(x, t)$ will have the effect that $\langle d, x, t\rangle$ is added to V. In the scope of λ_V^c a process X can perform the action $c \downarrow d(y, r)$ (asynchronous read along channel c of datum d at place y and time r) provided $|y-x| = v \cdot (r-t)$.

Because the message d, after being sent by $c \uparrow d(x, t)$, travels in a spherical wave, it can be received more than once and hence the communication mechanism is of a broadcasting nature. As pointed out in [BaW90], a broadcasting mechanism in process algebra calls for the use of the priority operator of [BBK86]. In the real time and space case this priority operator will express maximal progress with respect to some actions as well.

We can summarise this discussion as follows: P and Q, travelling through space, can communicate by performing actions $c \uparrow d(x, t)$ and $c \downarrow d(y, r)$. This works in a context $\lambda_V^c(P \parallel Q)$. Unsuccessful asynchronous reads are blocked by the action function of the state operator. In order to ensure successful reception of the messages a priority operator $\theta_{c \Downarrow D}$ is needed (here $c \Downarrow D$ contains all effectuated $c \downarrow d$ actions). This operator gives priority to all asynchronous communications at port c. We will allow synchronous communications as well. Unsuccessful synchronous reads and sends at the port sequence σ are blocked by the encapsulation operator $\partial_{H(\sigma)}$ ($H(\sigma)$ contains all synchronous send and receive actions at ports in σ). Thus we are led to process expressions of the form

$$\partial_{H(\sigma)} \circ \theta_{c \Downarrow D} \circ \lambda_V^c(P \parallel Q)$$

For instance, a concurrent alternating bit protocol with moving sender and receiver will take the following form:

$$\partial_{H(\sigma)}(\theta_{1 \Downarrow D} \circ \lambda_V^1(P_1 \parallel Q_1) \parallel \theta_{2 \Downarrow D} \circ \lambda_V^2(P_2 \parallel Q_2))$$

We conclude from our investigations that real space process algebra is an expressive description language that allows very detailed specification of communicating systems that are engaged in (continuous) spatial motion.

In addition, this paper also gives the real time equations for several additional features of ACP: asynchronous communication, the priority operator, state operators with uncountable state space, process creation and both mode transfer operators. Except for the mode transfer operators, these features are all covered in Chapter 6 of [BaW90]. The mode transfer operators were not included there, because their equations are not fully satisfactory in the untimed case. The real time setting provides a clearer picture and the equations given below seem perfectly adequate to us.

By now, there is much work on process algebras that incorporates notions of time (see e.g. [ReR88, MoT90, NiS91]). However, there is not much work that also involves real space or the use of locations. Besides papers already mentioned [BaB91b, BaB91c, BaB92], we only know of [Jef91a, Mur91]. [Jef91a] uses a testing semantics, and [Mur91] a true concurrency semantics. Both papers are much more of a foundational nature, and consider a more general framework. This makes comparison difficult. Our focus is not primarily on theoretical issues, rather we

want to set up a theory that can describe simple examples that occur in connection with communication protocols.

Finally, we remark that we only consider *concrete* process algebra here: there is no concept of a silent or empty step.

2. Real Space Process Algebra

We start by describing (classical) real space process algebra. We follow the presentation of [BaB91c]. The algebra was introduced in [BaB91b], but there we use an untimed deadlock. Here, we use a timed deadlock, as we also did in [BaB91a, BaB91c, BaB92]. We give an operational semantics in the style of Klusener [Klu91].

2.1. Atomic Actions

We start from a set A of (symbolic) atomic actions. The set A is a parameter of the theory, in examples we often have a particular choice for A. Further, we have a special constant δ, denoting inaction, the absence of any execution. In particular, δ cannot terminate. We put $A_\delta = A \cup \{\delta\}$. On A_δ, we have given a *communication function* $|$. This function $|: A_\delta \times A_\delta \to A_\delta$ is also a parameter of the theory, and should be commutative, associative and have δ as a neutral element:

$$a|b = b|a \tag{C1}$$
$$a|(b|c) = (a|b)|c \tag{C2}$$
$$\delta|a = \delta \tag{C3}$$

The set of atomic actions with space and time, AST is

$$AST = \{a(x,t) \mid a \in A, x \in \mathbb{R}^3, t \in \mathbb{R}_{\geqslant 0}\} \cup \{\delta(t) \mid t \in \mathbb{R}_{\geqslant 0}\}$$

Note that inaction is not located, and only has a time parameter, i.e. $\delta(x) = \delta$. It will be useful to consider also the set of atomic actions with just a space parameter, i.e. the set AS defined by

$$AS = \{a(x) \mid a \in A, x \in \mathbb{R}^3\} \cup \{\delta\}$$

We use $a(x)(t)$ as an alternative notation for $a(x,t)$.

2.2 Multi-actions

Multi-actions are process terms generated by actions with a space parameter and the *synchronisation function* &. Multi-actions contain actions that occur synchronously at different locations. MST is the set of multi-actions, generated by AST and &, MS likewise is the set of multi-actions with space coordinates only, generated by AS and &.

For α, β, γ elements of MS, we have the following conditions on the synchronisation function (Table 1). Further, $x \in \mathbb{R}^3$, $a, b \in A$.

Using the axioms of Table 1, each multi-action can be reduced to one of the following three forms:

- δ
- $a(x)$ $(a \in A, x \in \mathbb{R}^3)$
- $a_1(x_1) \,\&\, \ldots \,\&\, a_n(x_n)$ with all points x_i different, all $a_i \in A$, $n > 1$

Table 1. Synchronisation function on MS

$\alpha \,\&\, \beta = \beta \,\&\, \alpha$	(LO1)
$\alpha \,\&\, (\beta \,\&\, \gamma) = (\alpha \,\&\, \beta) \,\&\, \gamma$	(LO2)
$\delta \,\&\, \alpha = \delta$	(LO3)
$a(x) \,\&\, b(x) = \delta$	(LO4)

Table 2. Communication function on MS

$\alpha \mid \beta = \beta \mid \alpha$	(CL1)
$a \mid (\beta \mid \gamma) = (\alpha \mid \beta) \mid \gamma$	(CL2)
$\delta \mid \alpha = \delta$	(CL3)
$\delta(x) = \delta$	(CL4)
$a(x) \mid b(x) = (a \mid b)(x)$	(CL5)
$a(x) \mid (b(x) \,\&\, \beta) = (a \mid b)(x) \,\&\, \beta$	(CL6)
$(a(x) \,\&\, \alpha) \mid (b(x) \,\&\, \beta) = (a \mid b)(x) \,\&\, (\alpha \mid \beta)$	(CL7)
$\mathsf{locs}(\alpha) \cap \mathsf{locs}(\beta) = \varnothing \Rightarrow \alpha \mid \beta = \alpha \,\&\, \beta$	(CL8)
$\mathsf{locs}(\delta) = \varnothing$	(LOCS1)
$\mathsf{locs}(a(x)) = \{x\}$	(LOCS2)
$x \notin \mathsf{locs}(\alpha), \mathsf{locs}(\alpha) \neq \varnothing \Rightarrow \mathsf{locs}(a(x) \,\&\, \alpha) = \mathsf{locs}(\alpha) \cup \{x\}$	(LOCS3)
$\mathsf{ats}(\delta) = \varnothing$	(ATS1)
$\mathsf{ats}(a(x)) = \{a\}$	(ATS2)
$\mathsf{locs}(\alpha \,\&\, \beta) \neq \varnothing \Rightarrow \mathsf{ats}(\alpha \,\&\, \beta) = \mathsf{ats}(\alpha) \cup \mathsf{ats}(\beta)$	(ATS3)

Next, we define the communication function on multi-actions. In Table 2, $a, b \in A$, $\alpha, \beta, \gamma \in \mathsf{MS}$.

In order to state axiom CL8, we need an auxiliary function locs, that determines the set of points (locations) of a multi-action. Complementary to the function locs is the function ats, that determines the set of atomic actions of a multi-action. We will need the function ats later on, in Section 2.6.

2.3. Timed Multi-actions

It is now straightforward to extend the definition of the synchronisation and communication functions to timed multi-actions. In Table 3, $\alpha, \beta \in \mathsf{MS}$.

We have that $a \mid b$ denotes the action that takes place if atomic actions a, b are performed at the same place and time, $\alpha \mid \beta$ denotes the multi-action that takes place if α, β are performed at the same time, $\alpha \,\&\, \beta$ denotes the multi-action that takes place if α, β are performed at the same time provided none of their components are located at the same place.

2.4. Basic Process Algebra

Process algebra (see [BeK84, BaW90]) starts from a given *action alphabet*, here MST. Elements of MST are constants of the sort P of *processes*. The process $a(x,$

Table 3. Synchronisation and communication on MST

$t \neq s \Rightarrow \alpha(t) \,\&\, \beta(s) = \delta(\min(t,s))$	(LO5)
$\alpha(t) \,\&\, \beta(t) = (\alpha \,\&\, \beta)(t)$	(LO6)
$t \neq s \Rightarrow \alpha(t) \mid \beta(s) = \delta(\min(t,s))$	(CL9)
$\alpha(t) \mid \beta(t) = (\alpha \mid \beta)(t)$	(CL10)

t) can let time progress until t, will then execute action a at time t and place x, and then terminate successfully. For $\alpha \in \mathsf{MS}$, the process $\alpha(t)$ can let time progress until t, will then simultaneously execute its set of actions at their respective locations at time t, and terminates successfully. The process $\delta(t)$ can also let time progress until t, but then nothing more is possible (in particular, time cannot progress further). Therefore, the δ actions are often called *time stops*.

Time and Space Basic Process Algebra with Deadlock (BPA$\rho\sigma\delta$) has two binary operators $+, \cdot : \mathsf{P} \times \mathsf{P} \to \mathsf{P}$; $+$ stands for alternative composition and $\cdot$ for sequential composition. Moreover, there is the additional operator $\gg : \mathbb{R}_{\geqslant 0} \times \mathsf{P} \to \mathsf{P}$, the *initialisation operator*. This operator was called the time shift operator in earlier papers. $t \gg X$ denotes the process X starting at time t. This means that all actions that have to be performed at or before time t are turned into deadlocks because their execution has been delayed too long. BPA$\rho\sigma\delta$ has the axioms from Tables 4 and 5 ($\alpha \in \mathsf{MS}$). Axiom ATA4 is explained, since any $\alpha(t)$ action includes the waiting of $\delta(t)$. Notice, however, that $\mathsf{a}(1) + \delta(2)$ is different from $\mathsf{a}(1)$, as the former can reach time 1 and beyond without performing any actions. The letter A in the names of the axioms in Table 5 refers to *absolute time* (versions with relative time were also considered in [BaB91a], and are treated here in Section 9). The naming of the axioms here and in the sequel may not look very systematic to the reader, but is done in this way in order to achieve maximal consistency with earlier papers.

2.5. Algebra of Communicating Processes

An axiomatisation of parallel composition with communication uses the left merge operator $\lfloor\!\lfloor$, the communication merge operator $\mid$, and the encapsulation operator ∂_H of [BeK84]. Moreover, two extra auxiliary operators introduced in [BaB91a] are needed: the ultimate delay operator and the bounded initialisation operator.

The *ultimate delay* operator U takes a process expression X, and returns an

Table 4. BPA_δ

$X+Y = Y+X$	(A1)
$(X+Y)+Z = X+(Y+Z)$	(A2)
$X+X = X$	(A3)
$(X+Y)\cdot Z = X\cdot Z + Y\cdot Z$	(A4)
$(X\cdot Y)\cdot Z = X\cdot(Y\cdot Z)$	(A5)
$X+\delta = X$	(A6)
$\delta\cdot X = \delta$	(A7)

Table 5. Additional axioms of BPA$\rho\sigma\delta$

$\alpha(0) = \delta(0) = \delta$	(ATA1)
$\delta(t) \cdot X = \delta(t)$	(ATA2)
$t < r \Rightarrow \delta(t) + \delta(r) = \delta(r)$	(ATA3)
$\alpha(t) + \delta(t) = \alpha(t)$	(ATA4)
$\alpha(t) \cdot X = \alpha(t) \cdot (t \gg X)$	(ATA5)
$t < r \Rightarrow t \gg \alpha(r) = \alpha(r)$	(ATB1)
$t \geqslant r \Rightarrow t \gg \alpha(r) = \delta(t)$	(ATB2)
$t \gg (X+Y) = (t \gg X) + (t \gg Y)$	(ATB3)
$t \gg (X \cdot Y) = (t \gg X) \cdot Y$	(ATB4)

Table 6. Remaining axioms of ACP$\rho\sigma$

$U(\alpha(t)) = t$	(ATU1)
$U(\delta(t)) = t$	(ATU2)
$U(X+Y) = \max\{U(X), U(Y)\}$	(ATU3)
$U(X \cdot Y) = U(X)$	(ATU4)
$r \geqslant t \Rightarrow \alpha(r) \gg t = \delta(t)$	(ATB5)
$r < t \Rightarrow \alpha(r) \gg t = \alpha(r)$	(ATB6)
$(X+Y) \gg t = (X \gg t) + (Y \gg t)$	(ATB7)
$(X \cdot Y) \gg t = (X \gg t) \cdot Y$	(ATB8)
$X \parallel Y = X \between Y + Y \between X + X \mid Y$	(CM1)
$\alpha(t) \between X = (\alpha(t) \gg U(X)) \cdot X$	(ATCM2)
$(\alpha(t) \cdot X) \between Y = (\alpha(t) \gg U(Y)) \cdot (X \parallel Y)$	(ATCM3)
$(X+Y) \between Z = X \between Z + Y \between Z$	(CM4)
$(\alpha(t) \cdot X) \mid \beta(r) = (\alpha(t) \mid \beta(r)) \cdot X$	(CM5′)
$\alpha(t) \mid (\beta(r) \cdot X) = (\alpha(t) \mid \beta(r)) \cdot X$	(CM6′)
$(\alpha(t) \cdot X) \mid (\beta(r) \cdot Y) = (\alpha(t) \mid \beta(r)) \cdot (X \parallel Y)$	(CM7′)
$(X+Y) \mid Z = X \mid Z + Y \mid Z$	(CM8)
$X \mid (Y+Z) = X \mid Y + X \mid Z$	(CM9)
$\partial_H(a) = a \quad \text{if } a \notin H$	(D1)
$\partial_H(a) = \delta \quad \text{if } a \in H$	(D2)
$\partial_H(\alpha \,\&\, \beta) = \partial_H(\alpha) \,\&\, \partial_H(\beta)$	(ASD)
$\partial_H(\alpha(t)) = (\partial_H(\alpha))(t)$	(ATD)
$\partial_H(X+Y) = \partial_H(X) + \partial_H(Y)$	(D3)
$\partial_H(X \cdot Y) = \partial_H(X) \cdot \partial_H(Y)$	(D4)

element of $\mathbb{R}_{\geqslant 0}$. The intended meaning is that X can idle before $U(X)$, but X can never reach time $U(X)$ or a later time by just idling. The *bounded initialisation* operator is also denoted by $\gg$, and is the counterpart of the operator with the same name that we saw in the axiomatisation of BPA$\rho\sigma\delta$. With $X \gg t$ we denote the process X with its behaviour restricted to the extent that its first action must be performed at a time before $t \in \mathbb{R}_{\geqslant 0}$.

The axioms of ACP$\rho\sigma$ are in Tables 1–6. In Table 6, $H \subseteq A$, $\alpha, \beta \in MS$, $a \in A_\delta$.

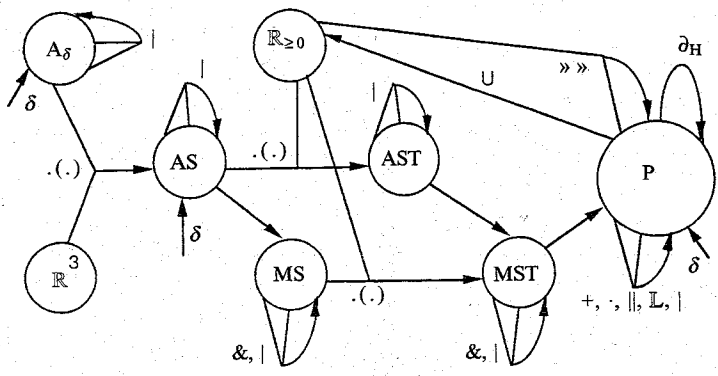

Fig. 1.

For easy reference, we give a picture of the signature of ACP$\rho\sigma$ (Fig. 1). The unlabelled arrows are embeddings. The communication function on A is not shown, as it is only a partial function. In order not to clutter up the picture too much, the auxiliary functions locs, ats are not shown.

2.6. Operational Semantics

We describe an operational semantics for ACP$\rho\sigma$ following Klusener [Klu91] and [BaB91c], a reformularisation of the original semantics in [BaB91a]. All operational semantics we give are in the style of Plotkin [Plo81]. We have a binary relation $\xrightarrow{\mu}$ and a unary relation $\xrightarrow{\mu}\surd$ on closed process expressions for each $\mu \in \mathsf{MST}$. In case $\mu \neq \delta(t)$, the extension of the relations is found as the least fixed point of a simultaneous inductive definition.

The inductive rules for the operational semantics are similar to those used in structured operational semantics. We list the rules for non-δ multi-actions. In Table 7, we have $\alpha,\beta,\gamma \in \mathsf{MS}-\{\delta\}$, $r > 0$ (we never allow timestamp 0 on a transition), $s,t \geqslant 0, x, x', y, y'$ are closed process expressions.

2.7. Delta-Transitions

Now we construct a transition system for a term as follows: first generate all transitions involving non-δ actions using the inductive rules of Table 7. Then, for every node (term) p in this transition system we do the following: first determine its ultimate delay $\mathsf{U}(p) = u$ by means of Table 5 and the additional axioms below. Then, if the ultimate delay is larger than the supremum of the time stamps of all outgoing transitions, we add a transition

$$p \xrightarrow{\delta(u)} \surd$$

Otherwise, we do nothing (add no transitions).

We see that the action rules for parallel composition, and the determination of δ-transitions, make use of the ultimate delay operator. We add the axioms in Table 8 so that the ultimate delay can be syntactically determined for every closed term.

Table 7. Action rules for non-δ actions for ACP$\rho\sigma$

$$r > 0 \Rightarrow \alpha(r) \xrightarrow{\alpha(r)} \surd$$

$$\frac{x \xrightarrow{\alpha(r)} x'}{x+y \xrightarrow{\alpha(r)} x', y+x \xrightarrow{\alpha(r)} x'} \qquad \frac{x \xrightarrow{\alpha(r)} \surd}{x+y \xrightarrow{\alpha(r)} \surd, y+x \xrightarrow{\alpha(r)} \surd}$$

$$\frac{x \xrightarrow{\alpha(r)} x'}{x \cdot y \xrightarrow{\alpha(r)} x' \cdot y} \qquad \frac{x \xrightarrow{\alpha(r)} \surd}{x \cdot y \xrightarrow{\alpha(r)} r \gg y}$$

$$\frac{x \xrightarrow{\alpha(r)} x', r > s}{s \gg x \xrightarrow{\alpha(r)} x'} \qquad \frac{x \xrightarrow{\alpha(r)} \surd, r > s}{s \gg x \xrightarrow{\alpha(r)} \surd}$$

$$\frac{x \xrightarrow{\alpha(r)} x', r < U(y)}{x \parallel y \xrightarrow{\alpha(r)} x' \parallel (r \gg y), x \between y \xrightarrow{\alpha(r)} x' \parallel (r \gg y), y \parallel x \xrightarrow{\alpha(r)} (r \gg y) \parallel x'}$$

$$\frac{x \xrightarrow{\alpha(r)} \surd, r < U(y)}{x \parallel y \xrightarrow{\alpha(r)} r \gg y, x \between y \xrightarrow{\alpha(r)} r \gg y, y \parallel x \xrightarrow{\alpha(r)} r \gg y}$$

$$\frac{x \xrightarrow{\alpha(r)} x', y \xrightarrow{\beta(r)} y', \alpha \mid \beta = \gamma \neq \delta}{x \parallel y \xrightarrow{\gamma(r)} x' \parallel y', x \mid y \xrightarrow{\gamma(r)} x' \parallel y'} \qquad \frac{x \xrightarrow{\alpha(r)} \surd, y \xrightarrow{\beta(r)} \surd, \alpha \mid \beta = \gamma \neq \delta}{x \parallel y \xrightarrow{\gamma(r)} \surd, x \mid y \xrightarrow{\gamma(r)} \surd}$$

$$\frac{x \xrightarrow{\alpha(r)} x', y \xrightarrow{\beta(r)} \surd, \alpha \mid \beta = \gamma \neq \delta}{x \parallel y \xrightarrow{\gamma(r)} x', y \parallel x \xrightarrow{\gamma(r)} x', x \mid y \xrightarrow{\gamma(r)} x', y \mid x \xrightarrow{\gamma(r)} x'}$$

$$\frac{x \xrightarrow{\alpha(r)} x', r < t}{x \gg t \xrightarrow{\alpha(r)} x'} \qquad \frac{x \xrightarrow{\alpha(r)} \surd, r < t}{x \gg t \xrightarrow{\alpha(r)} \surd}$$

$$\frac{x \xrightarrow{\alpha(r)} x', \mathrm{ats}(\alpha) \cap H = \varnothing}{\partial_H(x) \xrightarrow{\alpha(r)} \partial_H(x')} \qquad \frac{x \xrightarrow{\alpha(r)} \surd, \mathrm{ats}(\alpha) \cap H = \varnothing}{\partial_H(x) \xrightarrow{\alpha(r)} \surd}$$

2.8. Bisimulations

A *bisimulation* on is a binary relation R on closed process expressions such that ($\mu \in \mathrm{MST}$):

1. For each p and q with R(p, q): if there is a step μ possible from p to p′, then there is a closed process expression q′ such that R(p′, q′) and there is a step μ possible from q to q′.
2. For each p and q with R(p, q): if there is a step μ possible from q to q′, then there is a closed process expression p′ such that R(p′, q′) and there is a step μ possible from p to p′.
3. For each p and q with R(p, q): a termination step μ to $\surd$ is possible from p if it is possible from q.

Table 8. Ultimate delay axioms

$U(t \gg X) = \max\{t, U(X)\}$	(ATU5)
$U(X \parallel Y) = \min\{U(X), U(Y)\}$	(ATU6)
$U(X \between Y) = \min\{U(X), U(Y)\}$	(ATU7)
$U(X \mid Y) = \min\{U(X), U(Y)\}$	(ATU8)
$U(X \gg t) = \min\{t, U(X)\}$	(ATU9)
$U(\partial_H(X)) = U(X)$	(ATU10)

We say expressions p and q are *bisimilar*, denoted $p \leftrightarrow q$, if there exists a bisimulation on closed process expressions with $R(p, q)$. In [Klu91] it is shown that bisimulation is a congruence relation on closed process expressions, and that closed process expressions modulo bisimulation determine a model for ACP$\rho\sigma$. Indeed, this model is isomorphic to the initial algebra. The advantage of this operational semantics is that it allows extensions to models containing recursively defined processes.

It is also possible to give an explicit graph model for ACP$\rho\sigma$ as in [BaB92].

2.9. Graph Model

It is possible to construct a graph model for ACP$\rho\sigma$. However, we obtain a number of simplifications if we only consider the domain of process *trees*. Therefore, we will limit our domain to trees.

Process trees are directed rooted trees with edges labelled by timed multi-actions, satisfying the condition that for each pair of consecutive transitions $s_1 \xrightarrow{\alpha(r)} s_2 \xrightarrow{\beta(t)} s_3$ it is required that $r < t$ (however, in case $\beta \equiv \delta$, we also allow $r = t$). Moreover, we require that the endnode of a δ-transition has no outgoing edges, and that the timestamp of a δ-transition is larger than the supremum of the timestamps of its brother edges (edges starting from the same node).

Now $+$, $\cdot$, $\parallel$, $\between$, $|$, ∂_H, $\gg$, U and $\gg$ can be defined on these trees in a straightforward manner:

- For $+$, take the disjoint union of the trees and identify the roots. If one of the roots has an outgoing δ-edge, remove this edge if its timestamp is less than or equal to the supremum of the timestamps of the outgoing edges of the other root (but keep one of the two δ-edges, if both roots have a δ-edge with the same timestamp).
- $t \gg g$ is obtained by removing every edge from the root with label $a(r)$ with $r \leqslant t$. If this removes all edges starting from the root, add a $\delta(t)$-edge to an endpoint.
- $g \cdot h$ is constructed as follows: identify each non-δ endpoint s of g (endpoint with no incoming δ-edge) with the root of a copy of $t \gg h$, where t is the time of the edge leading to s.
- $U(g) = \sup\{r \in \mathbb{R}_{\geqslant 0} \mid root(g) \xrightarrow{a(r)} s \text{ for } a \in A_\delta \text{ and some } s \in g\}$. If g is the trivial one-node graph, put $U(g) = \infty$.
- Let for $s \in g$, $(g)_s$ denote the subgraph of g with root s. Then $g \parallel h$ is defined as follows.

  The set of states is the Cartesian product of the state sets of g and h, the root the pair of roots.

Transitions: if $s \xrightarrow{\alpha(r)} s', \alpha \neq \delta$, r is larger than the timestamp of the incoming edge of $\langle s, v\rangle$ (if any) and $r < U((h)_v)$ then $\langle s, v\rangle \xrightarrow{\alpha(r)} \langle s', v\rangle$.
if $v \xrightarrow{\alpha(r)} v', \alpha \neq \delta$, r is larger than the timestamp of the incoming edge of $\langle s, v\rangle$ (if any) and $r < U((g)_s)$ then $\langle s, v\rangle \xrightarrow{\alpha(r)} \langle s, v'\rangle$.
if $s \xrightarrow{\alpha(r)} s'$ and $v \xrightarrow{\beta(r)} v'$, r is larger than the timestamp of the incoming edge of $\langle s, v\rangle$ (if any) and $\alpha|\beta = \gamma \neq \delta$ then $\langle s, v\rangle \xrightarrow{\gamma(r)} \langle s', v'\rangle$.

Lastly, let $\langle s, v\rangle$ be a node with s and v not both endnodes in g resp. h. Let t be the timestamp of the incoming edge ($t = 0$ if there is no incoming edge), and put $u = \min\{U((g)_s), U((h)_v)\}$. If $u \leqslant t$, add a transition $\langle s, v\rangle \xrightarrow{\delta(r)}$ to an endpoint. If $u > t$ and u is larger than the supremum of the timestamps of the outgoing edges, add a transition $\langle s, v\rangle \xrightarrow{\delta(u)}$ to an endpoint. Otherwise, do nothing.

It is an exercise to show that this construction always yields a tree again.

- The construction of $g \between h$ and $g | h$ is similar to the construction of $g \parallel h$. States and root are the same, the transitions of $g \between h$, $g | h$ are the same as for $g \parallel h$, only if $\langle s, v\rangle$ is the root, any transition of $g \between h$ must be given by the clause:

  $$\text{if } s \xrightarrow{\alpha(r)} s', \alpha \neq \delta \text{ and } r < U((h)_v) \text{ then } \langle s, v\rangle \xrightarrow{\alpha(r)} \langle s', v\rangle$$

  and any transition of $g | h$ given by the clause:

  $$\text{if } s \xrightarrow{\alpha(r)} s' \text{ and } v \xrightarrow{\beta(r)} v' \text{ and } \alpha|\beta = \gamma \neq \delta \text{ then } \langle s, v\rangle \xrightarrow{\gamma(r)} \langle s', v'\rangle$$

  In both cases, we add δ-transitions under the same conditions as in $g \parallel h$.
- Finally, $\partial_H(g)$ is obtained from g by removing all edges with a label $\alpha(t)$ with $\mathrm{ats}(\alpha) \cap H \neq \varnothing$. If this removes all outgoing edges of a node s, add an edge $s \xrightarrow{\delta(u)}$ to an endpoint, where $u = U((g)_s)$.

Bisimulation on these graphs is defined as, e.g., in [BeK84]. One may prove in a standard fashion that bisimulation is a congruence for all operators of ACP$\rho\sigma$. Thus, we can define the operators on the set of bisimulation equivalence classes. We obtain a model of ACP$\rho\sigma$ that we will call M_A.

2.10. Integration

An extension of ACP$\rho\sigma$ (called ACP$\rho\sigma$I) that is very useful in applications is the extension with the integral operator, denoting a choice over a continuum of alternatives. That is, if V is a subset of $\mathbb{R}_{\geqslant 0}$, and v is a variable over $\mathbb{R}_{\geqslant 0}$, then $\int_{v \in V} P$ denotes the alternative composition of alternatives $P[t/v]$ for $t \in V$ (expression P with non-negative real t substituted for variable v). In this expression, a free occurrence of v in P becomes bound. Alpha conversion must be allowed to avoid name clashes. For more information, we refer the reader to [BaB91a] and [Klu91]. The operational semantics is straightforward (Table 9, $\mu \in \mathrm{MST}$). Delta-transitions are generated exactly as before.

Of course, we cannot give a complete axiomatisation of general integration because the general theory is undecidable, but the axioms INT1-7 (Table 10) are very useful in derivations (for a complete axiomatisation of a subtheory, see

Table 9. Action rules for integration

$\dfrac{x(t) \xrightarrow{\mu} x',\ t \in V}{\int_{v \in V} x(v) \xrightarrow{\mu} x'}$	$\dfrac{x(t) \xrightarrow{\mu} \surd,\ t \in V}{\int_{v \in V} x(v) \xrightarrow{\mu} \surd}$

Table 10. Axioms for integration

$\int_{v \in \{t\}} P = P[t/v]$	(INT1)
$v \notin FV(P)\ \&\ V \neq \varnothing \Rightarrow \int_{v \in V} P = P$	(INT2)
$\int_{v \in \varnothing} P = \delta$	(INT3)
$\int_{v \in V \cup W} = \int_{v \in V} P + \int_{v \in W} P$	(INT4)
$\int_{v \in V} (P + Q) = \int_{v \in V} P + \int_{v \in V} Q$	(INT5)
$v \notin FV(Q) \Rightarrow \int_{v \in V} (P \cdot Q) = \left(\int_{v \in V} P\right) \cdot Q$	(INT6)
for all $t \in V (P[t/v] = Q[t/v]) \Rightarrow \int_{v \in V} P = \int_{v \in V} Q$	(INT7)
$U\left(\int_{v \in V} P\right) = \sup\{U(P[t/v]) : t \in V\}$	(ATU11)

[Klu91]). We also give an extra axiom for the ultimate delay operator (ATU11), in order to derive δ-transitions also for expressions with integrals.

In the graph model, the graph of $\int_{v \in V} P$ is constructed by first identifying the roots of the graphs $P[t/v]$ for $t \in V$. Next, remove all δ-edges that do not satisfy the condition above (i.e. its timestamp is not larger than the supremum of the timestamps of its brother edges). Add again a δ-edge with timestamp t, if the ultimate delay t of the first graph is larger than the supremum of the timestamps of the remaining edges. Some examples:

$$\int_{v \in (0,1)} \delta(v) = \delta(1); \quad \int_{v \in (0,1)} a(v) = \int_{v \in (0,1)} a(v) + \delta(1);$$

$$\delta(1) \parallel \int_{v \in (0,1)} a(v) = \int_{v \in (0,1)} a(v) \cdot \delta(1)$$

2.11. Remark

A consequence of the addition of the integral construct is that the model M_A becomes very large. Inspection of the term $\int_{t \in \langle 0,1 \rangle} a(t)$ shows that the cardinality

of a transition system can become $2^{\aleph_0}$. Also, we will have at least $2^{2^{\aleph_0}}$ different processes in M_A. This is shown by the following example: let, for $V \subseteq \mathbb{R}_{>0}$, the process $p(V)$ be defined by

$$p(V) = \int_{t \in V} a(t)$$

It now follows that $p(V) \leftrightarrows p(W) \Leftrightarrow V = W$, so all processes $p(V)$ are in a different bisimulation class, and since there are $2^{2^{\aleph_0}}$ subsets of the positive reals, there are at least that many elements in M_A.

3. Examples

3.1. Clocks

This example is adapted from [BaB91a]. We describe three clocks placed at position x.

1. $C_1(x,t) = tick(x,t) \cdot C_1(x,t+1)$
   Process $C_1(x,1)$ will start ticking at time 1 and continue to do so each time unit with absolute precision. This is a stable exact clock.
2. The second clock allows some fluctuations of the ticks.

$$C_2(x,t) = \int_{v \in [t-0.01, t+0.01]} tick(x,v) \cdot C_2(x,t+1)$$

3. The third clock cumulates the errors:

$$C_3(x,t) = \int_{v \in [t-0.01, t+0.01]} tick(x,v) \cdot C_3(x,v+1)$$

Now let $x_0, \ldots, x_{n-1}$ be pairwise different locations, and assume $a \mid b = \delta$ for all $a, b \in A$. Define

$$C_i^n(t) = C_i(x_0,t) \parallel \ldots \parallel C_i(x_{n-1},t)$$

then $C_1^n(1), C_2^n(1), C_3^n(1)$ are three systems each having n clocks running in parallel.

The real space aspect mainly consists of the fact that nothing can prevent clocks at different locations from ticking at the same time. Thus multi-actions like $tick(x_0) \,\&\, \ldots \,\&\, tick(x_{n-1})$ are needed. Finally, $C_1(x_0,1) \parallel C_2(x_1,1) \parallel C_3(x_2,1)$ is a parallel composition of the three types of clocks.

3.2. Data Communication

In this example, we use the communication function that is standard in the setting of ACP. This means that the alphabet A contains actions $r_i(d)$, $s_i(d)$, $c_i(d)$ for $i \in I$ and $d \in D$, where I is a finite set of port names and D a finite set of data elements. The communication function has $r_i(d) \mid s_i(d) = s_i(d) \mid r_i(d) = c_i(d)$, and yields δ in all other cases. The intuitive meaning of $r_i(d)$ is "receive datum d along port i", and $s_i(d)$, $c_i(d)$ are send, communicate respectively.

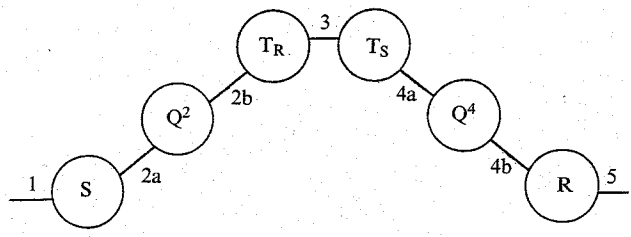

Fig. 2.

We describe data communication from x_1 to x_3 via a static intermediate station x_2. x_1 is the location of the sender S, x_3 the location of the receiver R and x_2 the location of the transmitter T. We split the transmitter into a receiving part T_R at $x_2-\varepsilon$ and a sending part T_S at $x_2+\varepsilon$. The media Q pass on messages (see Fig. 2).

The example is made more complicated than needed in order to allow an interesting modular decomposition. We proceed to give the specifications of the components:

$$S = \int_{t>0} \sum_{d\in D} r_1(d)\,(x_1,t)\cdot s_{2a}(d)\,(x_1,t+w_1)\cdot((t+2w_1) \gg S)$$

$$T_R = \int_{t>0} \sum_{d\in D} r_{2b}(d)\,(x_2-\varepsilon,t)\cdot s_3(d)\,(x_2,t+w_2)\cdot((t+2w_2) \gg T_R)$$

$$T_S = \int_{t>0} \sum_{d\in D} r_3(d)\,(x_2,t)\cdot s_{4a}(d)\,(x_2+\varepsilon,t+w_3)\cdot((t+2w_3) \gg T_S)$$

$$R = \int_{t>0} \sum_{d\in D} r_{4b}(d)\,(x_3,t)\cdot s_5(d)\,(x_3,t+w_4)\cdot((t+2w_4) \gg R)$$

$$Q^2 = \int_{t>0} \sum_{d\in D} r_{2a}(d)\,(x_1,t)\cdot s_{2b}(d)\left(x_2-\varepsilon,t+\frac{|x_2-\varepsilon-x_1|}{v}\right)\cdot((t+c_2) \gg Q^2)$$

$$Q^4 = \int_{t>0} \sum_{d\in D} r_{4a}(d)\,(x_2+\varepsilon,t)\cdot s_{4b}(d)\left(x_3,t+\frac{|x_3-(x_2+\varepsilon)|}{v}\right)\cdot((t+c_4) \gg Q^4)$$

Here w_1, w_2, w_3, w_4, c_2, c_4 are delay constants, and v is the transmission speed of the data elements. The (static) composition SYS_{st} of these components can now be found using ACP$\sigma\rho$. In Section 5, we will allow x_3 to be mobile and a "dynamic" composition can be given:

$$SYS_{st} = \partial_{H(2a,ab,4a,4b)}(S \| Q^2 \| \partial_{H(3)}(T_R \| T_S) \| Q^4 \| R)$$

Here, if σ is a list over I, then $H(\sigma) = \{r_i(d), s_i(d) : d \in D, i \text{ in } \sigma\}$.

In the bisimulation model, it can be found that SYS_{st} can also be modularised as follows:

$$SYS_{st} = \partial_{H(3)}(\partial_{H(2a,2b)}(S \| Q2 \| TR) \| \partial_{H(4a,4b)}(TS \| Q4 \| R))$$

An equational proof of this identity can be found by means of a straightforward real time/space adaptation of conditional alphabet axioms and techniques (see e.g. [BaW90]).

For this protocol to work properly (i.e. to be deadlock-free and to transmit all data) it is necessary and sufficient that $2w_1 \geqslant c_2 \geqslant 2w_2 \geqslant 2w_3 \geqslant c_4 \geqslant 2w_4$. Of course it will generate a deadlock if it is placed in an environment that offers inputs at a too high frequency; in particular $1/2w_1$ is the maximum admitted frequency.

4. State Operator

This section is based on [BaB92]. The state operator was introduced in [BaB88]. It keeps track of the global state of a system, and is used to describe actions that have a side effect on a state space. The state operator has showed itself useful in a range of applications, e.g. in the translation of programming or specification languages into process algebra (see e.g. [Vaa90b]).

The state operator comes equipped with two functions: given a certain state and an action to be executed from that state, the function action gives the resulting action and the function effect the resulting state. For instance, if $P = a \cdot P$ (in symbolic process algebra, without timing), and the state space is $\{0, 1, \ldots, 23\}$, $\mathsf{action}(a, s) = \mathsf{tick}$, $\mathsf{effect}(a, s) = s+1 \bmod 24$, then $\lambda_0(P)$ describes a clock showing hours, starting at 0 hours, for

$$\lambda_0(P) \xrightarrow{\mathsf{tick}} \lambda_1(P) \xrightarrow{\mathsf{tick}} \lambda_2(P) \xrightarrow{\mathsf{tick}} \ldots.$$

More simple examples can be found in [BaW90]. We give simple examples with time and space (including clocks) in Section 4.4.

Adding time and space, the actions we start from are the single actions in AST (not multi-actions), so the action and effect functions are defined on actions in time and space.

Things become more difficult if we consider multi-actions: in order to calculate the resulting global state, we need to apply the effect function to the component actions in a certain order. How is this order determined? We will assume that we have a partial ordering on located actions, and if we apply the effect function, we do so in the order determined by this ordering. We make this precise in the following definition.

4.1. Definition

Again, A is the set of atomic actions, AS the set of located actions and AST the set of located and timed actions. Let S be a set of states. Assume we have functions

$$\mathsf{action}: \mathsf{AST} \times \mathsf{S} \to \mathsf{AST} \quad \mathsf{effect}: \mathsf{AST} \times \mathsf{S} \to \mathsf{S}$$

such that $\mathsf{action}(\delta(t), s) = \delta$ and $\mathsf{effect}(\delta(t), s) = s$ for all $s \in \mathsf{S}$, $t \in \mathbb{R}_{\geqslant 0}$ (we say: δ is inert). Let $<$ be a partial ordering on located actions AS. We say that $a(x)$ is *subsequent to* $b(y)$ if $b(y) < a(x)$. This informally means that evaluation of a multi-action with respect to the state operator is performed in order of subsequence.

If $\alpha \in \mathsf{MS}$, we write $\mathsf{TO}(\alpha)$ if the set of located actions of α is totally ordered by $<$, and we write $a(x) < \alpha$ if $a(x) < b(y)$ for all constituents $b(y)$ of α. Then we define the extension of the functions action and effect to multi-actions as shown in Table 11 ($\alpha \in \mathsf{MS}$).

Table 11. Action and effect on multi-actions

$$\neg TO(\alpha) \Rightarrow action(\alpha(t), s) = \delta(t)$$
$$\neg TO(\alpha) \Rightarrow effect(\alpha(t), s) = s$$
$$TO(\alpha) \wedge a(x) < \alpha \Rightarrow action((a(x) \,\&\, \alpha)(t), s) = action(a(x,t), effect(\alpha(t), s)) \,\&\, action(\alpha(t), s)$$
$$TO(\alpha) \wedge a(x) < \alpha \Rightarrow effect((a(x) \,\&\, \alpha)(t), s) = effect(a(x,t), effect(\alpha(t), s))$$

Table 12. State operator

$$\lambda_s(\mu) = action(\mu, s) \qquad (SO1)$$
$$\lambda_s(\mu \cdot X) = action(\mu, s) \cdot \lambda_{effect(\mu,s)}(X) \qquad (SO2)$$
$$\lambda_s(X+Y) = \lambda_s(X) + \lambda_s(Y) \qquad (SO3)$$
$$\lambda_s\left(\int_{v \in V} P\right) = \int_{v \in V} \lambda_s(P) \qquad (SO4)$$

Table 13. Action relations for the state operator

$$\frac{x \xrightarrow{\mu} x',\ action(\mu, s) = \nu \neq \delta(t)}{\lambda_s(x) \xrightarrow{\nu} \lambda_{effect(\mu,s)}(x')} \qquad \frac{x \xrightarrow{\mu} \surd,\ action(\mu, s) = \nu \neq \delta(t)}{\lambda_s(x) \xrightarrow{\nu} \surd}$$

4.2. Definition

The defining equations for the state operator (Table 12) are now straightforward (cf. [BaB88]). If S is a set of states, then for each $s \in S$ we have an operator $\lambda_s : P \to P$. In Table 12, $s \in S$, $\mu \in MST$, $t \geqslant 0$, X, Y processes, P an expression possibly containing a free time variable v.

It is equally straightforward to give action rules for the operational semantics ($\mu \in MST, \mu \neq \delta(t)$) (Table 13).

In order to deal with δ-transitions, we add the axiom

$$U(\lambda_s(x)) = U(x) \qquad (ATU12)$$

Then, we determine the existence of δ-transitions as before (Section 2.7).

4.3. Ordering Located Actions

As we saw, the definition of the state operator uses an ordering on located actions. If we have a partial ordering $<_L$ on $\mathbb{R}^3$, and a partial ordering $<_A$ on atomic actions, we can define an ordering $<$ on AS as follows:

$$a(x) < b(y) \Leftrightarrow a <_A b \text{ or } (x <_L y \text{ and either } a = b \text{ or } a, b \text{ incomparable}).$$

In all our examples, we will be dealing with an ordering that can be generated in this way. There are two extremes of this, viz. $<_A = \varnothing$, $<_L = \varnothing$. In the examples in this section, we will always have $<_A = \varnothing$, and in the following section, on asynchronous communication, we have $<_L = \varnothing$.

One may feel that the ordering of locations introduces an overspecification. Indeed an alternative method is to evaluate a multi-action as a sum over all possible orderings on its located actions. We have chosen for a fixed ordering in order to simplify the situation.

4.4. Examples

Example 1. We specify a clock. The state space keeps track of time. Suppose we have a fixed point in space x. Define, for each $s \in \mathbb{N}$, $t \in \mathbb{R}_{\geqslant 0}$,

$$\mathsf{effect}(\mathsf{tick}(x,t),s) = s+1$$
$$\mathsf{action}(\mathsf{tick}(x,t),s) = \mathsf{tick}(x,t)$$

Then a clock is given by the process $\lambda_0(P(0))$, where $P(t) = \mathsf{tick}(x,t+1) \cdot P(t+1)$.

Example 2. We use the clock of the previous example in order to list when action a occurs at point y (measured in discrete time). The state space is $\mathbb{N} \times \mathbb{L}$, where $\mathbb{L}$ denotes the set of lists over $\mathbb{N}$. Look at the process $\lambda_{0,\varnothing}(Q \parallel P(0))$, where $P(t)$ is as in Section 4.3 and Q is given by $Q = \int_{t \geqslant 0} a(y,t) \cdot Q$.

We have the following definitions:

$$\mathsf{effect}(\mathsf{tick}(x,t),\langle n,\sigma\rangle) = \langle n+1,\sigma\rangle$$
$$\mathsf{action}(\mathsf{tick}(x,t),\langle n,\sigma\rangle) = \mathsf{tick}(x,t)$$
$$\mathsf{effect}(a(y,t),\langle n,\sigma\rangle) = \langle n,\sigma \frown \langle n\rangle\rangle$$
$$\mathsf{action}(a(y,t),\langle n,\sigma\rangle) = a(y,t).$$

Moreover, assume $y <_L x$. As a consequence, we have e.g.

$$\mathsf{effect}((a(y) \,\&\, \mathsf{tick}(x))(t),\langle n,\sigma\rangle) = \langle n+1,\sigma \frown \langle n+1\rangle\rangle$$

Example 3. Figure 3 shows a diagram for a serial switch.
The switches A and B are given by the equations

$$A = \int_{t \geqslant 0} \mathsf{switch}(\mathsf{left},t) \cdot A$$
$$B = \int_{t \geqslant 0} \mathsf{switch}(\mathsf{right},t) \cdot B$$

(here, the switches are located at points left, right). We have state space $S = \{\mathsf{on}, \mathsf{off}\}$; e.g. in Fig. 3, the lamp is in state off. We define the action and effect functions as follows.

$$\mathsf{action}(\mathsf{switch}(x,t), \mathsf{off}) = \mathsf{turnon}(x,t)$$
$$\mathsf{action}(\mathsf{switch}(x,t), \mathsf{on}) = \mathsf{turnoff}(x,t)$$
$$\mathsf{effect}(\mathsf{switch}(x,t), \mathsf{on}) = \mathsf{off}$$
$$\mathsf{effect}(\mathsf{switch}(x,t), \mathsf{off}) = \mathsf{on}$$

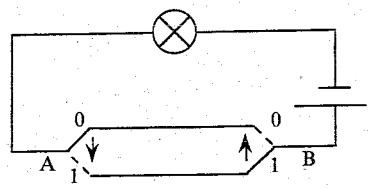

Fig. 3.

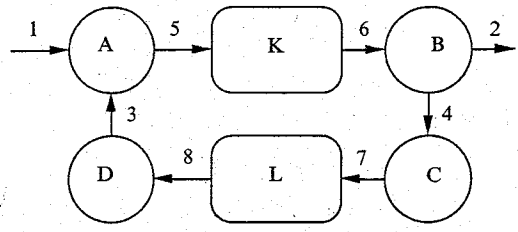

Fig. 4.

Starting in state off (as in Fig. 3) we have the process

$$P = \lambda_{off}(A \parallel B)$$

(We assume no communication occurs.)

An interesting situation occurs if both switches are turned at the same time. Suppose we have the ordering right $<_L$ left. Let t_0 be a fixed time. Then we have

$$\lambda_{off}(\mathsf{switch}(\mathsf{left}, t_0) \cdot A \parallel \mathsf{switch}(\mathsf{right}, t_0) \cdot B) = (\mathit{turnon}(\mathit{left}) \,\&\, \mathit{turnoff}(\mathit{right}))(t_0) \cdot \lambda_{off}(A \parallel B)$$

$$\lambda_{on}(\mathsf{switch}(\mathsf{left}, t_0) \cdot A \parallel \mathsf{switch}(\mathsf{right}, t_0) \cdot B) = (\mathit{turnoff}(\mathit{left}) \,\&\, \mathit{turnon}(\mathit{right}))(t_0) \cdot \lambda_{on}(A \parallel B)$$

Example 4. As a larger example, we present a version of the Concurrent Alternating Bit Protocol. We base our description on the specification in [GlV89]. Other descriptions can be found in [KoM90, LaM87].

In Fig. 4, elements of a finite data set D are sent from port 1 to port 2. From A to B, frames consisting of a data element and an alternating bit are sent, from C to D, independently, acknowledgements. We assume port i is located at point x_i.

K and L are unbounded faulty queues, that can lose data. It was shown in [GlV89] that we can assume that loss of data only occurs at the top of the queue. In the example of Section 3.2, we introduced separate processes in order to deal with data transmission. Here, we model the queues by means of a state operator. For the queue K, we allow elements to enter the queue at x_5, and to leave the queue at x_6, and similarly for L. We present the specification first, and then define the state operator.

In the specification, there are two parameters: w_0 is the amount of time that a sender allows to pass before a message is sent into the queue again, and w_1 is the amount of time after which a receiver checks the queue again after an unsuccessful attempt. Put another way, $1/w_0$ is the retransmission frequency, and $1/w_1$ is the polling frequency. These parameters can be filled in arbitrarily, and do not affect the correctness of the protocol. Moreover, in the specification, we use a system delay constant 1.

The specification of the senders and receivers now looks as follows:

$$A = A(0)$$

$$A(b) = \int_{t \geqslant 0} \sum_{d \in D} r_1(d)(x_1, t) \cdot \mathsf{enq}(db)(x_5, t+1) \cdot A(d, b, t+2) \quad \text{for each } b \in \{0, 1\}$$

$$A(d, b, t) = \mathsf{enq}(db)(x_5, t+w_0) \cdot A(d, b, t+w_0) + \int_{v \in (t, t+w_0]} r_3(\mathsf{next})(x_3, v) \cdot A(1-b) \quad \text{for each } b \in \{0, 1\}, d \in D, t \geqslant 0$$

$$B = B(0,0)$$
$$B(b,t) = (\mathrm{deq}(\perp)(x_6, t+w_1) + \sum_{d\in D} \mathrm{deq}(d(1-b))(x_6, t+w_1)) \cdot B(b, t+w_1) + \sum_{d\in D} \mathrm{deq}(db)(x_6, t+w_1) \cdot B(d, b, t+w_1)$$
for each $b \in \{0,1\}, t \geqslant 0$.
$$B(d,b,t) = s_2(d)(x_2, t+1) \cdot s_4(\mathrm{next})(x_4, t+2) \cdot B(1-b, t+2)$$
for each $b \in \{0,1\}, d \in D, t \geqslant 0$
$$C = C(1,0)$$
$$C(b,t) = \mathrm{enq}(b)(x_7, t+w_0) \cdot C(b, t+w_0) + \int_{v\in(t,t+w_0]} r_4(\mathrm{next})(x_4, v) \cdot \mathrm{enq}(b)(x_7, v+1) \cdot C(1-b, v+2)$$
for each $b \in \{0,1\}, t \geqslant 0$
$$D = D(0,0)$$
$$D(b,t) = (\mathrm{deq}(\perp)(x_8, t+w_1) + \mathrm{deq}(1-b)(x_8, t+w_1)) \cdot D(b, t+w_1) + \mathrm{deq}(b)(x_8, t+w_1) \cdot s_3(\mathrm{next})(x_3, t+w_1+1) \cdot D(1-b, t+w_1+1)$$
for each $b \in \{0,1\}, t \geqslant 0$
$$E = \int_{t>0} \mathrm{error}(x_6, t) \cdot E \quad F = \int_{t>0} \mathrm{error}(x_8, t) \cdot F$$

Here, error(x, t) is an action that communicates with no other actions. It represents a failure by a channel with endpoint at position x. The channel itself is represented by a state operator, and an error action in the scope of such a state operator effects the loss of that element in the channel, that has been waiting in it for the longest period.

Now the Concurrent Alternating Bit Protocol is defined by

$$\mathrm{CABP} = \partial_H(\lambda_\varnothing(A \| E \| B) \| \lambda_\varnothing(C \| F \| D)).$$

Here, the encapsulation set is

$$H = \{r_k(\mathrm{next}), s_k(\mathrm{next}) : k = 3, 4\}$$

In the process CABP, we are actually dealing with two state operators: the state operator for processes A, E, B has a state space consisting of lists of frames, the state operator for C, F, D has a state space consisting of lists of booleans. We have the following ordering on places: $x_6 > x_5$ and $x_8 > x_7$. This makes it impossible to add an element to an empty queue and read it out at the same instant of time: an element needs a positive amount of time to propagate through the queue.

We define the action and effect function for both state operators at the same time. Since the action and effect functions are independent of space and time, we will not list these coordinates, in order to save space. We have $f \in D \times B$, $b \in B$, and $\sigma \in (D \times B)^*, \beta \in B^*$:

$\mathrm{action}(\mathrm{enq}(f), \sigma) = c_5(f)$
$\mathrm{effect}(\mathrm{enq}(f), \sigma) = \sigma^* f$
$\mathrm{action}(\mathrm{deq}(\perp), \varnothing) = i$
$\sigma \neq \varnothing \Rightarrow \mathrm{action}(\mathrm{deq}(\perp), \sigma) = \delta$
$\sigma = \varnothing$ or $\mathrm{top}(\sigma) \neq f \Rightarrow \mathrm{action}(\mathrm{deq}(f), \sigma) = \delta$

$\mathrm{action}(\mathrm{enq}(b), \beta) = c_7(b)$
$\mathrm{effect}(\mathrm{enq}(b), \beta) = \beta^* b$
$\mathrm{action}(\mathrm{deq}(\perp), \varnothing) = j$
$\beta \neq \varnothing \Rightarrow \mathrm{action}(\mathrm{deq}(\perp), \beta) = \delta$
$\beta = \varnothing$ or $\mathrm{top}(\beta) \neq b \Rightarrow \mathrm{action}(\mathrm{deq}(b), \beta) = \delta$

$$\mathsf{top}(\sigma) = \mathsf{f} \Rightarrow \mathsf{action}(\mathsf{deq}(\mathsf{f}), \sigma) = \mathsf{c}_6(\mathsf{f}) \qquad \mathsf{top}(\beta) = \mathsf{b} \Rightarrow \mathsf{action}(\mathsf{deq}(\mathsf{b}), \beta) = \mathsf{c}_8(\mathsf{f})$$

$$\sigma = \varnothing \text{ or } \mathsf{top}(\sigma) \neq \mathsf{f} \Rightarrow \mathsf{effect}(\mathsf{deq}(\mathsf{f}), \sigma) = \sigma \qquad \beta = \varnothing \text{ or } \mathsf{top}(\beta) \neq \mathsf{b} \Rightarrow \mathsf{effect}(\mathsf{deq}(\mathsf{b}), \beta) = \beta$$

$$\mathsf{top}(\sigma) = \mathsf{f} \Rightarrow \mathsf{effect}(\mathsf{deq}(\mathsf{f}), \sigma) = \mathsf{tail}(\sigma) \qquad \mathsf{top}(\beta) = \mathsf{b} \Rightarrow \mathsf{effect}(\mathsf{deq}(\mathsf{b}), \beta) = \mathsf{tail}(\beta)$$

$$\mathsf{action}(\mathsf{error}, \varnothing) = \delta \qquad \mathsf{action}(\mathsf{error}, \varnothing) = \delta$$

$$\sigma \neq \varnothing \Rightarrow \mathsf{action}(\mathsf{error}, \sigma) = \mathsf{i} \qquad \beta \neq \varnothing \Rightarrow \mathsf{action}(\mathsf{error}, \beta) = \mathsf{j}$$

$$\mathsf{effect}(\mathsf{error}, \varnothing) = \varnothing \qquad \mathsf{effect}(\mathsf{error}, \varnothing) = \varnothing$$

$$\sigma \neq \varnothing \Rightarrow \mathsf{effect}(\mathsf{error}, \sigma) = \mathsf{tail}(\sigma) \qquad \beta \neq \varnothing \Rightarrow \mathsf{effect}(\mathsf{error}, \beta) = \mathsf{tail}(\beta)$$

Now we believe that this specification constitutes a correct protocol, if the queues behave fairly (i.e. data do not get lost infinitely many times in a row). At this point, we will not state explicitly what this statement means, so we will not give a specification that process CABP satisfies after abstraction of internal actions (the internal actions are the c_k actions and i, j).

Note that multi-actions can occur in this protocol also: we can add to a non-empty queue and read from it at the same time.

5. Asynchronous Communication

5.1. Definition

Let C be a finite collection of port names, and let D be a finite set of data. As special symbolic atomic actions we introduce, following [BKT85], for $\mathsf{c} \in \mathsf{C}$, $\mathsf{d} \in \mathsf{D}$:

$\mathsf{c} \uparrow \mathsf{d}$ potential send of message d along asynchronous channel c

$\mathsf{c} \Uparrow \mathsf{d}$ effectuated send of message d along asynchronous channel c

$\mathsf{c} \downarrow \mathsf{d}$ potential receive of message d along asynchronous channel c

$\mathsf{c} \Downarrow \mathsf{d}$ effectuated receive of message d along asynchronous channel c

We define $\mathsf{c} \Downarrow \mathsf{D} = \{\mathsf{c} \Downarrow \mathsf{d} \mid \mathsf{d} \in \mathsf{D}\}$, and likewise for the other actions. Besides these asynchronous communication actions, we have the standard synchronous communication actions $\mathsf{s}_k(\mathsf{d})$, $\mathsf{r}_k(\mathsf{d})$, $\mathsf{c}_k(\mathsf{d})$ as above.

5.2. State Operator

For asynchronous communication along a channel c, we define a special state operator λ^{c}. Here, the action and effect function will depend on the place and time of an action.

For each asynchronous channel $\mathsf{c} \in \mathsf{C}$, we will have a state operator λ^{c}. The set of states $\mathbb{V}$ is the collection of all finite subsets of $\mathsf{D} \times \mathbb{R}^3 \times \mathbb{R}_{\geqslant 0}$. Moreover, v is a given constant, the travelling speed of messages.

The state operator $\lambda^{\mathsf{c}}_{\mathbb{V}}(\mathsf{V} \in \mathbb{V})$ has functions

$$\mathsf{action}_{\mathsf{c}} : \mathsf{AST} \times \mathbb{V} \to \mathsf{AST} \qquad \mathsf{effect}_{\mathsf{c}} : \mathsf{AST} \times \mathbb{V} \to \mathbb{V}$$

given by

$$\begin{aligned}
&\mathrm{action}_c(c\uparrow d(x,t),V) = c \Uparrow d(x,t)\\
&\mathrm{effect}_c(c\uparrow d(x,t),V) = V \cup \{\langle d,x,t\rangle\}\\
&\mathrm{action}_c(c\downarrow d(y,r),V) = c \Downarrow d(y,r) && \text{if there is } \langle d,x,t\rangle \in V \text{ such that } |y-x| = v\cdot(r-t) \text{ and } r \neq t\\
&\mathrm{action}_c(c\downarrow d(y,r),V) = \delta(r) && \text{otherwise}\\
&\mathrm{effect}_c(c\downarrow d(y,r),V) = V && \text{for all a not of the form } c\uparrow d \text{ or } c\downarrow d\\
&\mathrm{action}_c(a(x,t),V) = a(x,t) && \text{for all a not of the form } c\uparrow d \text{ or } c\downarrow d\\
&\mathrm{effect}_c(a(x,t),V) = V
\end{aligned}$$

Now these functions are extended to multi-actions as given in Section 4. This extension is given by an ordering on located actions, but in fact it is immaterial which ordering is used. We leave it as an exercise to the reader to verify this fact. This means that we can write in general:

$$\begin{aligned}
&\mathrm{action}_c((a(x) \,\&\, \alpha)(t),V) = \mathrm{action}_c(a(x,t),V) \,\&\, \mathrm{action}_c(\alpha(t),V)\\
&\mathrm{effect}_c((a(x) \,\&\, \alpha)(t),V) = \mathrm{effect}_c(a(x,t),\mathrm{effect}_c(\alpha(t),V))
\end{aligned}$$

5.3. Example

Let us consider two processes S and R at fixed locations, communicating through an asynchronous channel 2 (see Fig. 5). 1 and 3 are synchronous ports. w_0 and w_1 are system delay constants. Suppose the distance between S and R is equal to the distance a message travels in 1 time unit.

We have the following specifications:

$$S = \int_{t>0} \sum_{d\in D} r1(d)(x,t)\cdot 2\uparrow d(x,t+w_0)\cdot((t+2w_0) \gg S)$$

$$R = \int_{t>0} \sum_{d\in D} 2\downarrow d(y,t)\cdot s3(d)(y,t+w_1)\cdot((t+2w_1) \gg R)$$

The system is now described by the expression $\lambda^2_{\emptyset}(S \parallel R)$.

After a certain $2\uparrow d(x,t_0)$-action at time t_0, the triple $\langle d,x,t_0\rangle$ is added to the state. Then, action $2\downarrow d(y,t_0+1)$ is transformed into $2 \Downarrow d(y,t_0+1)$, and all other $2\downarrow d(y,t)$ for $t \neq t_0+1$ are renamed to $\delta(t)$. However, the presence (in a sum context) of $\delta(t)$ with $t > t_0+1$ allows R to bypass the option $2 \Downarrow d(y,t_0+1)$, wait too long before trying to receive, and then it is too late, and no further action will be possible. Thus, we have to enforce that the action $2 \Downarrow d$ occurs as soon as it is possible.

This is a kind of *maximal progress* or *maximal liveness* assumption (cf. [ReR88]). We will ensure that communication takes place as soon as possible by use of the priority operator of [BBK86]. A similar operator was used by Jeffrey [Jef91b].

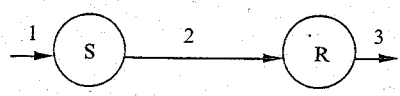

Fig. 5.

5.4. Priorities

Let us start by recalling the priority axioms from [BBK86]. There, an operator θ is defined, that cancels out all actions in a sum context, that do not have maximal priority. Priorities are given by a partial ordering on actions. Here, we have a very simple ordering: we have a certain set $H \subseteq A$, and all actions from H have priority over actions from outside H. In the example of Section 5.3, H is the set $2 \Downarrow D$.

Thus, on atomic actions A we have the following ordering:

$$h >_H a \Leftrightarrow h \in H, a \in A - H.$$

We extend this ordering to located actions in a straightforward way:

$$h(x) >_H a(y) \Leftrightarrow h >_H a$$
$$h(x) >_H \delta \Leftrightarrow h \in H$$

It gets more complicated when we go to multi-actions: if $\alpha, \beta \in MS$, then α has priority over β, $\alpha >_H \beta$, if α contains all H-actions of β, and at least one more, i.e. if we can write α in the form

$$\alpha \equiv h_0(x_0) \,\&\, \ldots \,\&\, h_n(x_n) \,\&\, a_0(y_0) \,\&\, \ldots a_k(y_k)$$

with $n > 0$, $k \geqslant 0$, $h_i \in H (i \leqslant n)$, $a_j \in A - H (j \leqslant k)$, all x_i, y_j different, then there is $m < n$ such that we can write

$$\beta \equiv h_0(z_0) \,\&\, \ldots \,\&\, h_m(z_m) \,\&\, b_0(w_0) \,\&\, \ldots b_l(w_l)$$

with $m + l > 0$, $b_j \in A - H (j \leqslant l)$, all z_i, w_j different.

5.5. Definition

Let a set $H \subseteq A$ be given. The *priority operator* θ_H has the axioms in Table 14 ($\alpha \in MS$).

The priority operator is axiomatised by means of the *unless operator* $\lhd_H$. In $X \lhd_H Y$, a starting action in Y containing a component from H will cancel all starting actions in X with a later timestamp, and all starting actions with the same timestamp that it has priority over. The unless operator is now axiomatised as shown in Table 15. In the axioms PI4, we use the set

$$W = \{w \geqslant 0 : \exists s \in V \alpha(t) \lhd_H P[s/v] = \delta(w)\}$$

Returning to the example of Section 5.3, we see that the correct expression for the system is as follows:

$$\theta_{2 \Downarrow D} \circ \lambda^2_{\varnothing}(S \parallel R)$$

Table 14. Priority operator

$\theta_H(\alpha(t)) = \alpha(t)$	(TH1)
$\theta_H(X+Y) = \theta_H(X) \lhd_H Y + \theta_H(Y) \lhd_H X$	(TH2)
$\theta_H(\alpha(t) \cdot X) = \alpha(t) \cdot \theta_H(t \geqslant X)$	(TH3)
$\theta_H\left(\int_{v \in V} P\right) = \int_{v \in V} \left(\theta_H(P) \lhd_H \int_{t \in V - \{v\}} P[t/v]\right)$	(TH4)

Table 15. Unless operator

$\alpha(t) \lhd_H \beta(t) = \delta(t)$	if $\beta >_H \alpha$	(P1 =)
$\alpha(t) \lhd_H \beta(t) = \alpha(t)$	if not $\beta >_H \alpha$	(P2 =)
$\alpha(t) \lhd_H \beta(s) = \delta(s)$	if $ats(\beta) \cap H \neq \emptyset$ and $s < t$	(P1 $\neq$)
$\alpha(t) \lhd_H \beta(s) = \alpha(t)$	if $ats(\beta) \cap H \neq \emptyset$ or $s > t$	(P2 $\neq$)
$X \lhd_H Y \cdot Z = X \lhd_H Y$		(P3)
$X \lhd_H (Y+Z) = (X \lhd_H Y) \lhd_H Z$		(P4)
$\alpha(t) \lhd_H \left(\int_{v \in V} P \right) = \delta(\inf W)$	if $W \neq \emptyset$	(PI4 $\neq$)
$\alpha(t) \lhd_H \left(\int_{v \in V} P \right) = \alpha(t)$	if $W = \emptyset$	(PI4 =)
$X \cdot Y \lhd_H Z = (X \lhd_H Z) \cdot Y$		(P5)
$(X+Y) \lhd_H Z = (X \lhd_H Z) + (Y \lhd_H Z)$		(P6)
$\left(\int_{t \in T} P \right) \lhd_H X = \int_{t \in T} (P \lhd_H X)$		(PI6)

Some examples on the use of these equations: if $h, k \in H$, $a, b \in A - H$, then

1. $\theta_H(h(x,t) + a(y,t)) = h(x,t) \lhd_H a(y,t) + a(y,t) \lhd_H h(x,t) = h(x,t) + \delta(t) = h(x,t)$
2. $\theta_H(a(x,t) + b(y,t)) = a(x,t) \lhd_H b(y,t) + b(y,t) \lhd_H a(x,t) = a(x,t) + b(y,t)$
3. $\theta_H(h(x,t) + k(y,t)) = h(x,t) \lhd_H k(y,t) + k(y,t) \lhd_H h(x,t) = h(x,t) + k(y,t)$
4. $\theta_H(h(x,t) + k(y,t+1)) = h(x,t) \lhd_H k(y,t+1) + k(y,t+1) \lhd_H h(x,t) = h(x,t) + \delta(t) = h(x,t)$
5. $\theta_H(a(x,t) + h(y,t+1)) = a(x,t) \lhd_H h(y,t+1) + h(y,t+1) \lhd_H a(x,t) = a(x,t) + h(y,t+1)$

5.6. Operational Semantics

We now proceed to give action rules for the priority operator and the unless operator. In Table 16, we have $\alpha, \beta \in MS - \{\delta\}$, $t, r > 0$. The expression $x \xrightarrow{\beta(t)}$ is a shorthand for: $x \xrightarrow{\beta(t)} x''$ for some x'' or $x \xrightarrow{\beta(t)} \surd$.

In order to determine the existence of δ-transitions, we need the ultimate delay of expressions with a priority or unless operator. Here, we cannot do this by means of axioms, and we are forced to infer the ultimate delay of such expressions by considering the transition system of simpler expressions. We get the rules of Table 17 ($\alpha, \beta \in MS$, so they may be δ!).

5.7. Priority Lock

A consequence of axioms TH4 and PI4 is that we have the following identity: $\theta_H(\int_{t \in (1,2)} h(t)) = \delta(1)$, if $h \in H$. This is because each h action is cancelled by one with lesser timestamp. We can call this phenomenon a *priority lock*.

It can be argued that the terms $\theta_H(\int_{t \in (1,2)} h(t))$ and $\delta(1)$ are not really the

Table 16. Action rules for priorities

$$\frac{x \xrightarrow{\alpha(t)} x',\ (x \xrightarrow{\beta(t)} \Rightarrow \neg\beta >_H \alpha),\ \forall r < t (x \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)}{\theta_H(x) \xrightarrow{\alpha(t)} \theta_H(x')}$$

$$\frac{x \xrightarrow{\alpha(t)} \surd,\ (x \xrightarrow{\beta(t)} \Rightarrow \neg\beta >_H \alpha),\ \forall r < t (x \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)}{\theta_H(x) \xrightarrow{\alpha(t)} \surd}$$

$$\frac{x \xrightarrow{\alpha(t)} x',\ (y \xrightarrow{\beta(t)} \Rightarrow \neg\beta >_H \alpha),\ \forall r < t (y \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)}{x \lhd_H y \xrightarrow{\alpha(t)} x'}$$

$$\frac{x \xrightarrow{\alpha(t)} \surd,\ (y \xrightarrow{\beta(t)} \Rightarrow \neg\beta >_H \alpha),\ \forall r < t (y \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)}{x \lhd_H y \xrightarrow{\alpha(t)} \surd}$$

Table 17. Ultimate delay for priorities

$U(\theta_H(x)) = \sup\{t: x \xrightarrow{\alpha(t)},\ \forall r < t (x \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)\}$
if this set is nonempty, $\inf\{t: x \xrightarrow{\alpha(t)}\}$ otherwise
$U(x \lhd_H y) = \sup\{t: x \xrightarrow{\alpha(t)},\ \forall r < t (y \xrightarrow{\beta(r)} \Rightarrow \mathsf{ats}(\beta) \cap H = \varnothing)\}$
if this set is nonempty, $\inf\{t: x \xrightarrow{\alpha(t)}\}$ otherwise

same, as the first term allows time to progress up to and including 1, whereas the second term allows time to progress up to but not including 1. Indeed, in the original semantics of [BaB91a], these two terms will be distinguished. However, the semantics we use here, based on [Klu91], is a little bit more abstract, and the two terms will be identified. If we allow a generalised merge operator, we have in both semantics that $\theta_H(\int_{t\in(1,2)} h(t)) = \|_{t\in(1,2)} \delta(t)$. Thus, in the original semantics of [BaB91a], axiom PI4$\neq$ should read

$$\alpha(t) \lhd_H \left(\int_{v\in V} P \right) = \left\| \right._{w\in W} \delta(w) \quad \text{if } W \neq \varnothing$$

For more on this issue, see Section 8.

5.8. Data Transmission via a Mobile Intermediate Station

We now discuss a larger example, a protocol transmitting data via a mobile intermediate station. Essentially, this is the example of Section 3.2, but now we allow the transmitter to move with time.

In Fig. 6, we have a sender S at location x_1, a receiver R at location x_3, and a transmitter T that is moving on a line between locations y_1 and y_2, starting at y_1 at time $t = 0$. The location of the transmitter is given by the following formula:

$$x_2(t) = y_1 + (\tfrac{1}{2} - \tfrac{1}{2}\cos(\omega t)) \cdot (y_2 - y_1)$$

The transmitter consists of two parts, a receiving part and a sending part, located

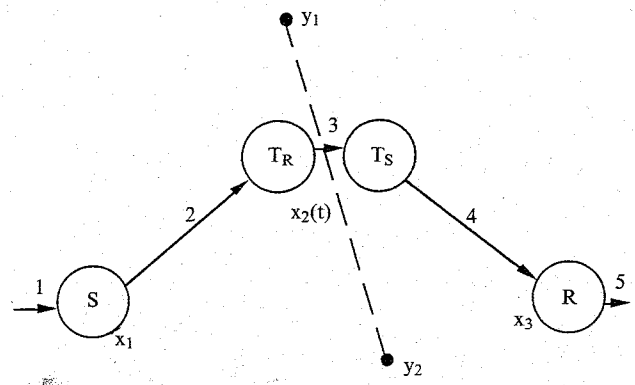

Fig. 6.

at $x_2(t)-\varepsilon$ and $x_2(t)+\varepsilon$, respectively. These parts are interconnected by a synchronous communication port 3. 1 and 5 are also synchronous ports, and 2 and 4 are asynchronous channels. We have system delay constants w_1, w_2, w_3, w_4.

The components are now given by the following specification:

$$S = \int_{t>0} \sum_{d\in D} r_1(d)\,(x_1,t)\cdot 2\uparrow d(x_1,t+w_1)\cdot((t+2w_1)\gg S)$$

$$T_R = \int_{t>0} \sum_{d\in D} 2\downarrow d(x_2(t)-\varepsilon,t)\cdot s_3(d)\,(x_2(t+w_2),t+w_2)\cdot((t+2w_2)\gg T_R)$$

$$T_S = \int_{t>0} \sum_{d\in D} r_3(d)\,(x_2(t),t)\cdot 4\uparrow d(x_2(t+w_3)+\varepsilon,t+w_3)\cdot((t+2w_3)\gg T_S)$$

$$R = \int_{t>0} \sum_{d\in D} 4\downarrow d(x_3,t)\cdot s_5(d)\,(x_3,t+w_4)\cdot((t+2w_4)\gg R)$$

Now the system is given by the identity

$$SYS_{dy} = \partial_{H(3)}(\theta_{2\Downarrow D}\circ\lambda^2_{\varnothing}(S\parallel T_R)\parallel\theta_{4\Downarrow D}\circ\lambda^4_{\varnothing}(T_S\parallel R))$$

The protocol works well if the data enter at a sufficiently low frequency.

5.9. The Spatial Replacement Operator: Structured Notation for a Mobile Process

Section 5.8 demonstrates a process T that is mobile in the sense that its actions take place at locations that vary in time. In this section, we will have a closer look at the description of such mobile processes.

Let $f:\mathbb{R}^3\times\mathbb{R}_{\geqslant 0}\to\mathbb{R}^3$ be a continuous function. The operator ρ_f modifies each action $a(x,t)$ of a process P into $a(f(x,t),t)$. Thus, at time t, all spatial coordinates are translated according to the mapping $f_t: x\mapsto f(x,t)$. The equations for ρ_f are very simple (Table 18). It is equally easy to come up with action rules for the replacement operator.

ρ_f is a renaming but in fact it affects spatial coordinates only, and not action labels. We prefer to call ρ_f a *(spatial) replacement operator*.

Table 18. Replacement operator

$\rho_f(\delta(t)) = \delta(t)$	(RRN1)
$\rho_f(a(x,t)) = a(f(x,t),t)$	(RRN2)
$\rho_f((\alpha \,\&\, \beta)(t)) = \rho_f(\alpha(t)) \,\&\, \rho_f(\beta(t))$	(RRN3)
$\rho_f(x+y) = \rho_f(x) + \rho_f(y)$	(RRN4)
$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$	(RRN5)
$\rho_f\left(\int_{t\in T} X\right) = \int_{t\in T} \rho_f(X)$	(RRN6)

In Section 5.8, we can rephrase the definition of T as follows: let x be a point in space. Then we can define

$$f(x,t) = x + (\tfrac{1}{2} - \tfrac{1}{2}\cos(\omega t)) \cdot (y_2 - y_1)$$

Now put

$$T_R^*(x) = \int_{t>r} \sum_{d\in D} 2 \downarrow d(x-\varepsilon, t) \cdot s_3(d)(x, t+w_2) \cdot ((t+2w_2) \gg T_R^*(x))$$

$$T_S^*(x) = \int_{t>r} \sum_{d\in D} r_3(d)(x,t) \cdot 4 \uparrow d(x+\varepsilon, t+w_3) \cdot ((t+2w_3) \gg T_S^*(x))$$

Now the processes in Section 5.8 are given by: $T_R = \rho_f(T_R^*(y_1))$, $T_S(t) = \rho_f(T_S^*(y_1))$

Using this notation we obtain a better modular structure. The system specification becomes

$$SYS = \partial_{H(3)}(\theta_{2 \Downarrow D} \circ \lambda_\varnothing^2(S \parallel \rho_f(T_R^*(y_1))) \parallel \theta_{4 \Downarrow D} \circ \lambda_\varnothing^4(\rho_f(T_S^*(y_1)) \mid R)).$$

Now suppose that the intermediate process T oscillates in a direction perpendicular to the segment (y_1, y_2) as well: let two points z_1, z_2 be given such that $(y_1, y_2) \perp (z_1, z_2)$. Define

$$g(x,t) = x + (\tfrac{1}{2} - \tfrac{1}{2}\cos(\omega^* t)) \cdot (z_2 - z_1)$$

then the combined motion of T is given by the expressions $\rho_g \circ \rho_f(T_R^*(y_1))$, $\rho_g \circ \rho_f(T_S^*(y_1))$

Thus, superposition of motions corresponds to composition of the corresponding replacement operators.

5.10. Signal Intensity

We proceed by discussing some variations in the communication mechanism. First of all, suppose we want to describe asynchronous communication with a decrease in signal intensity.

Assume that data are sent with intensity i_0 and that intensity decreases proportionally to the square of the distance travelled. So if i_0 is the signal intensity at distance 1 from the source, the signal intensity at distance l is $a \cdot i_0 \cdot l^{-2}$ for some constant $a \geqslant 0$. Let the receiving actions be equipped with an additional real parameter j that indicates at which intensity a signal can be received. Then the

action function will allow $c\downarrow(d,j)(y,r)$ provided there exist x, t such that d was sent at (x,t) and

1. $|y-x| = v\cdot(r-t)$
2. $a\cdot i_0\cdot|y-x|^{-2} = j$

A receiver that allows messages to be received between intensities lo, hi and between times fi, la will show integration over the intensity interval:

$$\text{Receiver} = \int_{j\in[lo,hi]}\int_{t\in[fi,la]}\sum_{d\in D} c\downarrow(d,j)(y(t),t)\cdot P(d,t)$$

5.11. Impenetrable Objects

Next, we look at how to model a signal transmission space with an impenetrable object. Let $B(2,z)$ be the closed ball with radius 2 and centre z. In this case, the action and effect functions work as follows:

$\text{action}_c(c\uparrow d(x,t),V) = \delta(t)$	if $x\in B(2,z)$ (i.e. if $\|x-z\|\leqslant 2$)
$\text{action}_c(c\uparrow d(x,t),V) = c \Uparrow d(x,t)$	if $x\notin B(2,z)$
$\text{effect}_c(c\uparrow d(x,t),V) = V$	if $x\in B(2,z)$
$\text{effect}_c(c\uparrow d(x,t),V) = V\cup\{\langle d,x,t\rangle\}$	if $x\notin B(2,z)$
$\text{action}_c(c\downarrow(d,j)(y,r),V) = c \Downarrow d(y,r)$	if there is $\langle d,x,t\rangle\in V$ such that $\|y-x| = v\cdot(r-t)$ and $a\cdot i_0\cdot\|y-x\|^{-2} = j$ and the line segment from y to x does not intersect $B(2,z)$
$\text{action}_c(c\downarrow(d,j)(y,r),V) = \delta(r)$	otherwise

As usual, $c\downarrow d$ actions have no effect on the state, so the effect function does not change the state.

5.12. Reflection

A more complex example is obtained if one allows reflection of the signal. Carrying on with the previous example, we allow the signal to be reflected at the surface of the ball $B(2,z)$. We assume perfect reflection without loss of signal intensity. Like before, we allow that a signal (or various signals) is simultaneously delivered at (y, r) through different paths. This can happen even with one source. It is possible that a signal d is sent by executing $c\uparrow d(x,t)$ and it reaches (y,r) along two different paths, both containing a reflection. In the case of the ball $B(2,z)$ this is not possible of course, but with two balls this could indeed happen. The action function remains the same in the case of $c\uparrow d(x,t)$ but for $c\downarrow(d,j)(y,r)$ we get

$$\text{action}_c(c\downarrow(d,j)(y,r),V) = c \Downarrow d(y,r)$$

if

1. There is $\langle d,x,t\rangle\in V$ such that $|y-x| = v\cdot(r-t)$ and $a\cdot i_0\cdot|y-x|^{-2} = j$ and the line segment from y to x does not intersect $B(2,z)$, or
2. There is $\langle d,x,t\rangle\in V$ and w on the boundary of $B(2,z)$ such that $|y-w|+|w-x|$

$= v \cdot (r-t)$, $a \cdot i_0 \cdot (|y-w|+|w-x|)^{-2} = j$, x, y, z, w are in one plane, and (z, w) bisects the angle (x, w, y)

$$\text{action}_c(c\downarrow(d,j)(y,r),V) = \delta(r) \quad \text{otherwise}$$

A state operator based on this action function describes an environment in which asynchronous communication through a non-trivial physical medium is supported. We remark that if the sending process also receives reflected signals, we are dealing with remote sensing, rather than asynchronous communication.

Interesting complications arise if we intend to model an asynchronous communication where, firstly, there are several independently moving objects that are impenetrable for signals but that do allow reflections with varying reflection coefficients, and, secondly, these objects have their motion guided by actions (in a discontinuous way) and by laws of classical mechanics between actions. This is in fact what happens if a sensor is used to assist a robot in navigating through a number of independently moving objects, each having its own surface characteristics. This is not just a matter of notational complexity. A mechanism is required that allows to track a moving object and to store its orbit in the state of a state operator. We elaborate on this further in Section 11 (paragraph 4).

It is our impression that the language of classical real space process algebra with priority and state operators is not sufficient to model the sensors needed for robot navigation in a dynamic environment.

We finish this section by specifying, in a number of steps, a more complicated version of the example of Section 5.7. We make the change that the communication between the sender and the transmitter, and also between the transmitter and the receiver, is sometimes blocked by an impenetrable object. This requires a communication protocol on both sides. For this, we use a Positive Acknowledgement with Retransmission (PAR) protocol. At no time, both communication links are open, so buffering is required in the transmitter. This situation occurs for instance when we want to describe communication to a place on the other side of the earth via an orbiting communication satellite. We make a couple of simplifying assumptions, but nevertheless, all necessary ingredients for this situation are present. In this way, we think we make a genuine contribution towards specification of real-life protocols.

5.13. PAR Protocol, Symbolic Synchronous Case

We start from the description of the *untimed* (*symbolic*) PAR protocol as given in [Vaa90a] (with some notational changes) (see Fig. 7). In order to avoid duplicates, messages from a set D labelled with an alternating bit from $B = \{0, 1\}$. The channels K2, L2 may lose messages, or send them on correctly. Upon timeout, the sender A will resend a message. Receiver B sends acknowledgement ack.

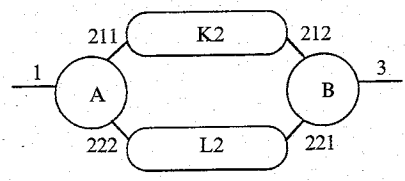

Fig. 7.

$$A = A(0)$$
$$A(b) = \sum_{d\in D} r_1(d)\cdot A(b,d) \qquad b = 0,1$$
$$A(b,d) = s_{211}(db)\cdot(r_{222}(ack)\cdot A(1-b) + timeout\cdot A(b,d)) \qquad b = 0,1,\ d\in D$$
$$K2 = \sum_{f\in D\times B} r_{211}(f)\cdot(s_{212}(f) + error_K)\cdot K2$$
$$L2 = r_{221}(ack)\cdot(s_{222}(ack) + error_L)\cdot L2$$
$$B = B(0)$$
$$B(b) = \sum_{d\in D} r_{212}(db)\cdot s_3(d)\cdot B^*(1-b) + \sum_{d\in D} r_{212}(d(1-b))\cdot B^*(b) \qquad b = 0,1$$
$$B^*(b) = s_{221}(ack)\cdot B(b) \qquad b = 0,1$$

Then, the symbolic synchronous PAR protocol is defined by

$$ssPAR = \partial_{H(211,212,221,222)}(A \parallel K2 \parallel L2 \parallel B).$$

This protocol is not correct. If timeouts are raised too early, no acknowledgements can get through. This problem can be solved in the symbolic case by using a priority operator (see [Vaa90a]). In the real time version in [BaB91a], the problem was solved by making the timeout period larger than the duration of one complete cycle. However, using a slightly more sophisticated protocol, any fixed non-zero lower bound for the timespan between timeouts suffices.

5.14. PAR Protocol, Symbolic Asynchronous Case

The next step is that we replace the channels with synchronous communication by asynchronous channels. We combine the asynchronous communication of Section 5.3 with the error mechanism of Example 4 in Section 4.7. We give the symbolic (untimed) version:

$$A = A(0)$$
$$A(b) = \sum_{d\in D} r_1(d)\cdot A^*(b,d) \qquad b = 0,1$$
$$A^*(b,d) = c_{21}\uparrow db\cdot(c_{22}\downarrow ack\cdot A(1-b) + timeout\cdot A^*(b,d)) \qquad b = 0,1,\ d\in D$$
$$E = (error_K + error_L)\cdot E$$
$$B = B(0)$$
$$B(b) = \sum_{d\in D} c_{21}\downarrow db\cdot s_3(d)\cdot B^*(1-b) + \sum_{d\in D} c_{21}\downarrow d(1-b)\cdot B^*(b) \qquad b = 0,1$$
$$B^*(b) = c_{22}\uparrow ack\cdot B(b) \qquad b = 0,1$$

The symbolic asynchronous PAR protocol is now given by

$$saPAR = \lambda^{21,22}_{\varnothing,\varnothing}(A \parallel E \parallel B)$$

Here, the state operator has state space $(\{\varnothing\} \cup \{db : d\in D, b\in B\}) \times \{\varnothing, ack\}$, and functions given by

$$action(c_{21}\uparrow db, \langle p,q\rangle) = c_{21} \Uparrow db \quad effect(c_{21}\uparrow db, \langle p,q\rangle) = \langle db, q\rangle$$
$$action(c_{22}\uparrow ack, \langle p,q\rangle) = c_{22} \Uparrow ack \quad effect(c_{22}\uparrow ack, \langle p,q\rangle) = \langle p, ack\rangle$$
$$action(c_{21}\downarrow db, \langle p,q\rangle) = \textbf{if } p = db \quad effect(c_{21}\downarrow db, \langle p,q\rangle) = \langle \varnothing, q\rangle$$
$$\textbf{then } c_{21} \Downarrow db \textbf{ else } \delta$$

$$\text{action}(c_{22}\downarrow \text{ack}, \langle p, q\rangle) = \textbf{if } p = \text{ack} \textbf{ then } c_{22} \Downarrow \text{ack} \textbf{ else } \delta \qquad \text{effect}(c_{22}\downarrow \text{ack}, \langle p, q\rangle) = \langle p, \varnothing\rangle$$

$$\text{action}(\text{error}_K, \langle p, q\rangle) = \textbf{if } p = \varnothing \textbf{ then } \delta \textbf{ else } \text{error}_K \qquad \text{effect}(\text{error}_K, \langle p, q\rangle) = \langle \varnothing, q\rangle$$

$$\text{action}(\text{error}_L, \langle p, q\rangle) = \textbf{if } q = \varnothing \textbf{ then } \delta \textbf{ else } \text{error}_L \qquad \text{effect}(\text{error}_L, \langle p, q\rangle) = \langle p, \varnothing\rangle$$

All other actions are inert (i.e. $\text{action}(a,s) = a$, $\text{effect}(a,s) = s$). The resulting protocol is very similar to ssPAR. In particular, it is incorrect in the same way. Indeed, if we have a renaming function f that has

$$f(c_{21}\uparrow db) = c_{211}(db),\ f(c_{22}\uparrow ack) = c_{221}(ack),$$
$$f(c_{21}\downarrow db) = c_{212}(db),\ f(c_{22}\downarrow ack) = c_{222}(ack)$$

(and leaves other actions fixed), then we have $\rho_f(\text{saPAR}) = \text{ssPAR}$
(for an explanation of the renaming operator, see e.g. [BaB88] or [BaW90]).

5.15. PAR Protocol, Timed Asynchronous Case

Now we add timing information to the asynchronous protocol of Section 5.14. The way we implement this, is that we will have the sender periodically check port 1, to see if new messages are available. If no message is available, an error will be read and discarded. Moreover, we will raise a transmission failure if the path through space the message should travel intersects a solid object, as in Section 5.11. The sender is located at u_1, the receiver at u_2. Later on, these locations will vary with time, using a spatial replacement operator as in Section 5.9. The solid object, a screen, is a line segment between locations z_1 and z_2 (fixed in space) (see Fig. 8).

$$A = A(0,0)$$

$$A(b,t) = r_1(\text{error}_A)(u_1, t+w_1)\,A(b, t+w_1) + \sum_{d\in D} r_1(d)(u_1, t+w_1)\cdot A^*(b,d,t+w_1) \qquad b = 0,1,\ t \geqslant 0$$

$$A^*(b,d,t) = c_{21}\uparrow db(u_1+\varepsilon_1, t+w_1)\cdot\Big(\int_{v\in(t,t+w_1+q)} c_{22}\downarrow ack(u_1-\varepsilon_1, v) \cdot A(1-b,v) + \text{timeout}(u_1, t+w_1+q)\cdot A^*(b,d,t+w_1+q)\Big)$$
$$b = 0,1,\ t \geqslant 0,\ d\in D$$

$$B = B(0)$$

$$B(b) = \int_{v\geqslant 0}\sum_{d\in D} c_{21}\downarrow db(u_2+\varepsilon_2, v)\cdot s_3(d)(u_2, v+w_2)\cdot B^*(1-b, v+w_2) + \int_{v\geqslant 0}\sum_{d\in D} c_{21}\downarrow d(1-b)(u_2+\varepsilon_2, v)\cdot B^*(b,v)$$

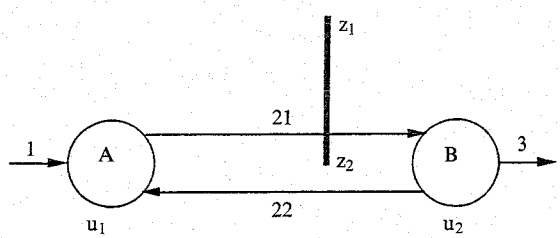

Fig. 8.

$$B^*(b,t) = c_{22}\uparrow ack(u_2-\varepsilon_2, t+w_2)\cdot B(b)$$
$$taPAR = \theta_{21\Downarrow D}\circ\lambda^{21}_{\varnothing}\circ\theta_{22\Downarrow D}\circ\lambda^{22}_{\varnothing}(A \parallel B)$$
$$action_{c21}(c_{21}\uparrow f(x,t),V) = c \Uparrow f(x,t) \quad f\in D\times B, x\in\mathbb{R}^3, t\geqslant 0, V\in\mathbb{V}_{21}$$
$$effect_{c21}(c_{21}\uparrow f(x,t),V) = V\cup\{\langle d,x,t\rangle\} \quad f\in D\times B, x\in\mathbb{R}^3, t\geqslant 0, V\in\mathbb{V}_{21}$$
$action_{c21}(c_{21}\downarrow f(y,r),V) = c \Downarrow f(y,r)$ if there is $\langle d,x,t\rangle\in V$ such that $|y-x| = v\cdot(r-t)$ and the line segment from y to x does not intersect the line segment from z_1 to z_2
$action_{c21}(c_{21}\downarrow f(y,r),V) = \delta(r)$ otherwise
$$effect_{c21}(c_{21}\downarrow f(y,r),V) = V$$
$$action_{c22}(c_{22}\uparrow ack(x,t),V) = c \Uparrow ack(x,t) \quad x\in\mathbb{R}^3, t\geqslant 0, V\in\mathbb{V}_{22}$$
$$effect_{c22}(c_{22}\uparrow ack(x,t),V) = V\cup\{\langle ack,x,t\rangle\} \quad x\in\mathbb{R}^3, t\geqslant 0, V\in\mathbb{V}_{21}$$
$action_{c22}(c_{22}\downarrow ack(y,r),V) = c \Downarrow ack(y,r)$ if there is $\langle d,x,t\rangle\in V$ such that $|y-x| = v\cdot(r-t)$ and the line segment from y to x does not intersect the line segment from z_1 to z_2

(Here, v is again the transmission speed of messages.)

$action_{c22}(c_{22}\downarrow ack(y,r),V) = \delta(r)$ otherwise
$$effect_{c21}(c_{21}\downarrow f(y,r),V) = V.$$

All other actions are inert. Again, the ordering on located actions is immaterial.

Crucial for the correct operation of the protocol is an appropriate choice for the constant q. This constant should be chosen larger than the maximum time that can pass between the sending of a message and the receipt of the acknowledgement. We see that the following choice works:

$$q = \frac{\sup\{|u_1(t)+\varepsilon_1-u_2(t')-\varepsilon_2| : t,t'\geqslant 0\}}{v} + \frac{\sup\{|u_1(t)-\varepsilon_1-u_2(t')+\varepsilon_2| : t,t'\geqslant 0\}}{v} + 2w_2+w_1$$

provided this supremum exists (v is the propagation speed of the message).

5.16. Data Transmission via a Mobile Intermediate Station Using an Unreliable Medium

Starting from the example of Section 5.8, we will put two screens in the picture, and use the timed asynchronous PAR protocol for the communication between S and T_R, and also for the communication between T_S and R. We obtain the required variants of the protocol by using the spatial replacement operator of Section 5.9, together with a simple renaming of port names. Since one of the two links is always blocked, we need buffering capacity in the transmitter. We picture the whole system in Fig. 9.

We start with the specification of the buffer. We assume it has input location $u-\varepsilon_3$ and output location $u+\varepsilon_3$. It is always ready to accept input and it is always

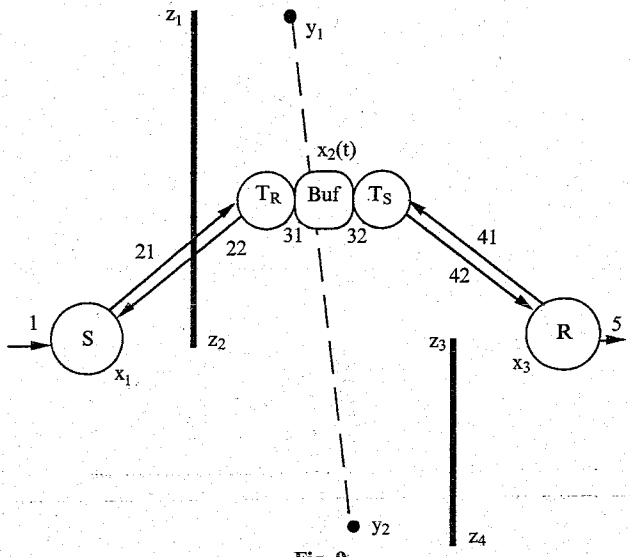

Fig. 9.

ready to supply an output (if empty, it will supply an error message). We have to mention explicitly the possibility that both actions happen at the same time.

$$\mathsf{Buf}(u) = \mathsf{Buf}(u, \varnothing)$$

$$\mathsf{Buf}(u, \varnothing) = \int_{t \geq 0} \sum_{d \in D} r_{31}(d)\,(u - \varepsilon_3, t) \cdot \mathsf{Buf}(u, d) + \int_{t \geq 0} s_{32}(\mathsf{error}_B)\,(u + \varepsilon_3, t) \cdot \mathsf{Buf}(u, \varnothing) + \int_{t \geq 0} \sum_{d \in D} (r_{31}(d)\,(u - \varepsilon_3) \;\&\; s_{32}(\mathsf{error}_B)\,(u + \varepsilon_3))\,(t) \cdot \mathsf{Buf}(u, d)$$

$$\mathsf{Buf}(u, \sigma d) = \int_{t \geq 0} \sum_{e \in D} r_{31}(e)\,(u - \varepsilon_3, t) \cdot \mathsf{Buf}(u, e\sigma d) + \int_{t \geq 0} s_{32}(d)\,(u + \varepsilon_3, t) \cdot \mathsf{Buf}(u, \sigma) + \int_{t \geq 0} \sum_{e \in D} (r_{31}(e)\,(u - \varepsilon_3) \;\&\; s_{32}(d)\,(u + \varepsilon_3))\,(t) \cdot \mathsf{Buf}(u, e\sigma)$$

for $\sigma \in D^*, d \in D$

Now we put the whole thing together. First the PAR protocol between the sender and the transmitter. We start by giving the spatial replacements:

Sender: $f_S(x, t) = x + x_1 - u_1$ (the sender is at fixed position x_1)

Transmitter: $f_{TR}(x, t) = x + y_1 - u_2 - \varepsilon_2 - \varepsilon_3$ (starting position)

$f_M(x, t) = x + (\frac{1}{2} - \frac{1}{2}\cos(\omega t)) \cdot (y_2 - y_1)$ (movement)

We need an action renaming operator in order to change port name 3 into 31. The

function g has $g(s_3(d)) = s_{31}(d)$, and leaves all other action names, and all times and places fixed. We get

$$S = \rho_{f_S}(A)$$
$$T_R = \rho_g \circ \rho_{f_M} \circ \rho_{f_{TR}}(B)$$
$$\text{leftPAR} = \theta_{21 \Downarrow D} \circ \lambda^{21}_{\emptyset} \circ \theta_{22 \Downarrow D} \circ \lambda^{22}_{\emptyset}(S \parallel T_R)$$

Here, A, B, the priority operators and the state operators are as in Section 5.15. We can take

$$q = \frac{\max_{i=1,2}(|x_1 + \varepsilon_1 - y_i - \varepsilon_2 + \varepsilon_3|) + \max_{i=1,2}(|x_1 - \varepsilon_1 - y_i + \varepsilon_2 + \varepsilon_3|)}{v} + 2w_2 + w_1$$

Next, the PAR protocol between the transmitter and the receiver. The spatial replacements are

Receiver: $f_R(x,t) = x + x_3 - u_2$
Transmitter: $f_{TS}(x,t) = x + y_1 - u_1 - \varepsilon_1 + \varepsilon_3$, f_M as above

This time, we need a more elaborate action renaming function h, given by

$$h(r_1(\text{error}_A)) = r_{32}(\text{error}_B), h(r_1(d)) = r_{32}(d), h(s_3(d)) = s_5(d),$$
$$h(c_{21} \uparrow f) = c_{41} \uparrow f, h(c_{21} \downarrow f) = c_{41} \downarrow f, h(c_{22} \uparrow \text{ack}) = c_{42} \uparrow \text{ack}, h(c_{22} \downarrow \text{ack}) = c_{42} \downarrow \text{ack}$$

The function h leaves all other action names, and all times and places fixed. We get

$$R = \rho_h \circ \rho_{f_R}(A)$$
$$T_S = \rho_h \circ \rho_{f_M} \circ \rho_{f_{TS}}(B)$$
$$\text{rightPAR} = \theta_{41 \Downarrow D} \circ \lambda^{41}_{\emptyset} \circ \theta_{42 \Downarrow D} \circ \lambda^{42}_{\emptyset}(T_S \parallel R)$$

The operators are as above, but in the definition of the action function, we have to replace z_1, z_2 by z_3, z_4. We can take

$$q = \frac{\max_{i=1,2}(|x_3 + \varepsilon_2 - y_i - \varepsilon_1 - \varepsilon_3|) + \max_{i=1,2}(|x_3 - \varepsilon_2 - y_i + \varepsilon_1 - \varepsilon_3|)}{v} + 2w_2 + w_1$$

For the buffer, we only have to put in the initial position and the movement, so the total system is now specified by

$$\text{SYSTEM} = \partial_{H(31,32)}(\text{leftPAR} \parallel \rho_{f_M}(\text{Buf}(y_1)) \parallel \text{rightPAR})$$

The transmitter can be in two modes: when it is in the receiving window (the bottom part of the picture), it can receive messages from the sender. Buffer capacity must be large enough to accommodate the maximum number of messages that can be sent during a single pass through the receiving window. When the transmitter is in the sending window (the top part of the picture) it will send as many messages as possible from the buffer to the receiver. Care must be taken that the sending window is as least as large as the receiving window, so that all messages received in the receiving window can be sent on. Otherwise, the buffer can become fuller at each pass, and (in practice, but not in our specification) capacity will run out at some point.

6. Asynchronous Message Passing Expressed Using Process Creation

In many cases, asynchronous message passing can be expressed using the available synchronous communication mechanism of ACP$\rho\sigma$. In this approach, which appears in many forms throughout the literature, a process is introduced that represents the medium through which the data are being transported.

In this section we will provide various examples of such processes, representing an asynchronous transport medium. It turns out that a real time version of the process creation mechanism of [Ber90] is useful to define a variety of such processes in a uniform way. For that purpose we describe process creation first, a mechanism which is of independent interest also.

It should be noted that the use of the state operator in the previous sections has been motivated by our inability to describe the particular form of asynchronous message transfer of Section 5.8 in the more traditional fashion of this section. This in no way excludes that an elegant description of the example of Section 5.8 based on synchronous communication is possible, but our search has been without success.

6.1. Process Creation

For the description of process creation, we assume that we have special disjoint subsets $\mathsf{cr}(\mathsf{D}) = \{\mathsf{cr}(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D}\}$ and $\overline{\mathsf{cr}}(\mathsf{D}) = \{\overline{\mathsf{cr}}(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D}\}$ within the set of symbolic atomic actions A, with $\mathsf{cr}(\mathsf{d}) \mid \mathsf{a} = \overline{\mathsf{cr}}(\mathsf{d}) \mid \mathsf{a} = \delta$ for all $\mathsf{d} \in \mathsf{D}$ and $\mathsf{a} \in \mathsf{A}_\delta$.

Further, we assume the existence of a function $\phi: \mathsf{D} \times \mathbb{R}^3 \times \mathbb{R}_{\geqslant 0} \to \mathsf{P}$, and we require ϕ to be defined by means of guarded recursion equations. Like in [Ber90], the function ϕ determines a process to be created from action $\mathsf{cr}(\mathsf{d})$. In the real time case, $\phi(\mathsf{d}, \mathsf{x}, \mathsf{t})$ represents the process created from the timed action $\mathsf{cr}(\mathsf{d})\ (\mathsf{x}, \mathsf{t})$.

Next we use, as in the symbolic case, an operator E_ϕ which enforces process creation from $\mathsf{cr}(\mathsf{d})$ actions occurring in its scope. The equations of E_ϕ are in Table 19. In order to state these axioms, we need some extra notation. We need to be able to determine the set $\mathsf{CR}(\alpha) = \{\langle \mathsf{d}, \mathsf{x} \rangle \in \mathsf{D} \times \mathbb{R}^3 \mid \mathsf{cr}(\mathsf{d}) \text{ occurs in } \alpha\}$ for a multi-action α. This set can be defined recursively as follows:

$$\mathsf{CR}(\mathsf{a}(\mathsf{x}) \,\&\, \alpha) = \mathsf{CR}(\alpha) \quad \text{if } \mathsf{a} \notin cr(D)$$
$$\mathsf{CR}(\mathsf{cr}(\mathsf{d})\ (\mathsf{x}) \,\&\, \alpha) = \{\langle \mathsf{d}, \mathsf{x} \rangle\} \cup \mathsf{CR}(\alpha)$$

Table 19. Process creation

$\mathsf{E}_\phi(\alpha(\mathsf{t})) = \alpha(\mathsf{t}) \quad \text{if } \mathsf{CR}(\alpha) = \varnothing$		(PCT1)
$\mathsf{E}_\phi(\alpha(\mathsf{t})) = \bar{\alpha}(\mathsf{t}) \cdot \mathsf{E}_\phi\left(\left\|\!\!\left\|_{\langle \mathsf{d},\mathsf{x}\rangle \in \mathsf{CR}(\alpha)} \phi(\mathsf{d}, \mathsf{x}, \mathsf{t})\right.\right\|\!\!\right)$	otherwise	(PCT2)
$\mathsf{E}_\phi(\alpha(\mathsf{t}) \cdot \mathsf{X}) = \alpha(\mathsf{t}) \cdot \mathsf{E}_\phi(\mathsf{X}) \quad \text{if } \mathsf{CR}(\alpha) = \varnothing$		(PCT3)
$\mathsf{E}_\phi(\alpha(\mathsf{t}) \cdot \mathsf{X}) = \bar{\alpha}(\mathsf{t}) \cdot \mathsf{E}_\phi\left(\mathsf{X} \,\|\!\!\left\|\!\!\left\|_{\langle \mathsf{d},\mathsf{x}\rangle \in \mathsf{CR}(\alpha)} \phi(\mathsf{d}, \mathsf{x}, \mathsf{t})\right.\right\|\!\!\right)$	otherwise	(PCT4)
$\mathsf{E}_\phi(\mathsf{x}+\mathsf{y}) = \mathsf{E}_\phi(\mathsf{x}) + \mathsf{E}_\phi(\mathsf{y})\}$		(PCT5)
$\mathsf{E}_\phi\left(\int_{\mathsf{t}\in\mathsf{T}} \mathsf{X}\right) = \int_{\mathsf{t}\in\mathsf{T}} \mathsf{E}_\phi(\mathsf{X})$		(PCT6)

Also, we need the notation $\overline{\alpha}$ for the multi-action α, where all $cr(d)$ actions are changed into $\overline{cr}(d)$ actions, i.e.

$$\overline{a(x) \,\&\, \alpha} = a(x) \,\&\, \overline{\alpha} \quad \text{if } a \notin cr(D)$$
$$\overline{cr(d)\,(x) \,\&\, \alpha} = \overline{cr}(d)\,(x) \,\&\, \overline{\alpha}$$

The operational semantics of the process creation operator now follows easily on the basis of the axioms. Note that the process creation mechanism can be used in real time (without space) just as well. All examples of [Ber90] can be adapted to a real time setting. We concentrate here on examples concerning message transport media.

6.2. Message Handler

The following recursive specification defines a process that, upon receiving an input, creates a message handler to take care of the input:

$$M = \int_{t>0} \sum_{d \in D} r(d)\,(x(t), t) \cdot cr(d)\,(x(t+t_1), t+t_1) \cdot ((t+t_1+t_0) \gg M)$$

Here t_0 is a parameter that determines a minimum delay between the creation of the message handler and reception of a new input, whereas t_1 determines the delay between reception of a datum and creation of the corresponding handler. $x(t)$ is some function from $\mathbb{R}_{\geqslant 0}$ to $\mathbb{R}^3$ that determines the input location of the medium. We assume that the handler is created at that very same logical location but with a delay of t_1 (i.e. at $x(t+t_1)$).

Next, we provide possible functions $\phi_n : D \times \mathbb{R}^3 \times \mathbb{R}_{\geqslant 0} \to P$ that determine the handler created. This leads to an asynchronous message transport medium $M_n = E_{\phi_n}(M)$ for each case. In all cases, $y(x, r)$ is the output location of the medium at time $t+r$ if at time t the input location is x. Thus, input at (x, t) leads to output at $(y(x, \Delta), t+\Delta)$ for some $\Delta > 0$.

Case 1: $\phi_1(d, x, t) = s(d)\,(y(x, \Delta_1), t+\Delta_1)$. In this case data need a constant time Δ_1 to travel through the medium. Moreover, the medium introduces no errors or omissions.

Case 2: $\phi_2(d, x, t) = s(d)\,(y(x, \Delta_2(t)), t+\Delta_2(t))$. In this case, the transmission time depends on t. This happens e.g. if the output location is moving relative to the input location. Notice that the difference with Section 5.8 is that there, due to the broadcasting nature of the mechanism, the output location is not known in the same way. In particular, the function $\Delta_2(t)$ is not easy to define in that example, even if one assumes that broadcasting is irrelevant and each message is received exactly once. This is caused by the fact that in Section 5.8 the output location (i.e. location (x, t) at which an action $c \Downarrow d(x, t)$ happens) must be derived from the combined behaviour of *two* processes.

Case 3: $\phi_3(d, x, t) = \int_{r \in (t+\Delta_1-\varepsilon_1,\, t+\Delta_1+\varepsilon_1)} s(d)\,(y(x, r-t), r)$. M_3 is like M_1, be it that there is a tolerance ε_1 in the arrival time. If $\varepsilon_1 > \frac{1}{2}(t_0+t_1)$, complications can arise because M_3 may deliver different messages at the same time and place. Assuming $\varepsilon_1 < \frac{1}{2}(t_0+t_1)$, this unwanted interference is not possible.

Next, we assume that the orbit of a message d travelling from (x, t) to $(y(x, \Delta), t+\Delta)$ is given by $(z(r-t), r)$, so in particular, we have $(x, t) = (z(0), t+0)$ and $(y(x, \Delta), t+\Delta) = (z(\Delta), t+\Delta)$.

Case 4: $\phi_4(d,x,t) = s(d)\,(y(x,\Delta_1), t+\Delta_1) + \int_{r\in(0,\Delta_1)} lost(z(r), t+r)\cdot s(\perp)\,(y(x,\Delta_1), t+\Delta_1)$. This example allows the datum to be changed into $\perp$ anywhere during transmission. In particular, M_4 keeps track where a datum gets lost. In the next example, M_5 will not deliver lost data.

Case 5: $\phi_5(d,x,t) = s(d)\,(y(x,\Delta_1), t+\Delta_1) + \int_{r\in(0,\Delta_1)} lost(z(r), t+r)$.

Case 6: $\phi_6(d,x,t) = s(d)\,(y(x,\Delta_1), t+\Delta_1) + \int_{r\in(0,\Delta_1)} lost(z(r), t+r)\cdot cr(d)\,(x(t+r+\varepsilon_2), t+r+\varepsilon_2)$. M_6 will spontaneously retransmit lost data (very unlikely in practice). Note that this protocol can lead to collisions of different cr actions.

7. Asynchronous Message Passing Using Mode Transfer

7.1. A Faulty Queue for Asynchronous Message Transfer

In this section we will start out from the following description of a queue that transports data in D from x to y in time Δ:

$$Q = \int_{t>0} \sum_{d\in D} r(d)\,(x,t)\cdot(s(d)\,(y,t+\Delta) \parallel Q)$$

This queue is the classical real space absolute time version of example 10.1 of [BaB91a], with port names 1,2 replaced by locations x, y.

We intend to describe a related queue that shows faults. In order to do this, the mode transfer operators $\rightarrow$ and $\hookrightarrow$ of [Ber89] are casted in a real time and space setting. Informally, $X \rightarrow Y$ is a process that behaves like X but can at any time start behaving like Y (provided it decides to do so before X has terminated). $X \hookrightarrow Y$ is like $X \rightarrow Y$ with the additional constraint that its first action must be taken from X.

Using the mode transfer operator (to be discussed in detail below), a faulty version of the queue can be given as follows: let $r\in\mathbb{R}$, $n\in\mathbb{N}$, define

$$Y(r,n) = s(\perp)\,(y, n\cdot r)\cdot Y(r, n+1).$$

$Y(r,n)$ will produce erroneous signals at regular intervals. Now define Q' by

$$Q' = Q \rightarrow \sum_{n\in\omega} Y(r,n)$$

This process allows the queue to transmit data until it switches to mode Y.

Finally, we describe a queue that can be put back in the original mode by means of an action restart(z, t). Here, z is a location from which the queue is controlled.

$$Q'' = Q \rightarrow \left(\sum_{n\in\omega} Y(r,n) \hookrightarrow \int_{t>0} restart(z,t)\cdot Q'' \right)$$

7.2. Mode Transfer

The equations for the mode transfer operators are given in Table 20 ($\alpha\in MS$).

Action rules for the mode transfer operators can be given as shown in Table 21 ($\alpha\in MS-\{\delta\}$). In order to derive δ-transitions, we add the identity

$$U(X\rightarrow Y) = U(X \hookrightarrow Y) = U(X).$$

Table 20. Mode transfer

$\alpha(t) \to X = \alpha(t) + (X \gg t)$	(MTT1)
$\alpha(t) \cdot X \to Y = \alpha(t) \cdot (X \to Y) + (Y \gg t)$	(MTT2)
$(X+Y) \to Z = (X \to Z) + (Y \to Z)$	(MTT3)
$\left(\int_{t \in T} X\right) \to Y = \int_{t \in T} (X \to Y)$ (t not free in Y)	(MTT4)
$\alpha(t) \hookrightarrow X = \alpha(t)$	(DMTT1)
$\alpha(t) \cdot X \hookrightarrow Y = \alpha(t) \cdot (X \to Y)$	(DMTT2)
$(X+Y) \hookrightarrow Z = (X \hookrightarrow Z) + (Y \hookrightarrow Z)$	(DMTT3)
$\left(\int_{t \in T} X\right) \hookrightarrow Y = \int_{t \in T} (X \hookrightarrow Y)$ (t not free in Y)	(DMTT4)

Table 21. Action rules for mode transfer

$\dfrac{x \xrightarrow{\alpha(t)} x'}{(x \to y) \xrightarrow{\alpha(t)} (x' \to y)}$	$\dfrac{x \xrightarrow{\alpha(t)} \surd}{(x \to y) \xrightarrow{\alpha(t)} \surd}$
$\dfrac{y \xrightarrow{\alpha(t)} y',\ U(x) > t}{(x \to y) \xrightarrow{\alpha(t)} y'}$	$\dfrac{y \xrightarrow{\alpha(t)} \surd,\ U(x) > t}{(x \to y) \xrightarrow{\alpha(t)} \surd}$
$\dfrac{x \xrightarrow{\alpha(t)} x'}{(x \hookrightarrow y) \xrightarrow{\alpha(t)} (x' \to y)}$	$\dfrac{x \xrightarrow{\alpha(t)} \surd}{(x \hookrightarrow y) \xrightarrow{\alpha(t)} \surd}$

7.3. Remark

The equations for mode transfer in [Ber89] are as follows:

$$a \to X = a + X$$
$$\delta \to X = \delta$$
$$a \cdot X \to Y = a \cdot (X \to Y) + Y$$
$$(X+Y) \to Z = (X \to Z) + (Y \to Z)$$

In particular the equation $\delta \to X = \delta$ is not obvious. However, if a system has deadlocked, it seems impossible to recover from that deadlock in a context $X \to Y$. In the real time case, we see a more differentiated picture:

$$\delta(2) \to a(3) \cdot b(4) = \delta(2)$$

while

$$\delta(2) \to a(1) \cdot b(4) = \delta(2) + a(1) \cdot b(4)$$

Likewise

$$c(2) \to a(3) \cdot b(4) = c(2) \text{ and}$$
$$c(2) \to a(1) \cdot b(4) = c(2) + a(1) \cdot b(4)$$

8. On the Variety of Semantic Options for Time Stops

The description of deadlocks (time stops) in real time or real time and space process algebra shows a degree of freedom, as hinted at in Section 5.7. Within the setting of bisimulation semantics several semantic options exist. In this section we will survey several of these options and motivate our present choice. It should be emphasised that time stops are theoretical entities that do not model real systems, therefore their merits are to be judged in terms of the efficiency or clarity of the resulting formalism.

8.1. A Taxonomy

We have the following processes in real time process algebra (disregarding space for the moment).

- δ Full time stop. No activity is possible, time cannot progress.
- $\delta(t)$ Open time stop. No action can take place, and time can progress up to time t, but time t itself cannot be reached.
- $\hat{\delta}(t)$ Closed time stop. No action can take place, and time can progress up to and including time t.
- $a(t) \cdot \delta$ Dead termination. Action a is executed at time t, and the process immediately terminates unsuccessfully.

8.2. Semantics of [BaB91a]

The real time process algebra as described in [BaB91a] has the full time stop, open time stop and dead termination. Moreover, we have the following identification:

$$\delta = \delta(0) \quad \textit{full time stop} = \textit{open time stop at 0}$$

The closed time stop does not occur in the setting of real time ACP. However, several expressions can be found that equal $\hat{\delta}(t)$ in the model of [BaB91a] (see Section 5.7):

$$\Big\|_{r>t} \delta(r) = \hat{\delta}(t)$$

$$\theta_{\{a\}}\Big(\int_{r>t} a(r)\Big) = \hat{\delta}(t)$$

These equations are both plausible. In the operational semantics, we have the following axioms for time stops:

$$t < s < r \Rightarrow \langle \delta(r), t\rangle \to \langle \delta(r), s\rangle \quad \text{and}$$
$$t < s \leqslant r \Rightarrow \langle \hat{\delta}(r), t\rangle \to \langle \hat{\delta}(r), s\rangle$$

On the basis of this semantics, we can derive the following laws:

$$\hat{\delta}(0) = \delta(0)$$
$$r > t \Rightarrow a(r) + \hat{\delta}(t) = a(r)$$
$$\hat{\delta}(t) + \delta(t) = \hat{\delta}(t)$$
$$\hat{\delta}(t) = \hat{\delta}(t) \cdot \delta$$

$$\mathsf{a}(t)\cdot\delta = \mathsf{a}(t)\cdot\hat{\delta}(t)$$
$$\hat{\delta}(t)\,|\,\mathsf{a}(t) = \hat{\delta}(t)\,|\,\delta(t) = \delta(t)$$
$$\hat{\delta}(t)\,\lfloor\,\hat{\delta}(t) = \hat{\delta}(t)$$
$$\partial_H(\hat{\delta}(t)) = \hat{\delta}(t).$$

8.3. Negative Time

In the presence of negative time stamps (as e.g. in [BaB91b]) the identification *full time stop = open time stop at 0* is not justified. This is because $\mathsf{x}+\delta = \mathsf{x}$ for the full time stop δ, but $\delta(0)+\mathsf{a}(-1) \neq \mathsf{a}(-1)$, because $\delta(0)+\mathsf{a}(-1)$ can wait past time -1. Consequently, the very presence of a full time stop in a theory with negative time becomes an option.

8.4. Semantics of the Present Paper

In the present paper we have the full time stop, open time stop and dead termination. The operational semantics, following [Klu91], has the following identifications:

$\delta = \delta(0)$ *full time stop = open time stop at 0*
$\delta(t) = \hat{\delta}(t)$ *open time stop = closed time stop*

The identification of closed and open time stops is a useful simplification, though certainly not a necessary one. As we saw no advantage in distinguishing open and closed time stops, we have adopted the model of [Klu91] which identifies them. Thus, our motivation was to simplify the mathematics of the model. Next, three further semantic alternatives can be mentioned. As none of these further identifications presents immediate benefits in our setting, we have not imposed these identifications on our model.

8.5. Further Optional Identifications

1. $\delta = \delta(t)$ *time stop = full time stop.*
   This simplifies dealing with time stops somewhat. It brings the theory closer to that of symbolic ACP. This option was adopted in [BaB91b]. It is a reasonable alternative as it simplifies axioms and operational rules, but is less attractive when viewed from our operational intuition.
2. $\mathsf{a}(t)\cdot\delta = \delta(t)$ *dead termination = time stop.*
   This is a possible conceptual simplification. However, it brings the theory further apart from ACP, which is reason not to adopt it.
3. Zero process. This is what we get if *time stop = full time stop* and *dead termination = time stop* are combined:

   $$\mathsf{a}(t)\cdot\delta = \delta(t) = \delta.$$

   Thus, δ acts as a zero (i.e. $\mathsf{x}\cdot\delta = \delta\cdot\mathsf{x} = \delta$), see [BaB91b]. This seems to be necessary in the setting of relativistic space and time.

8.6. Real Space: Located Closed Time Stops

In the real space setting, the introduction of closed time stops adds more options. Indeed, if one accepts closed time stops $\hat{\delta}(t)$ then these may just as well be located: $\hat{\delta}(x,t)$. The identification $\hat{\delta}(x,t) = \hat{\delta}(t)$ expresses the semantic assumption that closed time stops are not located. If one distinguishes the various $\hat{\delta}(x,t)$ for $x \in \mathbb{R}^3$, new multi-actions appear, such as

$$(\hat{\delta}(x_0) \,\&\, a(x_1) \,\&\, b(x_2) \,\&\, \hat{\delta}(x_3))\,(t).$$

The difference between $\hat{\delta}(x_0) \,\&\, a(x_1)$ and $a(x_1)$ comes about as follows:

$$(\hat{\delta}(x_0) \,\&\, a(x_1)) \,|\, b(x_0) = \delta(x_0) \,\&\, a(x_1) = \delta$$
$$a(x_1) \,|\, b(x_0) = a(x_1) \,\&\, b(x_0).$$

As said above, at the time of writing this paper, we are not aware of any clear advantage in distinguishing open and closed time stops, let alone in distinguishing various located closed time stops.

9. Relative Time Notation

9.1. Relative Time in Real Time, No Real Space

First we concentrate on the real time case, without space parameters. In some cases, relative time notation is profitable. In [BaB91a], we have introduced a relative time notation as follows: $a[t]$ denotes action a t time units after the previous atomic action or, if such action doesn't exist, after system initialisation. In this section, we will deal with relative time in a different way. The basis for this approach to relative time notation is the *initial abstraction operator*. Let for $t \in \mathbb{R}_{\geqslant 0}$, $F(t)$ be a process in P, then

$$\surd t.\, F(t)$$

denotes a process that, when started at time r, proceeds as $F(r)$. Semantically, $\surd t.\, F(t)$ is just a function f from $\mathbb{R}_{\geqslant 0}$ into P that satisfies $f(t) = t \gg f(t)$. We make things more precise in the following.

9.2. Definition

Let P be the sort of processes. Put

$$P^* = \{f : f : \mathbb{R}_{\geqslant 0} \to P \text{ such that } f(r) = r \gg f(r) \text{ for all } r \geqslant 0\}$$

We introduce the initial abstraction operator $\surd : T \times P^* \to P^*$, where T is a set of variables ranging over $\mathbb{R}_{\geqslant 0}$ called *time variables*. In the expression $\surd t.\, F$, free occurrences of t in F become bound. Semantically, we have the following. Let M_A be the bisimulation model of Sections 2.9–2.11. We extend this model to a model M_A^* with domain

$$|M_A^*| = \{f : \mathbb{R}_{\geqslant 0} \to |M_A| : \text{for all } t \geqslant 0 \text{ we have } f(t) = t \gg f(t)\}$$

The operators are defined on this domain as follows. We use λ-notation for

functions. Note that for each $F \in |M_A^*|$, the function application $F(t) \in |M_A|$ is an equivalence class of process trees.

$a(t) = \lambda r.r \gg a(t)$ for $a \in A_\delta$
$a[t] = \lambda r.a(r+t)$ for $a \in A_\delta$
$F \square G = \lambda t.(F(t) \square G(t))$ for $\square = +, \|, \lfloor\!\lfloor, |$
$F \cdot G = \lambda t.(F(t) \cdot G$ where $F(t) \cdot G$ is the process obtained as follows: after each successfully terminating branch of $F(t)$, say ending in $a(r)$, append a process tree for $G(r)$
$t \gg F = F(t)$
$F \gg t = \lambda r.r \gg (F(r) \gg t)$
$U(F) = U(F(0))$.

Here, $a[t]$ is the square bracket notation of [BaB91a].

9.3. Axioms

All axioms of real time and space process algebra with integration presented before remain valid, with the exception of axioms ATCM2,3 of Table 6, that have to be adapted slightly to the present setting. Note that the new axioms ATCM2*,3* reduce to the old axioms if we limit ourselves to expressions not involving initial abstraction operators.

On top of the old axioms, we have the extra axioms in Table 22, governing the

Table 22. Axioms for initial abstraction

$\alpha(t) \lfloor\!\lfloor X = \surd r.\alpha(t) \gg U(r \gg X)) \cdot (r \gg X)$	(ATCM2*)
$\alpha(t) \cdot X \lfloor\!\lfloor Y = \surd r.\alpha(t) \gg U(r \gg Y)) \cdot (X \| (r \gg Y))$	(ATCM3*)
$a[r] = \surd t.a(t+r)$	(IA0)
$\surd s.G = \surd t.G[t/s]$	(IA1)
$s \gg \surd t.F = s \gg F[s/t]$	(IA2)
$\surd t.\surd r.F = \surd t.F[t/r]$	(IA3)
$G = \surd t.G$	(IA4)
$\forall t \geqslant 0\ t \gg X = t \gg Y \Rightarrow X = Y$	(IA5)
$(\surd t.F) + G = \surd t.(F + t \gg G)$	(IA6)
$(\surd t.F) \cdot G = \surd t.(F \cdot G)$	(IA7)
$(\surd t.F) \lfloor\!\lfloor G = \surd t.(F \lfloor\!\lfloor t \gg G)$	(IA8)
$G \lfloor\!\lfloor (\surd t.F) = \surd t.(t \gg G \lfloor\!\lfloor F)$	(IA9)
$(\surd t.F) \mid G = \surd t.(F \mid t \gg G)$	(IA10)
$G \mid (\surd t.F) = \surd t.(t \gg G \mid F)$	(IA11)
$\partial_H(\surd t.F) = \surd t.\partial_H(F)$	(IA12)
$(\surd t.F) \gg r = \surd t.(F \gg r)$	(IA13)
$U(F) = U(0 \gg F)$	(IA14)
$\int_{v \in V} \surd t.F = \surd t.\left(\int_{v \in V} F\right)$ if $t \neq v$	(IA15)
$\rho_f(\surd t.F) = \surd t.\rho_f(F)$	(IA16)

use of initial abstraction. In all axioms in Table 22, it is assumed that t is not free in G. Axiom IA0 is the definition of the square bracket notation, axiom IA1 gives α-conversion, axiom IA2 achieves β-conversion, i.e. function application. IA4 gives the embedding of absolute time notation processes in the extended domain. IA5 is an extensionality axiom.

9.4. Examples

1. $a(3)\cdot(\surd t.\surd r.a(t+r+1)) = a(3)\cdot(3 \gg \surd t.\surd r.a(t+r+1))$
   $= a(3)\cdot(3 \gg \surd r.a(r+4)) = a(3)\cdot(3 \gg a(7)) = a(3)\cdot a(7).$
2. $a(t)\cdot b[r] = a(t)\cdot(\surd s.b(r+s)) = a(t)\cdot(t \gg \surd s.b(r+s))$
   $= a(t)\cdot(t \gg b(r+t)) = a(t)\cdot b(r+t).$
3. We can calculate that $a(t)\cdot\partial_H(b[r]\cdot X \parallel Y) = a(t)\cdot\partial_H(b(t+r)\cdot X \parallel Y)$. Choose a fresh variable v (i.e. a time variable not occurring in X or Y). Then

   $$\begin{aligned} a(t)\cdot\partial_H(b[r]\cdot X \parallel Y) &= a(t)\cdot\partial_H((\surd v.b(v+r))\cdot X \parallel Y) \\ &= a(t)\cdot\partial_H((\surd v.b(v+r)\cdot X) \parallel \surd v.Y) = a(t)\cdot\partial_H(\surd v.(b(v+r)\cdot X \parallel Y)) \\ &= a(t)\cdot\surd v.\partial_H(b(v+r)\cdot X \parallel Y) = a(t)\cdot(t \gg \surd v.\partial_H(b(v+r)\cdot X \parallel Y)) \\ &= a(t)\cdot(t \gg \partial_H(b(t+r)\cdot X \parallel Y)) = a(t)\cdot\partial_H(b(t+r)\cdot X \parallel Y). \end{aligned}$$

4. We can prove for all closed terms that $s \gg s \gg X = s \gg X$. We can use this to prove the following identity: $\surd t.F = \surd t.(t \gg F)$.
   For, take $s \geqslant 0$, then $s \gg \surd t.F = s \gg F[s/t] = s \gg (s \gg F[s/t]) = s \gg (\surd t.t \gg F)$. The result follows by extensionality (IA5).
5. We consider again a couple of clocks. First a perfect clock, that can stop at any moment:

   $$C1 = tick[1]\cdot C1 + \int_{0\leqslant t<1} exit[t]$$

   Next, we initialise C1 so that it always starts at an integer time value. [t] is the entier of t, the largest integer less than or equal to t.

   $$C2 = (\surd t.start([t]+1))\cdot C1$$

9.5. Rigidity

We call a process X *rigid* iff $X = 0 \gg X$. It can be seen that the rigid processes are exactly the processes that can be written with absolute time notation only. We have the following proposition, that can be called the *unique initialisation property*:

$t \gg X$ is rigid

Another observation: X is rigid iff $\forall s,t(s > t \Rightarrow s \gg t \gg X = s \gg X)$. From this it follows that we have for all X that $\forall s,t,r(s > t > r \Rightarrow s \gg t \gg r \gg X = s \gg r \gg X)$.

9.6. Elimination

In closed process expressions with a restricted form of integration, we can eliminate the operators $\parallel, \between, |, \partial_H, t \gg, \gg t$. [FoK92] explains in detail how such an elimination

theorem can be obtained formally. They use the special case of prefixed integration. Generalisation of their results to a setting with initial abstraction seems unproblematic.

9.7. Adding Space Coordinates

We can now extend the relative time notation above by also taking space coordinates into account. We have two options: we can use also relative space notation, or we can use absolute space notation only. In the first option, we need initial time and space abstraction, so that, e.g.

$$a(x,t) \cdot b[y,r] = a(x,t) \cdot b(x+y,t+r).$$

As an example, consider a clock at fixed position x_0, starting at time t_0:

$$\mathsf{Clock} = \mathsf{tick}(x_0,t_0) \cdot \mathsf{Clock}^*$$
$$\mathsf{Clock}^* = \mathsf{tick}[0,1] \cdot \mathsf{Clock}^*.$$

If now we want to consider this clock moving along path $t \mapsto f(t)$, with $f(t_0) = x_0$, we can use the spatial replacement operator of Section 5.9, and write $\rho_g(\mathsf{Clock})$ with $g(x,t) = x - x_0 + f(t)$.

We will not provide the semantic details of relative space notation, because its applicability seems quite limited, and instead focus on the notation with relative time and absolute space. In this case, we can extend the theory above in a very straightforward manner, and obtain actions as $a(x)$ [t]. We will limit ourselves to giving a non-trivial example. First, a few remarks on real space algebra over a finite space.

9.8. Real Space Algebra Over a Finite Space

Let $V \subseteq \mathbb{R}^3$ be a finite set. Let M_A be the bisimulation model of Section 2.9–2.11. By $M_{A,V}$ we denote the subalgebra of M_A that contains all processes which have all their (multi-)actions (apart from $\delta(t)$ actions) located on V. Clearly, if the cardinality of V and W is the same, then $M_{A,V}$ is isomorphic to $M_{A,W}$. Further, if V contains exactly one element, then $M_{A,V}$ is just a real time process algebra in the sense of [BaB91a]. This description replaces (and improves upon) the treatment of multi-actions of [BaB91a].

Many communication protocols can be profitably described in real space process algebra over a finite space. The finite space then represents the collection of port names at which communications occur.

Now we can compare the merits of absolute time notation and relative time notation. We see evidence for the following point of view.

1. In real time process algebra and in finite space real time process algebra the relative time notation is often superior because it leads to simpler process descriptions (e.g., the alternating bit protocol in [BaB91a]) and is particularly useful in verification. The advantage of relative time notation is a modest one however, and absolute time notation will be workable as well.
2. For theoretical work, relative time notation becomes indispensable if concepts concerning finite state and regular processes in real time have to be developed and analysed.

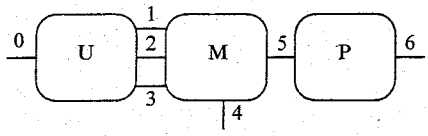

Fig. 10.

3. In the case of (continuous) real space process algebra the absolute time notation becomes more practical. We illustrated this with the example of the moving clock in Section 9.4. There, a spatial replacement operator is needed, which requires an absolutely timed argument process. In the case of relativistic space/time, where space and time cannot be separated (see [BaB91b]), we cannot see how a relative space/time notation can be formulated.

9.9. Example: User Interaction with a Computer and a Printer

Consider a static system consisting of a user U, a machine M and a printer P. We have communication ports 1,2,3,4,5,6 at locations $x_1, x_2, x_3, x_4, x_5, x_6$ (see Fig. 10).

The user will switch on the system, send a sequence of 5 messages, and will then request that this sequence is printed. After this has been completed, he switches off the system. The machine can, after switch on, receive messages which it stores in a buffer. The maximal capacity of this buffer is 100 messages. Upon receiving a print request, the machine reads the time (measured in whole hours). It then causes the printer to print the time, the sequence of messages and the largest value it has received during any session. After successful completion of the print, the machine is ready to switch off.

$$U = \Big(\int_{t \geq 0} s_1(\text{sw\_on})(x_1)[t] \cdot \Big(\sum_{d \in D} s_2(d)(x_2)[\Delta_1] \Big)^5 \cdot s_3(\text{req\_print})(x_3)[\Delta_2] \cdot \int_{t \geq 0} r_3(\text{print\_ok})(x_3)[t] \cdot s_1(\text{sw\_off})(x_1)[\Delta_3] \cdot U$$

$$M = \lambda_{\varnothing}^{\text{store}}(\text{Session}^{\omega})$$

Here, a process X^{ω} is defined by the recursive equation $X^{\omega} = X \cdot X^{\omega}$.)

$$\text{Session} = \text{SW\_ON} \cdot \text{INPUT} \cdot \text{PRINT} \cdot \text{SW\_OFF}$$

$$\text{SW\_ON} = \int_{t \geq 0} r_1(\text{sw\_on})(x_1)[t]$$

$$\text{INPUT} = \int_{t \geq 0} \sum_{d \in D} r_2(d)(x_2)[t] \cdot \text{store}(d)(x_4)[\Delta_4] \cdot \text{INPUT} + \int_{t \geq 0} r_3(\text{req\_print})(x_3)[t]$$

$$\text{PRINT} = \sqrt{}\, t.\text{read\_time}(\lfloor t + \Delta_5 \rfloor \bmod 24)(x_4)(t + \Delta_5) \cdot s_5[\lfloor t + \Delta_5 \rfloor \bmod 24)(x_5)[\Delta_6] \cdot \text{sendprinter}(\text{seq}, \text{max})(x_5)[\Delta_7] \cdot s_5(\text{print})(x_5)[\Delta_8] \cdot \int_{t \geq 0} r_5(\text{printed})(x_5)[t] \cdot s_3(\text{print\_ok})(x_3)[\Delta_9]$$

$$SW\_OFF = \int_{t\geqslant 0} r_1(sw\_off)\,(x_1)\,[t]$$

$$P = \int_{t\geqslant 0} \sum_{n\in\mathbb{Z}_{24}} r_5(n)\,(x_5)\,[t] \cdot \int_{t\geqslant 0} r_5(print)\,(x_5)\,[t]$$
$$\cdot \int_{t\geqslant 0} \sum_{\sigma\in D^*, e\in D} r_5(\sigma, e)\,(x_5)\,[t+\Delta_{10}]$$
$$\cdot print(n, \sigma, e)\,(x_6)\,[\Delta_{11}] \cdot s_5(printed)\,(x_5)\,[\Delta_{12}].$$

Now let

$$I(1,2,3,5) = \{c_i(m) : i\in\{1,2,3,5\}, m\in D \cup D^* \times D \cup \mathbb{Z}_{24} \cup \{sw\_on, req\_print, print\_ok, print, printed, sw\_off\}\}$$

This set contains all successful communications at ports 1,2,3,5. Likewise, $H(1,2,3,5)$ contains all send and receive actions at these ports. Now the system is specified by the equation

$$SYSTEM = \theta_{I(1,2,3,5)} \circ \partial_{H(1,2,3,5)}(U \parallel M \parallel P)$$

The relative time notation makes it clear that there is exactly one place where the absolute time is read and used (rounded off to whole hours on a 24-hour clock), so in particular the user and the printer need have no access to an external clock. This example is formulated in a finite space. The only added value of the use of space coordinates is in the occurrence of multi-actions.

It remains to specify the action and effect functions of the space operator. The set of states is

$$S = \{\sigma\in D^* : length(\sigma) \leqslant 100\} \times D$$

We assume that the set D is totally ordered. In a pair $\langle\sigma, e\rangle \in S$, σ will contain the values in D that have been input during a session and e is the maximum value that has been input since the first session. At the end of each session, σ is reset to the empty sequence $\varnothing$, but e is kept in store. There are two actions interacting with the state operator λ_s^{store}. All other actions are inert.

$$action(store(d)\,(x_4, t), \langle\sigma, e\rangle) = store(d)\,(x_4, t)$$
$$effect(store(d)\,(x_4, t), \langle\sigma, e\rangle) = \langle\sigma d, \max(d, e)\rangle \text{ if } length(\sigma) < 100$$
$$= \langle\sigma, \max(d, e)\rangle \text{ if } length(\sigma) = 100$$
$$action(sendprinter(seq, max)\,(x_5, t), \langle\sigma, e\rangle) = s_5(\sigma, e)\,(x_5, t)$$
$$effect(sendprinter(seq, max)\,(x_5, t), \langle\sigma, e\rangle) = \langle\varnothing, e\rangle$$

10. Conclusion

We conclude that we have introduced real space process algebra, based on Cartesian coordinates. We leave it to future research to investigate applications of variants based on relativistic space/time (as in [BaB91b]). We have obtained a setting in which asynchronous communication can be adequately described. This allows the description of communication between processes moving in space (e.g. communication with a satellite). We have motivated our approach by specifying several small scale examples. Our approach allows very detailed descriptions but introduces a degree of complexity that discourages formal verification by means of equational reasoning. This is not to say that formal verification is impossible, we

just do not see how to do it at this moment. Therefore, it is worthwhile to investigate other techniques. One such technique that is worth investigating is the use of a high level property language (perhaps a variant of temporal logic; see paragraph 3 below). Certainly, it is interesting to look at simulation (or testing) techniques for the analysis of detailed system descriptions (see paragraph 5 below).

As stated in the introduction, there is as yet only very little work on the description of processes operating in real space in a process algebraic framework. What is new in this paper is that we focus on providing tools for the detailed description of the example systems. Of course, one has to realise that we have only described small scale systems, and as such these examples have primarily theoretical value (comparable to the value of a description of the Alternating Bit Protocol in untimed process algebra). A description of practically useful systems may well pose several additional difficulties. An example of a further description that can be undertaken is given in paragraph 4 below.

11. Suggestions for Further Research

We will mention several suggestions for further research.

1. Logical analysis of the real space process algebra: complete axiomatisations.
2. Development of an appropriate property language that allows the expression of system properties at a more abstract level. It seems likely that process algebra itself cannot be used (unlike to the untimed case) and some form of temporal logic will be needed.
3. On the basis of a high level service specification, expressed in an appropriate property language, verification of the protocols given in our paper becomes a research problem. This will be a very complicated matter and we expect that it will be fruitful first to study validation by means of simulation (testing).
4. *Remote sensing*. We want to mention one particular direction of future research in some more detail. As was said in Section 5.12, we think that our setup is not sufficiently expressive to describe remote control and remote sensing, in particular remote sensing. Let us focus on radar as a remote sensing mechanism, the research problem being how to model this in a process algebra setting. We will just outline a possible extension of the process algebra that can be used to describe radar communication. A solid object is described as a subset of $\mathbb{R}^3$. Motion corresponds to a continuous function $f: \mathbb{R}_{\geqslant 0} \to \mathsf{P}(\mathbb{R}^3)$. Solid objects (in motion) are added as time dependent signals, where signals are exactly as in [BaW90]. Then the state operator is equipped with a so-called signal inscription mechanism that allows a state operator to track the signals emitted by objects in its scope. Let the moving object be a ball (its motion depending on system actions). Now the state operator can also arrange for signal reflection on this ball as in Section 5.12. In this way, a radar-like remote sensing system can be modelled. Getting the details of this program worked out is a further non-trivial project.
5. *Simulation*. Experience with protocol description in the untimed case has shown that the availability of a simulation facility is very helpful if not simply necessary in order to get protocol descriptions free of faults to the extent that a subsequent costly verification effort is justified. If we assume that this experience is valid for real time/space process algebra as well, the need for simulation occurs. But

this is not so easy. We expect that simulation requires discretisation. Here, discretisation is some transformation that turns a real time/space process description into a discrete time/space description. Discretisation replaces the real numbers by some finite domain. The research problem here is to develop a discrete time/space process algebra that facilitates discretisation and subsequently, effective simulation. As was said in paragraph 3 above, we expect that simulation and testing must precede work on verification, due to the complexity of the formalism and its underlying subject area.

Acknowledgements

This research was supported in part by ESPRIT basic research action 3006, CONCUR, and ESPRIT basic research action 7166, CONCUR2. The second author also is partially supported by RACE contract 1046, SPECS. This document does not necessarily reflect the views of the SPECS consortium.

We thank Willem Jan Fokkink (CWI Amsterdam) for pointing out that an earlier version of equation TH3 in Section 5.5 was incorrect. We thank Alan Jeffrey (University of Sussex) for stimulating discussions on maximal progress and priority locks and David Murphy (GMD) for a discussion on time stops. We thank the referees for many helpful remarks and suggestions.

References

[BaB88] Baeten, J. C. M. and Bergstra, J. A.: Global Renaming Operators in Concrete Process Algebra, *Information and Computation*, **78**(3), 205–245 (1988).

[BaB91a] Baeten, J. C. M. and Bergstra, J. A.: Real Time Process Algebra. *Formal Aspects. of Computing*, **3**(2), 142–188 (1991).

[BaB91b] Baeten, J. C. M. and Bergstra, J. A.: Real Space Process Algebra. In: J. C. M. Baeten and J. F. Groote (eds), *Proc. CONCUR'91*, Amsterdam. LNCS 527, Springer, pp. 96–110 (1991).

[BaB91c] Baeten, J. C. M. and Bergstra, J. A.: Asynchronous Communication in Real Space Process Algebra. In: J. Vytopil (ed.), *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems, Nijmegen*. LNCS 571, Springer, pp. 473–492 (1991).

[BaB92] Baeten, J. C. M. and Bergstra, J. A.: The State Operator in Real Time Process Algebra. In: J. W. de Bakker, C. Huizing, W. P. de Roever and G. Rozenberg (eds), *Real-Time: Theory in Practice, Proc. REX Workshop Mook 1991*. LNCS 600, Springer, pp. 107–123 (1992).

[BaW90] Baeten, J. C. M. and Weijland, W. P.: Process Algebra. Cambridge Tracts in Theor. Comp. Sci. 18, Cambridge University Press, 1990.

[BBK86] Baeten, J. C. M., Bergstra, J. A. and Klop, J. W.: Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra. *Fundamenta Informaticae*, **IX**(2), pp. 127–168 (1986).

[BeK84] Bergstra, J. A. and Klop, J. W.: Process Algebra for Synchronous Communication. *Information and Control*, **60** 109–137 (1984).

[Ber89] Bergstra, J. A.: A Mode Transfer Operator in Process Algebra. Report P8808b, Programming Research Group, University of Amsterdam (1989).

[Ber90] Bergstra, J. A.: A Process Creation Mechanism in Process Algebra. In: J. C. M. Baeten (ed.), *Applications of Process Algebra*. Cambridge Tracts in Theor. Comp. Sci. 17, Cambridge University Press, pp. 81–88 (1990).

[BKP92] de Boer, F. S., Klop, J. W. and Palamidessi, C.: Asynchronous Communication in Process Algebra. Report CS-R9206, CWI Amsterdam 1992. (To appear in Proc. LICS 92, Santa Cruz.)

[BKT85] Bergstra, J. A., Klop, J. W. and Tucker, J. V.: Process Algebra with Asynchronous

Communication Mechanisms. In: S. D. Brookes, A. W. Roscoe and G. Winskel (eds), *Proc. Seminar on Concurrency*. LNCS 197, Springer, pp. 76–95 (1985).

[FoK92] Fokkink, W. J. and Klusener, A. S.: Real Time Process Algebra with Prefixed Integration. Report CS-R9219, CWI Amsterdam 1992.

[GlV89] van Glabbeek, R. J. and Vaandrager, F. W.: Modular Specifications in Process Algebra (with Curious Queues). In: M. Wirsing and J. A. Bergstra (eds), *Algebraic Methods: Theory, Tools and Applications*, Passau 1987. LNCS 394, Springer, pp. 465–506 (1989).

[Jef91a] Jeffrey, A.: Observation Spaces and Timed Processes. In: J. C. M. Baeten and J. F. Groote (eds), *Proc. CONCUR'91*, Amsterdam. LNCS 527, Springer, pp. 332–345 (1991).

[Jef91b] Jeffrey, A.: Translating Timed Process Algebra into Prioritized Process Algebra. In: J. Vytopil (ed.), *Proc. Formal Techniques in Real-Time and Fault-Tolerant Systems*, Nijmegen 1992. LNCS 571, Springer, pp. 493–506 (1991).

[Klu91] Klusener, A. S.: Completeness in Real Time Process Algebra, Report CS-R9106, CWI Amsterdam 1991. (Extended abstract in: J. C. M. Baeten and J. F. Groote (eds), *Proc. CONCUR'91*, Amsterdam). LNCS 527, Springer, pp. 376–392 (1991).

[KoM90] Koymans, C. P. J. and Mulder, J. C.: A Modular Approach to Protocol Verification Using Process Algebra. In: J. C. M. Baeten (ed.), *Applications of Process Algebra*. Cambridge Tracts in Theor. Comp. Sci. 17, Cambridge University Press, pp. 261–306 (1990).

[LaM87] Larsen, K. G. and Milner, R.: A Complete Protocol Verification Using Relativized Bisilulation. In: Th. Ottmann (ed.), *Proc. 14th ICALP*, Karlsruhe. LNCS 267, Springer, pp. 126–135 (1987).

[MoT90] Moller, F. and Tofts, C.: A Temporal Calculus of Communicating Systems. In: J. C. M. Baeten and J. W. Klop (eds), *Proc. CONCUR'90*, Amsterdam. LNCS 458, Springer, pp. 401–415 (1990).

[Mur91] Murphy, D. V. J.: Testing, Betting and Timed True Concurrency. In: J. C. M. Baeten and J. F. Groote (eds), *Proc. CONCUR'91*, Amsterdam. LNCS 527, Springer, pp. 439–454 (1991).

[NiS91] Nicollin, X. and Sifakis, J.: The Algebra of Timed Processes ATP: Theory and Application (revised version). Report RT-C26, IMAG Grenoble, 1991.

[Plo81] Plotkin, G. D.: A Structural Approach to Operational Semantics. Report DAIMI FN-19, Comp. Sci. Dept, Aarhus University, 1981.

[ReR88] Reed, G. M. and Roscoe, A. W.: A Timed Model for Communicating Sequential Processes. Theor. Comp. Sci., **58**, 249–261 (1988).

[Vaa90a] Vaandrager, F. W.: Two Simple Protocols. In: J. C. M. Baeten (ed.), *Applications of Process Algebra*. Cambridge Tracts in Theor. Comp. Sci. 17, Cambridge University Press, pp. 23–44 (1990).

[Vaa90b] Vaandrager, F. W.: Process Algebra Semantics of POOL. In: J. C. M. Baeten (ed.), *Applications of Process Algebra*. Cambridge Tracts in Theor. Comp. Sci. 17, Cambridge University Press, 173–236 (1990).

Received October 1990
Accepted in revised form September 1992 by Zhou Chaochen