

# Timed Process Algebra (with a Focus on Explicit Termination and Relative-Timing)

J.C.M. Baeten and M.A. Reniers

Department of Mathematics and Computing Science,  
Eindhoven University of Technology,  
{J.C.M.Baeten,M.A.Reniers}@tue.nl  
June 23, 2004

**Abstract.** We treat theory and application of timed process algebra. We focus on a variant that uses explicit termination and action prefixing. This variant has some advantages over other variants. We concentrate on relative timing, but the treatment of absolute timing is similar. We treat both discrete and dense timing. We build up the theory incrementally. The different algebras are interrelated by embeddings and conservative extensions. As an example, we consider the PAR communication protocol.

## 1 Introduction

Process algebras that incorporate some form of timing, enabling quantitative analysis of time performance, have been studied extensively by now (see e.g. [1–5]), and successfully applied in a number of case studies (see e.g. [6, 7]).

For different purposes and applications different versions of describing timing have been presented in the literature. The following fundamental choices have to be made:

1. The nature of the time domain.
  - (a) Discrete time versus dense time. This choice is with respect to which type of time domain, timing is described. Usually, two types of time domains are distinguished. These are *discrete time*, where the time domain is of a discrete nature, and *dense time*, where the time domain is of a continuous nature, i.e. between every two moments in time there is another moment in time.
  - (b) Linear time versus branching time. In a linear time domain each two moments in time are ordered by some total ordering  $\leq$ , in a branching time domain this is only a partial ordering.
  - (c) Least element available or not.

For example in timed  $\mu$ CRL [8], the user of the language can specify his own time domain by means of algebraic specifications as long as it has a least element and a total order. In this process algebra both discrete and dense time domains can be used.

In the present work, we consider both the natural numbers  $N$  and the non-negative reals  $R^{\geq 0}$  as the time domain.

2. The way time is described syntactically.
  - (a) Time-stamped description versus two-phase description. If the description of time is attached to the atomic actions, we speak of a time-stamping mechanism. If, on the other hand, time delay is decoupled from action execution, we have a two-phase approach.
 

In the present work, we start out from a two-phase approach. This is the syntax the overwhelming majority of literature uses, both when giving axioms and when giving operational semantics. We stick to this approach in the axiom systems, but surprisingly, we find that giving operational rules for the dense-time case works better if in the transition steps, actions and time, and termination and time, are combined.
3. The way time is incorporated semantically.
  - (a) Absolute timing versus relative timing. Sometimes, it is convenient to describe the passage of time with respect to a global clock, sometimes, it is convenient to describe passage of time relative to the previous action. The first is called *absolute timing*, the second *relative timing*. The combination of these two notions is called *parametric timing* [9, 10]. Here, we focus on relative-timing, but the theory with absolute timing is similar.
  - (b) Time-determinism versus time-nondeterminism. A choice to be made with far reaching consequences is whether a delay may determine a choice. In the literature three versions are encountered. Firstly, if the delay by itself may not determine any choice we speak of *strong time-determinism*. Secondly, if a delay of  $t$  time by itself cannot determine a choice between alternatives that allow a delay of  $t$  time we speak of *weak time-determinism*. Finally, if a delay can determine a choice we speak of *time-nondeterminism*. Strong time-determinism is found in ATP [4], weak time-determinism in many ACP-like timed process algebras (also in the present work), and time-nondeterminism in  $ACP_{\tau\epsilon}^t$  [11]. In the literature, time-determinism is also referred to as time-factorisation.
  - (c) Duration of actions. One can either assume that actions have no duration, and hence are instantaneous, or that they do have a duration. In the latter case, the duration can be specified or unspecified. Here, we assume actions have no duration, and so we explicitly have to model delays between actions.
  - (d) Urgent actions versus multi-actions. If actions occurring at the same time can be ordered these are called urgent actions, otherwise these are called multi-actions. Here, we use urgent actions.

In [12], several process algebras are treated in a common framework, and related by embeddings and conservative extension relations. These process algebras,  $ACP^{\text{sat}}$ ,  $ACP^{\text{srt}}$ ,  $ACP^{\text{dat}}$  and  $ACP^{\text{drt}}$ , allow the execution of two or more actions consecutively at the same point in time, separate the execution of actions from the passage of time, adhere to the principle of weak time-determinism, and consider actions to have no duration. The process algebra  $ACP^{\text{sat}}$  is a real-time process algebra with absolute time,  $ACP^{\text{srt}}$  is a real-time process algebra with relative time. Similarly,  $ACP^{\text{dat}}$  and  $ACP^{\text{drt}}$  are discrete-time process algebras

with absolute time and relative time respectively. In these process algebras, considerable attention was given to the inaction constant  $\delta$ , standing for unsuccessful termination or deadlock and the different roles it plays in the untimed theory.

In this paper, we extend the framework of [12] with a constant representing successful termination for the relative-time case. We consider a linear-time process algebra with urgent actions which have no duration. Furthermore, we will adopt the principle of weak time-determinism. It is not necessary to include the deadlocked process  $\delta$  as in [12]. The discrete-time part is taken from [13], the dense-time part is from [14].

Explicit termination is denoted by the constant  $\epsilon$ , called *empty process*. This process  $\epsilon$  denoting successful termination or *skip* has not been studied nearly as well as the unsuccessful termination constant  $\delta$ . The untimed theory was investigated in [15–17]. In the context of ACP-like process algebras the empty process in a timed setting is mentioned in [11, 18, 19]. In [18, 19] a relative-time, discrete-time process algebra has been extended with both a non-delayable and a delayable successful termination constant.

As is the case for deadlock in the untimed theory, also the empty process has more roles. On the one hand, it serves as the neutral element for sequential composition (and parallel composition), on the other hand, it stands for the process that executes no actions but terminates at some unspecified time.

Thus, we have extended the integrated framework of process algebras with timing from [12] with a relative-time process algebra with the empty process. We show that the various subtheories are still related by means of embeddings and conservative extensions. This extension with the empty process, in itself needed for a clear understanding of termination in timed process algebra, is also needed in order to give semantics for programming languages and specification languages that involve skip: examples are CSP (see [20, 7]),  $\chi$  (see [21]) and MSC (see [22]).

This article is structured as follows. In Section 2, we review the basic theory of relative-time process algebra with explicit termination, first discrete-time, and then dense-time. In Section 3, we extend with sequential composition, in Section 4 with parallel composition. In Section 5, we discuss various embeddings of different time-free process algebras. In Section 6, we discuss a couple of extra operators and notations that we will use in the examples to come. As examples, we use buffers with various timing behaviours in Section 7, and the PAR communication protocol in Section 8. We conclude in Section 9 with a short description of related work.

*Acknowledgments* We like to thank Sjouke Mauw, Kees Middelburg and Tim Willemsse for their various useful suggestions.

## 2 Minimal Process Algebra for Relative Timing

### 2.1 Minimal Process Algebra: $\text{MPA}_{drt}$

In this section we give a short overview of the process algebra  $\text{MPA}_{drt}$  from [13] as it is the process algebra that is closest to the process algebra that we introduce. It is a process algebra that uses relative timing and discrete time. Further, it is two-phase and has weak time-determinism.

The process algebra  $\text{MPA}_{drt}$  is parametrised by a set  $A$  of actions. The signature of  $\text{MPA}_{drt}$  contains the following constants:

- *undelayable deadlock*  $\underline{\delta}$ . This is the process that cannot perform any action, cannot terminate and cannot let time progress beyond the current time slice (it cannot 'tick'). Operationally, it is characterized by having no operational rules at all. Undelayable deadlock is the neutral element of alternative composition (discussed below).
- *undelayable termination*  $\underline{\epsilon}$ . This is the process that cannot perform any action and cannot let time progress beyond the current time slice (cannot 'tick'). It can only terminate successfully in the current time slice. Undelayable termination is the neutral element of sequential composition (to be discussed in a later section).

More complex processes are constructed using the following operators:

- for each action  $a \in A$ , the *current time slice action prefix* operator  $\underline{a}..$ . The process  $\underline{a}.x$  executes action  $a$  in the current time slice (it cannot progress to the next time slice) and next starts the execution of  $x$  in the current time slice.
- the *next time slice prefix* operator  $\sigma..$ . The process  $\sigma.x$  can 'tick' (can let time progress to the next time slice), and can then continue with  $x$  (in the next time slice). Since we have explicit termination, we can consider this operator as a prefix operator. Often, it is written  $\sigma_{rel}$  to emphasize the fact that we have relative timing. As we will not consider absolute timing in this work, it is not necessary to write the subscript here.
- alternative composition  $+$ . The process  $x + y$  executes either  $x$  or  $y$ , but not both. The choice is resolved upon execution of the first action.

We have omitted from the signature of  $\text{MPA}_{drt}$  the delayable counterparts of  $\underline{\epsilon}$ ,  $\underline{\delta}$  and  $\underline{a}..$ . We consider such extensions later in Section 5 when embedding untimed process algebras into the timed process algebras introduced in this paper.

The axioms of  $\text{MPA}_{drt}$  are presented in Table 1. Axioms A1-A3 explain that alternative composition is commutative, associative and idempotent. Axiom A6DR states that undelayable deadlock is the neutral element of alternative composition. Finally, Axiom DRTD (Discrete Relative Time Determinism) expresses weak time-determinism: passage of time cannot introduce non-determinism, so passage of time leads to a unique remaining process. This indicates that passage of time is different from action execution, which can introduce non-determinism (upon execution of an action, different processes can result).

**Table 1.** Axioms of  $\text{MPA}_{drt}$ 

|  |      |
|--|------|
| $x + y = y + x$                        | A1   |
| $(x + y) + z = x + (y + z)$            | A2   |
| $x + x = x$                            | A3   |
| $x + \underline{\delta} = x$           | A6DR |
| $\sigma.x + \sigma.y = \sigma.(x + y)$ | DRTD |

In [13], by means of the operational rules of Table 2, an operational semantics is given for closed  $\text{MPA}_{drt}$ -terms defining binary relations  $- \xrightarrow{a} -$  (for  $a \in A$ ) and  $- \xrightarrow{1} -$ , and a unary relation (predicate)  $- \downarrow$ . Intuitively, these have the following meaning:

- $x \xrightarrow{a} x'$  means that  $x$  evolves into  $x'$  by executing atomic action  $a$  (in the current time slice);
- $x \xrightarrow{1} x'$  means that  $x$  evolves into  $x'$  by passing to the next time slice;
- $x \downarrow$  means that  $x$  has an option to terminate successfully (in the current time slice).

Deduction rules (8 – 10) clearly state that in an alternative composition a choice is made by passing to the next time slice only in case the other alternative is not able to proceed to this time slice. In case both alternatives allow a passage to the next time slice, no choice is made (yet).

**Table 2.** Deduction rules for  $\text{MPA}_{drt}$  ( $a \in A$ )

|   |   |  |   |
|---|---|--|---|
| $\underline{\epsilon} \downarrow$ (1)   | $\underline{a}.x \xrightarrow{a} x$ (2)   | $\sigma.x \xrightarrow{1} x$ (3)   |   |
| $\frac{x \downarrow}{x + y \downarrow}$ (4)   | $\frac{y \downarrow}{x + y \downarrow}$ (5)   | $\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'}$ (6)                              | $\frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$ (7) |
| $\frac{x \xrightarrow{1} x' \quad y \xrightarrow{1} y'}{x + y \xrightarrow{1} x' + y'}$ (8) | $\frac{x \xrightarrow{1} x' \quad y \not\xrightarrow{1}}{x + y \xrightarrow{1} x'}$ (9) | $\frac{x \not\xrightarrow{1} \quad y \xrightarrow{1} y'}{x + y \xrightarrow{1} y'}$ (10) |   |

Observe that for enforcing weak time-determinism, in the deduction rules for time transitions of process terms in the form of a sum, negative premises are used. For deduction systems in which negative premises are used, it is not obvious which set of positive formulas can be deduced. If a stratification<sup>1</sup> can

<sup>1</sup> See [23] for a definition of stratification.

be provided for the deduction system, it is well-defined in the sense that a set of positive formulas is defined by it. In this case, it is not difficult to come up with a stratification.

The axioms introduced before are meant to identify processes that are strongly bisimilar.

**Definition 1 (Strong bisimilarity).** *A symmetric, binary relation  $R$  on processes is called a strong bisimulation relation if for all process terms  $p, q$  such that  $(p, q) \in R$  we have*

- if  $p \downarrow$  then  $q \downarrow$ ;
- for all  $a \in A$  and process terms  $p'$ : if  $p \xrightarrow{a} p'$ , then there exists a process term  $q'$  such that  $q \xrightarrow{a} q'$  and  $(p', q') \in R$ ;
- for all process terms  $p'$ : if  $p \xrightarrow{1} p'$ , then there exists a process term  $q'$  such that  $q \xrightarrow{1} q'$  and  $(p', q') \in R$ .

*Two processes  $p$  and  $q$  are strongly bisimilar, notation  $p \simeq q$ , if there exists a strong bisimulation relation  $R$  such that  $(p, q) \in R$ . If a relation  $R$  is given that witnesses the strong bisimilarity of processes  $p$  and  $q$ , then we write  $R : p \simeq q$ .*

The notion of strong bisimilarity on closed  $\text{MPA}_{drt}$ -terms is both an equivalence and a congruence for all the operators of the process algebra  $\text{MPA}_{drt}$ . As we have used the standard definition of strong bisimilarity, equivalence is for free (follows from the format of the deduction rules).

**Theorem 1 (Equivalence).** *Strong bisimilarity is an equivalence relation.*

**Theorem 2 (Congruence).** *Strong bisimilarity is a congruence for the operators of the process algebra  $\text{MPA}_{drt}$ .*

*Proof.* The deduction system is stratifiable and in panth format and hence strong bisimilarity is a congruence [24, 25].

We establish that the structure of transition systems modulo strong bisimilarity is a model for our axioms, or, put differently, that our axioms are sound with respect to the set of closed terms modulo strong bisimilarity. We also state that the axiomatisation is complete, a proof of this fact is outside the scope of this survey.

**Theorem 3 (Soundness).** *The process algebra  $\text{MPA}_{drt}$  is a sound axiomatisation of strong bisimilarity on closed  $\text{MPA}_{drt}$ -terms.*

**Theorem 4 (Completeness).** *The process algebra  $\text{MPA}_{drt}$  is a complete axiomatisation of strong bisimilarity on closed  $\text{MPA}_{drt}$ -terms.*

The next time slice prefix  $\sigma$  allows the passage of one unit of time. It can be generalized to an operator  $\sigma^n$  for  $n \in \mathbb{N}$ , that allows the passage of  $n$  units of time. This relative delay prefix operator  $\sigma^n$  can be added to  $\text{MPA}_{drt}$  by

**Table 3.** Axioms of relative delay prefix ( $n \in N$ )

|                                      |       |
|--------------------------------------|-------|
| $\sigma^0.x = x$                     | DRDP1 |
| $\sigma^{n+1}.x = \sigma.\sigma^n.x$ | DRDP2 |

considering the axioms given in Table 3. In specifications, this operator is very useful. In fact, in [12], it is taken as a basic operator.

The emphasis of the present work is to present variants of this discrete time theory and its extensions to dense time theories. It will turn out that discrete time and dense time theories have very much in common, the axioms look more or the less the same. In order to bring out these similarities sharply, we present the axiomatization of this basic discrete time theory based on the  $\sigma^n$  prefix rather than the  $\sigma$  prefix.

In the next section, we introduce such a process algebra, called  $MPT_{drt}$ .

## 2.2 Relative delay prefix operator

The axioms of  $MPT_{drt}$  are given in Table 4. Axiom DRT1 states that a delay of zero time units has no effect on the following process. Axiom DRT2 captures the relative-time nature of the relative delay prefix operators: two consecutive delays can be added to obtain a single delay. Axiom DRT3 is a generalization of axiom DRTD from  $MPA_{drt}$  to capture weak time-determinism for any delay, not just for one time unit. The resulting theory is called  $MPT_{drt}$ .

**Table 4.** Axioms of  $MPT_{drt}$  ( $a \in A, n, m \in N$ )

|                              |      |  |      |
|------------------------------|------|--|------|
| $x + y = y + x$              | A1   | $\sigma^0.x = x$                             | DRT1 |
| $(x + y) + z = x + (y + z)$  | A2   | $\sigma^n.(\sigma^m.x) = \sigma^{n+m}.x$     | DRT2 |
| $x + x = x$                  | A3   | $\sigma^n.x + \sigma^n.y = \sigma^n.(x + y)$ | DRT3 |
| $x + \underline{\delta} = x$ | A6DR |  |      |

The theory  $MPA_{drt}$  can be embedded into  $MPT_{drt}$  by interpreting  $\sigma_.$  as  $\sigma^1_.$ . Observe that Axiom DRTD is obtained (modulo notations) by instantiating axiom DRT3 with  $n = 1$ . This embedding is discussed in more detail in Section 5.4.

An operational semantics of  $MPT_{drt}$  can be found relatively easily by using the same relations and predicate  $\_ \xrightarrow{a} \_$  and  $\_ \xrightarrow{1} \_$  and  $\_ \downarrow$ . Alternatively, we can use relations  $\_ \xrightarrow{n} \_$  (for all  $n \in N$ ) instead of  $\_ \xrightarrow{1} \_$ . We give neither approach

here, as they cannot be generalized to dense time theories, for reasons to be explained later.

Now for this discrete time theory  $\text{MPT}_{\text{drt}}$ , it is very easy to obtain a dense time counterpart  $\text{MPT}_{\text{srt}}$ . All that needs to be done is to change the time slice syntax to time point syntax, and change the range of the delay superscripts from  $N$  to  $R^{\geq 0}$ . We spell this out in the following.

The process algebra  $\text{MPT}_{\text{srt}}$  is a minimal process theory with real-time relative timing. It is parametrised by a set  $A$  of actions. The signature of  $\text{MPT}_{\text{srt}}$  contains the following constants:

- *undelayable deadlock*  $\tilde{\delta}$ . This is the process that cannot execute any action, cannot terminate, and cannot let time progress beyond the current point of time.
- *undelayable termination*  $\tilde{\epsilon}$ . The process that cannot execute any action, cannot let time progress beyond the current point of time, but can terminate at the current point of time.

More complex processes are constructed using the following operators:

- for each action  $a \in A$ , the *urgent action prefix* operator  $\tilde{a} \dots$ . The process  $\tilde{a} \dots x$  can only execute  $a$  at the current point in time and then continue with  $x$  at the current point of time.
- for  $t \in R^{\geq 0}$ , the *relative delay prefix* operator  $\sigma^t \dots$ . The process  $\sigma^t \dots x$  can only delay for an amount of time  $t$  and then continues with  $x$  (at time  $t$  after the current point of time);
- alternative composition  $+$ . The process  $x + y$  executes either  $x$  or  $y$ , but not both. The choice is resolved upon execution of the first action.

Observe that the signatures and the axioms of  $\text{MPT}_{\text{drt}}$  and  $\text{MPT}_{\text{srt}}$  only differ in their notations for the constants and the action prefix operators.

The axioms of  $\text{MPT}_{\text{srt}}$  are presented in Table 5. It turns out that these are the same as the axioms of  $\text{MPT}_{\text{drt}}$  except for the notations as mentioned before.

**Table 5.** Axioms of  $\text{MPT}_{\text{srt}}$  ( $t, s \in R^{\geq 0}$ )

|                             |      |  |      |
|-----------------------------|------|--|------|
| $x + y = y + x$             | A1   | $\sigma^0 \dots x = x$   | SRT1 |
| $(x + y) + z = x + (y + z)$ | A2   | $\sigma^t \dots (\sigma^s \dots x) = \sigma^{t+s} \dots x$     | SRT2 |
| $x + x = x$                 | A3   | $\sigma^t \dots x + \sigma^t \dots y = \sigma^t \dots (x + y)$ | SRT3 |
| $x + \tilde{\delta} = x$    | A6SR |  |      |

For both  $\text{MPT}_{\text{drt}}$  and  $\text{MPT}_{\text{srt}}$  an operational semantics can easily be obtained by generalizing the time transition relation  $\xrightarrow{1}$  of  $\text{MPA}_{\text{drt}}$  to the time

transitions  $\_ \xrightarrow{t} \_$  (for  $t \in N^{>0}$  and  $t \in R^{>0}$  respectively). The deduction rules are then simple reformulations of the deduction rules of  $\text{MPA}_{drt}$  from Table 2.

Given these similarities between a discrete-time and a real-time process algebra with relative timing, we only consider process algebras with real-time relative timing in the remainder of this paper. The corresponding discrete-time variants can be derived easily by changing the range of the delay parameter and changing time slice constants and prefixes to time point constants and prefixes.

In the next section we extend  $\text{MPT}_{\text{srt}}$  with an operator for sequential composition. Due to the presence of a constant for explicit termination and our desire to have weak time-determinism, the formulation of deduction rules for time transitions of sequential composition turn out to be difficult (see [26] and the next section). For this reason we present a different operational semantics for  $\text{MPT}_{\text{srt}}$ . Instead of the two-phase operational approach used in the previous section, we now integrate passage of time with action transitions and termination in the operational semantics. We define binary relations  $\_ \xrightarrow{a}_t \_$  and unary relations  $\_ \downarrow_t$  and  $\Delta_t(\_)$  on closed terms (for  $a \in A$  and  $t \in R^{\geq 0}$ ). Intuitively, they have the following meaning:

1.  $x \xrightarrow{a}_t x'$  means that  $x$  evolves into  $x'$  at time  $t$  after the current point of time upon the execution of the atomic action  $a$  at that time;
2.  $x \downarrow_t$  means that  $x$  has an option to terminate successfully at time  $t$  after the current point of time.
3.  $\Delta_t(x)$  means that  $x$  has the possibility of delaying for  $t$  time.

The predicate  $\downarrow_0$  discriminates between  $\tilde{\delta}$  and  $\tilde{\epsilon}$ . The predicate  $\Delta_{\max(t,t')}$  discriminates between  $\sigma^t(\tilde{\delta})$  and  $\sigma^{t'}(\tilde{\delta})$  for different  $t$  and  $t'$ . The deduction rules for  $\text{MPT}_{\text{srt}}$  are given in Table 6. Note that in these deduction rules no negative premises appear. This is due to the decision to integrate time passage with action transitions and termination. Note that the predicate  $\Delta_0$  holds for all processes. This predicate does not hold for the process  $\dot{\delta}$  of [12] that is not considered here.

**Table 6.** Deduction rules for  $\text{MPT}_{\text{srt}}$  ( $a \in A$ ,  $t, s \in R^{\geq 0}$ )

|  |  |  |  |
|--|--|--|--|
| $\tilde{\epsilon} \downarrow_0$ (1)                      | $\Delta_0(x)$ (2)  | $\tilde{a}.x \xrightarrow{a}_0 x$ (3)                          |  |
| $\frac{x \downarrow_t}{\sigma^s.x \downarrow_{t+s}}$ (4) | $\frac{x \xrightarrow{a}_t x'}{\sigma^s.x \xrightarrow{a}_{t+s} x'}$ (5) | $\frac{t \leq s}{\Delta_t(\sigma^s.x)}$ (6)                    | $\frac{\Delta_t(x)}{\Delta_{t+s}(\sigma^s.x)}$ (7)             |
| $\frac{x \downarrow_t}{x+y \downarrow_t}$ (8)            | $\frac{y \downarrow_t}{x+y \downarrow_t}$ (9)                            | $\frac{x \xrightarrow{a}_t x'}{x+y \xrightarrow{a}_t x'}$ (10) | $\frac{y \xrightarrow{a}_t y'}{x+y \xrightarrow{a}_t y'}$ (11) |
|  | $\frac{\Delta_t(x)}{\Delta_t(x+y)}$ (12)                                 | $\frac{\Delta_t(y)}{\Delta_t(x+y)}$ (13)                       |  |

Since we have replaced the relations and predicates that are used in the operational semantics, the notion of strong bisimilarity must be adapted accordingly.

**Definition 2 (Strong bisimilarity).** *A symmetric, binary relation  $R$  on process terms is called a strong bisimulation relation if for all process terms  $p, q$  such that  $(p, q) \in R$  we have*

- for all  $t \in R^{\geq 0}$ : if  $p \downarrow_t$  then  $q \downarrow_t$ ;
- for all  $a \in A$ ,  $t \in R^{\geq 0}$ , and process terms  $p'$ : if  $p \xrightarrow{a}_t p'$ , then there exists process term  $q'$  such that  $q \xrightarrow{a}_t q'$  and  $(p', q') \in R$ ;
- for all  $t \in R^{\geq 0}$ : if  $\Delta_t(p)$  then  $\Delta_t(q)$ .

Two processes  $p$  and  $q$  are strongly bisimilar, notation  $p \equiv q$ , if there exists a strong bisimulation relation  $R$  such that  $(p, q) \in R$ . If a relation  $R$  is given that witnesses the strong bisimilarity of processes  $p$  and  $q$ , then we write  $R : p \equiv q$ .

The notion of strong bisimilarity on closed  $\text{MPT}_{\text{srt}}$ -terms is both an equivalence and a congruence for all the operators of the process algebra  $\text{MPT}_{\text{srt}}$ . As we have used the standard definition of strong bisimilarity, equivalence is for free.

**Theorem 5 (Equivalence).** *Strong bisimilarity is an equivalence relation.*

**Theorem 6 (Congruence).** *Strong bisimilarity is a congruence for the operators of the process algebra  $\text{MPT}_{\text{srt}}$ .*

*Proof.* The deduction system is in path format and hence strong bisimilarity is a congruence [27, 25].

We establish that the structure of transition systems modulo strong bisimilarity is a model for our axioms, or, put differently, that our axioms are sound with respect to the set of closed terms modulo strong bisimilarity. We also state that the axiomatisation is complete. The proof of the latter fact (not given here) relies on the elimination theorem stated first.

**Theorem 7 (Soundness).** *The process algebra  $\text{MPT}_{\text{srt}}$  is a sound axiomatisation of strong bisimilarity on closed  $\text{MPT}_{\text{srt}}$ -terms.*

We define a notion of *basic terms*. These are useful in the proofs to come. It turns out that every closed  $\text{MPT}_{\text{srt}}$ -term is derivably equal to a basic term. This provides us with an easier to use means of induction: instead of proving properties by induction on the structure of closed terms, it suffices to prove these properties by induction on the structure of basic terms. The proof that the axiomatization is complete with respect to the set of closed terms modulo strong bisimilarity uses this induction principle.

**Definition 3 (Basic terms).** *The set of all basic terms is the smallest set  $\mathcal{B}$  that satisfies*

1. for  $t \in R^{\geq 0}$ :  $\sigma^t. \tilde{\epsilon} \in \mathcal{B}$ ;

2. for  $t \in R^{\geq 0}$ :  $\sigma^t.\tilde{\delta} \in \mathcal{B}$ ;
3. for  $a \in A$ ,  $t \in R^{\geq 0}$  and  $x \in \mathcal{B}$ :  $\sigma^t.\tilde{a}.x \in \mathcal{B}$ ;
4. for  $x, y \in \mathcal{B}$ :  $x + y \in \mathcal{B}$ .

**Theorem 8 (Elimination).** *Every closed  $MPT_{\text{srt}}$ -term  $p$  is derivably equal to a basic term  $q \in \mathcal{B}$ , i.e. for all process term  $p$  there is a basic term  $q$  with  $MPT_{\text{srt}} \vdash p = q$ .*

*Proof.* By induction on the structure of closed  $MPT_{\text{srt}}$ -term  $p$ .

**Theorem 9 (Completeness).** *The process algebra  $MPT_{\text{srt}}$  is a complete axiomatization of strong bisimilarity on closed  $MPT_{\text{srt}}$ -terms.*

### 3 Sequential Composition

In this section we extend the process algebra  $MPT_{\text{srt}}$  from the previous section with sequential composition. The sequential composition of two processes  $p$  and  $q$  is denoted  $p \cdot q$ . The process  $p \cdot q$  starts with the execution of  $p$ , and upon termination of  $p$  continues with the execution of  $q$ . Sequential composition binds stronger than alternative composition and weaker than action prefix. The axioms of  $TSP_{\text{srt}}$  are the axioms of  $MPT_{\text{srt}}$ , and in addition the axioms presented in Table 7.

**Table 7.** Axioms of  $TSP_{\text{srt}}$  ( $a \in A$ ,  $t \in R^{\geq 0}$ )

|   |       |   |       |
|---|-------|---|-------|
| $(x + y) \cdot z = x \cdot z + y \cdot z$                 | A4    | $\tilde{\delta} \cdot x = \tilde{\delta}$ | A7SR  |
| $(x \cdot y) \cdot z = x \cdot (y \cdot z)$               | A5    | $\tilde{\epsilon} \cdot x = x$            | A8SDR |
| $\tilde{a} \cdot x \cdot y = \tilde{a} \cdot (x \cdot y)$ | A10SR | $x \cdot \tilde{\epsilon} = x$            | A9SR  |
| $\sigma^t \cdot x \cdot y = \sigma^t \cdot (x \cdot y)$   | SRSEQ |   |       |

The axioms A4 and A5 describe straightforward properties of alternative and sequential composition: sequential composition distributes over alternative composition from the right, and sequential composition is associative. Notice that the other distributive law does not hold, as the moment of choice in  $x \cdot (y + z)$  is after the execution of  $x$ , whereas in  $x \cdot y + x \cdot z$  it is before the execution of  $x$ . The axiom A7SR explains that undelayable deadlock is a left-zero element for sequential composition. As  $\tilde{\delta}$  cannot terminate, nothing of a following process can occur. The axioms A8SR-A9SR explain that undelayable termination is a neutral element for sequential composition. This is because  $\tilde{\epsilon}$  terminates immediately and successfully, so a following process can be started at the same point in time, and possible termination of a previous process is not affected. The axioms A10SR and SRSEQ describe the interplay between the

action prefix operators and sequential composition and between the delay prefix operators and sequential composition, respectively.

Compared to [12], we can present the delay operators as prefix operators rather than a general unary operator, and we have no need of a special deadlocked process constant  $\delta$ . This simplifies the set of axioms and operational rules.

We present the deduction rules for sequential composition in Table 8. The operational semantics of  $\text{TSP}_{\text{srt}}$  consists of the deduction rules of  $\text{MPT}_{\text{srt}}$  and in addition the deduction rules for sequential composition from Table 8.

**Table 8.** Deduction rules for sequential composition ( $a \in A, t, s \in T$ )

|   |   |   |
|---|---|---|
| $\frac{x \downarrow_t, y \downarrow_s}{x \cdot y \downarrow_{t+s}} (1)$ | $\frac{x \xrightarrow{a}_t x'}{x \cdot y \xrightarrow{a}_t x' \cdot y} (2)$ | $\frac{x \downarrow_t, y \xrightarrow{a}_s y'}{x \cdot y \xrightarrow{a}_{t+s} y'} (3)$ |
| $\frac{\Delta_t(x)}{\Delta_t(x \cdot y)} (4)$                           | $\frac{x \downarrow_t \quad \Delta_s(y)}{\Delta_{t+s}(x \cdot y)} (5)$      |   |

*Intermezzo: One-phase versus two-phase operational semantics.* At this point, it is possible to further explain our switch from the two-phase approach in defining operational semantics for  $\text{MPA}_{\text{drt}}$  with (generalized) time transitions  $\_ \xrightarrow{1} \_$  (or  $\_ \xrightarrow{n} \_$ ) to the current setting. For discrete time theories, in the two-phase approach using the single time-unit time transitions gives the following deduction rules for sequential composition (see [13], note that the second deduction rule is missing in [13]):

$$\frac{x \xrightarrow{1} x' \quad x \not\downarrow}{x \cdot y \xrightarrow{1} x' \cdot y} \quad \frac{x \xrightarrow{1} x' \quad y \not\downarrow}{x \cdot y \xrightarrow{1} x' \cdot y} \quad \frac{x \not\downarrow \quad x \downarrow \quad y \xrightarrow{1} y'}{x \cdot y \xrightarrow{1} y'} \quad \frac{x \xrightarrow{1} x' \quad x \downarrow \quad y \xrightarrow{1} y'}{x \cdot y \xrightarrow{1} x' \cdot y + y'}$$

Observe that the premises of these deduction rules exclude each other in cases where the result of the conclusion differs. Consequently, time-determinism is preserved by these deduction rules. In a dense setting, where there is no smallest unit of time, this approach can not be used at all.

Using generalized time transitions in a discrete or dense setting is already impossible without introducing additional syntax as in order to consider a possible delay of  $x \cdot y$ , it is required to consider all combinations: the delay can come completely from  $x$ , it can come partly from  $x$  followed by termination of  $x$  and the rest of the delay from  $y$ , and in case  $x$  can terminate immediately, the delay can come completely from  $y$ . In [26], saturation of the transition systems was proposed as a solution, thereby allowing the transition systems to be time-nondeterministic.

The notion of strong bisimilarity on closed  $TSP_{\text{srt}}$ -terms is both an equivalence and a congruence for all the operators of the process algebra  $TSP_{\text{srt}}$ . As we have used the standard definition of strong bisimilarity, equivalence is for free.

**Theorem 10 (Equivalence).** *Strong bisimilarity is an equivalence relation.*

**Theorem 11 (Congruence).** *Strong bisimilarity is a congruence for the operators of the process algebra  $TSP_{\text{srt}}$ .*

*Proof.* The deduction system is in path format and hence strong bisimilarity is a congruence [27, 25].

Next, we establish that the structure of transition systems modulo strong bisimilarity is a model for our axioms, or, put differently, that our axioms are sound with respect to the set of closed terms modulo strong bisimilarity. We also prove that the axiomatisation is complete.

**Theorem 12 (Soundness).** *The process algebra  $TSP_{\text{srt}}$  is a sound axiomatisation of strong bisimilarity on closed  $TSP_{\text{srt}}$ -terms.*

The following theorem states that every closed  $TSP_{\text{srt}}$ -term is derivably equal to a closed  $MPT_{\text{srt}}$ -term, i.e., that every occurrence of sequential composition in a closed term can be eliminated. The virtue of this theorem is that in future proofs about closed  $TSP_{\text{srt}}$ -terms we are allowed to apply induction on the structure of closed  $MPT_{\text{srt}}$ -terms or even basic terms instead of on the structure of closed  $TSP_{\text{srt}}$ -terms, thereby limiting the number of cases to be considered. An example of a proof in which this reduction of the number of cases to be considered is applied, is the proof of the completeness theorem (Theorem 15).

**Theorem 13 (Elimination).** *For every closed  $TSP_{\text{srt}}$ -term  $p$  there exists a closed  $MPT_{\text{srt}}$ -term  $q$  such that  $TSP_{\text{srt}} \vdash p = q$ .*

*Proof.* The only difference between closed  $TSP_{\text{srt}}$ -terms and closed  $MPT_{\text{srt}}$ -terms is that sequential composition cannot occur in the latter. For proving that all occurrences of sequential composition can be eliminated, it suffices to prove the following property: for every two closed  $MPT_{\text{srt}}$ -terms  $p_1$  and  $p_2$  there exists a closed  $MPT_{\text{srt}}$ -term  $q$  such that  $p_1 \cdot p_2 = q$ . This property is easily proven with induction on the structure of closed  $MPT_{\text{srt}}$ -term  $p_1$ .

Repeated application of this property to the smallest subterms of the form  $p_1 \cdot p_2$  that occur in a closed  $TSP_{\text{srt}}$ -term, eventually results in a closed  $MPT_{\text{srt}}$ -term.

**Theorem 14 (Conservativity).** *The process algebra  $TSP_{\text{srt}}$  is a conservative extension of the process algebra  $MPT_{\text{srt}}$ , i.e., for all closed  $MPT_{\text{srt}}$ -terms  $p$  and  $q$ :  $MPT_{\text{srt}} \vdash p = q$  iff  $TSP_{\text{srt}} \vdash p = q$ .*

*Proof.* This theorem follows, using the meta-theory of [28], from the completeness of  $MPT_{\text{srt}}$  (Theorem 9), from the soundness of  $TSP_{\text{srt}}$  (Theorem 12), and

from the observation that the term deduction system of  $TSP_{\text{srt}}$  is an operationally conservative extension of the term deduction system of  $MPT_{\text{srt}}$ . This last observation follows from the observations that both term deduction systems are in path format and that the term deduction system for  $MPT_{\text{srt}}$  is pure (see, e.g., [28] for a definition of pure deduction systems) and well-founded.

**Theorem 15 (Completeness).** *The process algebra  $TSP_{\text{srt}}$  is a complete axiomatization of strong bisimilarity on closed  $TSP_{\text{srt}}$ -terms.*

*Proof.* This theorem follows, using the meta-theory of [28], from the fact that  $TSP_{\text{srt}}$  is a conservative extension of  $MPT_{\text{srt}}$  (Theorem 14) and the elimination theorem (Theorem 13).

## 4 Parallelism and Communication

In this section, we extend the process algebra  $TSP_{\text{srt}}$  with the parallel composition operator  $\parallel$ . We also add the encapsulation operator  $\partial_H$ , which is well-known from standard process algebra. The operator  $\partial_H$  will block all actions from the set of actions  $H$  ( $H \subseteq A$ ). The operator is used to enforce communication between parallel components. The resulting process algebra is denoted  $TCP_{\text{srt}}$ .

The parallel composition of two processes  $x$  and  $y$ , notation  $x \parallel y$ , describes the interleaving of their behaviour. An action can be executed by  $x \parallel y$  if and only if one of the operands can execute this action or this action is the result of the simultaneous execution of an action from  $x$  and an action from  $y$ . This last possibility is called synchronisation or communication. The possible communications are specified by the *communication* function  $\gamma : A \times A \rightarrow A$ . This function is a parameter of the process algebra and can be chosen dependent on the application. The function  $\gamma$  is partial, commutative and associative.

A parallel composition  $x \parallel y$  can only terminate successfully if both  $x$  and  $y$  have an option to terminate successfully.

The timing behaviour of  $x \parallel y$  is as follows: a delay can take place only if both components allow this delay, or if one of the components can terminate successfully and the other can delay. The axiomatisation is now as follows, using two auxiliary operators  $\parallel\!\!\!|$  (left merge) and  $\mid$  (communication merge). The left-merge will capture that part of the behaviour of the merge where one components can execute an action independently and immediately, and the communication merge will capture that part of the behaviour of the merge where the initial activity is a termination, a delay or a synchronisation.

The axioms are given in Table 9. For the left operand of a left-merge operator the axioms are based on the structure of  $MPT_{\text{srt}}$ -terms. The axioms for the communication merge operator are based on the structure of  $MPT_{\text{srt}}$ -terms in both operands. Note that if both processes involved in communication merge can delay, the delays are synchronised.

In the operational rules we encounter a difficulty. Because of earlier decisions, we have to give an expression for a process after a certain delay has occurred. We find we have a need for an extra operator in order to express this. This is

**Table 9.** Axioms of  $TCP_{\text{srt}}$  ( $a, b \in A, t \in R^{\geq 0}, u \in R^{> 0}$ )

|   |  |
|---|--|
| $x \parallel y = (x \parallel y + y \parallel x) + x \mid y$                        | $x \mid y = y \mid x$  |
| $\tilde{\delta} \parallel x = \tilde{\delta}$                                       | $\tilde{\delta} \mid x = \tilde{\delta}$   |
| $\tilde{\epsilon} \parallel x = \tilde{\delta}$                                     | $\tilde{\epsilon} \mid \tilde{\epsilon} = \tilde{\epsilon}$  |
| $\tilde{a} . x \parallel y = \tilde{a} . (x \parallel y)$                           | $\tilde{a} . x \mid \tilde{\epsilon} = \tilde{\delta}$   |
| $(x + y) \parallel z = x \parallel z + y \parallel z$                               | $\tilde{a} . x \mid \tilde{b} . y = \tilde{c} . (x \parallel y) \quad \text{if } \gamma(a, b) = c$ |
| $\sigma^u . x \parallel y = \tilde{\delta}$   | $\tilde{a} . x \mid \tilde{b} . y = \tilde{\delta} \quad \text{otherwise}$                         |
| $\partial_H(\tilde{\delta}) = \tilde{\delta}$                                       | $\tilde{a} . x \mid \sigma^u . y = \tilde{\delta}$   |
| $\partial_H(\tilde{\epsilon}) = \tilde{\epsilon}$                                   | $\sigma^u . x \mid \tilde{\epsilon} = \sigma^u . x$  |
| $\partial_H(\tilde{a} . x) = \tilde{a} . \partial_H(x) \quad \text{if } a \notin H$ | $\sigma^u . x \mid \sigma^u . y = \sigma^u . (x \parallel y)$                                      |
| $\partial_H(\tilde{a} . x) = \tilde{\delta} \quad \text{if } a \in H$               | $(x + y) \mid z = x \mid z + y \mid z$   |
| $\partial_H(\sigma^t . x) = \sigma^t . \partial_H(x)$                               |  |
| $\partial_H(x + y) = \partial_H(x) + \partial_H(y)$                                 |  |

the time shift operator  $t \gg \_$ . The process  $t \gg x$  is the process that results after a delay has occurred of  $t$  time units starting at the current time. We present the operational rules in Table 10.

Rules for the left-merge and communication merge are given in Table 11.

In the axiomatisation of parallel composition, the time shift operator is not needed. It is not difficult to give axioms anyway. In this way, the signatures of the process algebra and the term deduction system will be equal. See Table 12.

Strong bisimilarity (Definition 2) is a congruence for the operators of the process algebra  $TCP_{\text{srt}}$ .

**Theorem 16 (Congruence).** *Strong bisimilarity is a congruence for the operators of the process algebra  $TCP_{\text{srt}}$ .*

*Proof.* The term deduction is in path format. Therefore congruence follows immediately.

It is a routine exercise to check that the axioms of  $TCP_{\text{srt}}$  are sound for our model of transition systems modulo strong bisimilarity.

**Theorem 17 (Soundness).** *The process algebra  $TCP_{\text{srt}}$  is a sound axiomatisation of strong bisimilarity on closed  $TCP_{\text{srt}}$ -terms.*

The newly introduced operators can be eliminated from every closed term. We skip the (quite elaborate!) proof.

**Theorem 18 (Elimination).** *For every closed  $TCP_{\text{srt}}$ -term  $p$  there exists a closed  $TSP_{\text{srt}}$ -term  $q$  such that  $TCP_{\text{srt}} \vdash p = q$ .*

**Table 10.** Deduction rules for parallel composition, encapsulation and time shift ( $a, b, c \in A$ ,  $H \subseteq A$ ,  $t, s \in R^{\geq 0}$ )

|   |   |   |
|---|---|---|
| $\frac{x \downarrow_t \quad y \downarrow_s}{x \parallel y \downarrow_{\max(t,s)}} [1]$  | $\frac{x \xrightarrow{a}_t x' \quad \Delta_t(y)}{x \parallel y \xrightarrow{a}_t x' \parallel (t \gg y)} [2]$ | $\frac{x \xrightarrow{a}_t x' \quad y \downarrow_s \quad s \leq t}{x \parallel y \xrightarrow{a}_t x'} [3]$ |
| $\frac{\Delta_t(x) \quad y \xrightarrow{a}_t y'}{x \parallel y \xrightarrow{a}_t (t \gg x) \parallel y'} [4]$                           | $\frac{x \downarrow_s \quad s \leq t \quad y \xrightarrow{a}_t y'}{x \parallel y \xrightarrow{a}_t y'} [5]$   |   |
| $\frac{x \xrightarrow{a}_t x' \quad y \xrightarrow{b}_t y' \quad \gamma(a,b) = c}{x \parallel y \xrightarrow{c}_t x' \parallel y'} [6]$ |   |   |
| $\frac{\Delta_t(x) \quad \Delta_t(y)}{\Delta_t(x \parallel y)} [7]$   | $\frac{\Delta_t(x) \quad y \downarrow_s \quad s \leq t}{\Delta_t(x \parallel y)} [8]$                         | $\frac{x \downarrow_s \quad \Delta_t(y) \quad s \leq t}{\Delta_t(x \parallel y)} [9]$                       |
| $\frac{x \downarrow_t}{\partial_H(x) \downarrow_t} [10]$  | $\frac{x \xrightarrow{a}_t x' \quad a \notin H}{\partial_H(x) \xrightarrow{a}_t \partial_H(x')} [11]$         | $\frac{\Delta_t(x)}{\Delta_t(\partial_H(x))} [12]$  |
| $\frac{x \downarrow_{t+s}}{s \gg x \downarrow_t} [13]$  | $\frac{x \xrightarrow{a}_{t+s} x'}{s \gg x \xrightarrow{a}_t x'} [14]$  | $\frac{\Delta_{t+s}(x)}{\Delta_t(s \gg x)} [15]$  |

**Table 11.** Deduction rules for left-merge and communication merge ( $a, b, c \in A$ ,  $t, s \in R^{\geq 0}$ ,  $u, v \in R^{> 0}$ )

|  |  |
|--|--|
| $\frac{x \xrightarrow{a}_0 x'}{x \parallel y \xrightarrow{a}_0 x' \parallel y} [1]$  | $\frac{x \downarrow_t \quad y \downarrow_s}{x \mid y \downarrow_{\max(t,s)}} [2]$                      |
| $\frac{x \xrightarrow{a}_u x' \quad \Delta_u(y)}{x \mid y \xrightarrow{a}_u x' \parallel (u \gg y)} [3]$                           | $\frac{x \xrightarrow{a}_u x' \quad y \downarrow_v \quad v \leq u}{x \mid y \xrightarrow{a}_u x'} [4]$ |
| $\frac{\Delta_u(x) \quad y \xrightarrow{a}_u y'}{x \mid y \xrightarrow{a}_u (u \gg x) \parallel y'} [5]$                           | $\frac{x \downarrow_v \quad v \leq u \quad y \xrightarrow{a}_u y'}{x \mid y \xrightarrow{a}_u y'} [6]$ |
| $\frac{x \xrightarrow{a}_t x' \quad y \xrightarrow{b}_t y' \quad \gamma(a,b) = c}{x \mid y \xrightarrow{c}_t x' \parallel y'} [7]$ |  |
| $\frac{\Delta_t(x) \quad \Delta_t(y)}{\Delta_t(x \mid y)} [8]$   | $\frac{\Delta_t(x) \quad y \downarrow_s \quad s \leq t}{\Delta_t(x \mid y)} [9]$                       |
| $\frac{x \downarrow_s \quad \Delta_t(y) \quad s \leq t}{\Delta_t(x \mid y)} [10]$  |  |

**Table 12.** Axioms of time shift ( $a \in A$ ,  $t, s \in R^{\geq 0}$ ,  $u \in R^{> 0}$ )

|   |                                     |
|---|-------------------------------------|
| $0 \gg x = x$                             | $t \gg \sigma^{t+s}.x = \sigma^s.x$ |
| $t \gg \tilde{\delta} = \tilde{\delta}$   | $(t+s) \gg \sigma^s.x = t \gg x$    |
| $u \gg \tilde{\epsilon} = \tilde{\delta}$ | $t \gg (x+y) = t \gg x + t \gg y$   |
| $u \gg \tilde{a}.x = \tilde{\delta}$      |                                     |

It is not hard to prove that the process algebra  $TCP_{\text{srt}}$  is a conservative extension of the process algebra  $TSP_{\text{srt}}$ . Combining this result with the elimination theorem, the completeness of  $TSP_{\text{srt}}$  gives that  $TCP_{\text{srt}}$  is complete. Note that we can also prove that  $TCP_{\text{srt}}$  is a conservative extension of  $MPT_{\text{srt}}$ .

**Theorem 19 (Conservativity).** *The process algebra  $TCP_{\text{srt}}$  is a conservative extension of the process algebra  $TSP_{\text{srt}}$ , i.e., for all closed  $TSP_{\text{srt}}$ -terms  $p$  and  $q$ :  $TSP_{\text{srt}} \vdash p = q$  iff  $TCP_{\text{srt}} \vdash p = q$ .*

*Proof.* This theorem follows, using the meta-theory of [28], from the completeness of  $TSP_{\text{srt}}$  (Theorem 15), from the soundness of  $TCP_{\text{srt}}$  (Theorem 17), and from the observation that the term deduction system of  $TCP_{\text{srt}}$  is an operationally conservative extension of the term deduction system of  $TSP_{\text{srt}}$ . This last observation follows from the observations that both term deduction systems are in path format and that the term deduction system for  $TSP_{\text{srt}}$  is pure and well-founded.

**Theorem 20 (Completeness).** *The process algebra  $TCP_{\text{srt}}$  is a complete axiomatization of strong bisimilarity on closed  $TCP_{\text{srt}}$ -terms.*

*Proof.* This theorem follows, using the meta-theory of [28], from the fact that  $TCP_{\text{srt}}$  is a conservative extension of  $TSP_{\text{srt}}$  (Theorem 19) and the elimination theorem (Theorem 18).

It is quite common to add to the axioms for  $TCP_{\text{srt}}$  the axioms of Table 13. These are called the *Axioms of Standard Concurrency*. For closed  $TCP_{\text{srt}}$ -terms these are derivable from the axioms of  $TCP_{\text{srt}}$ . This fact requires a quite lengthy proof.

**Table 13.** Axioms of Standard Concurrency

|  |
|--|
| $x \parallel \tilde{\epsilon} = x$ $(x \parallel y) \parallel z = x \parallel (y \parallel z)$ $(x \mid y) \mid z = x \mid (y \mid z)$ $(x \parallel y) \parallel z = x \parallel (y \parallel z)$ $(x \mid y) \parallel z = x \mid (y \parallel z)$ |
|--|

**Theorem 21.** *For closed  $TCP_{\text{srt}}$ -terms  $x$ ,  $y$ , and  $z$ , the equalities of Table 13 are derivable from the axioms of  $TCP_{\text{srt}}$ .*

Using the axioms of Standard Concurrency, we can derive the Expansion Theorem, generalizing the merge axiom to a parallel composition of a number of processes.

## 5 Relation with other Process Algebras

In the previous sections, the notion of conservative extension has already been used for comparing the process algebras we introduced. Conservative extension results can only be applied in case the syntax of one of the process algebras is fully included in the syntax of the other process algebra. For comparison of our process algebras with existing process algebras this is in general not the case. Therefore, we first introduce another means of comparing process algebras: embeddings.

Then, we first consider the embedding of the discrete-time process algebras into the real-time process algebras. Second, we consider the relation of the real-time versions of the process algebras we have introduced with some existing untimed process algebras. Finally, we relate our process algebras to some timed variants found in literature which do not have a constant for successful termination.

### 5.1 Comparing process algebras

A process algebra  $T' = (\Sigma', E')$  is a *conservative extension* of the process algebra  $T = (\Sigma, E)$  if (1)  $\Sigma \subseteq \Sigma'$  and (2) for all closed terms  $s$  and  $t$  over the signature  $\Sigma$ :  $(\Sigma, E) \vdash s = t$  if and only if  $(\Sigma', E') \vdash s = t$ . This is an interesting notion of comparing process algebras as it says that the smaller process algebra is contained precisely in the larger process algebra. No identities are lost and none are added.

An *explicit definition* of an operator or constant  $f$  in the process algebra  $(\Sigma, E)$  is an equation  $f(x_1, \dots, x_n) = t$  where  $t$  is a term over the signature  $\Sigma$  that does not contain other free variables than  $x_1, \dots, x_n$ . An extension of  $(\Sigma, E)$  with only explicitly defined operators and constants is called a *definitional extension* of  $(\Sigma, E)$ . The following theorem states that a definitional extension is a special type of conservative extension.

**Theorem 22.** *If the process algebra  $T'$  is a definitional extension of the process algebra  $T$ , then  $T'$  is a conservative extension of  $T$ .*

An *embedding* of a process algebra  $T = (\Sigma, E)$  in a process algebra  $T' = (\Sigma', E')$  is a term structure preserving injective mapping  $\hbar$  from the terms of  $T$  to the terms of  $T'$  such that for all closed terms  $s, t$  of  $T$ ,  $T \vdash s = t$  implies  $T' \vdash \hbar(s) = \hbar(t)$ .

**Theorem 23 (Baeten and Middelburg).** *Let  $T = (\Sigma, E)$  and  $T' = (\Sigma', E')$  be process algebras such that all operators from the signature  $\Sigma$  that are not in the signature  $\Sigma'$  can be defined in  $T'$  by explicit definitions. Let  $T'' = (\Sigma'', E'')$  be the resulting definitional extension of  $T'$ . If the axioms of  $T$  are derivable for closed terms in  $T''$ , then  $T$  can be embedded in  $T'$  as follows:  $\hbar(x) = x$ ,  $\hbar(f(t_1, \dots, t_n)) = f(\hbar(t_1), \dots, \hbar(t_n))$  if  $f$  in the signature of  $T'$ , and finally  $\hbar(f(t_1, \dots, t_n)) = t[\hbar(t_1), \dots, \hbar(t_n)/x_1, \dots, x_n]$  if the explicit definition of  $f$  is  $f(x_1, \dots, x_n) = t$ .*

## 5.2 Embedding Discrete Relative Timing into Continuous Relative Timing

The discrete relative timing process algebras are easily embedded into their continuous relative timing counterparts by interpreting  $\underline{\delta}$ ,  $\underline{\epsilon}$  and  $\underline{a}.$  as  $\tilde{\delta}$ ,  $\tilde{\epsilon}$  and  $\tilde{a}.$ , respectively.

However, this is not the embedding that conforms to our intuition. The embedding that does that, is the following:

- discrete time process  $\underline{\delta}$  is mapped to the real time process that allows any delay from the current point of time up to the end of the current time slice (up to the next integer value), and no further activity;
- discrete time process  $\underline{\epsilon}$  is mapped to the real time process that allows any delay from the current point of time up to the end of the current time slice followed by immediate termination ( $\tilde{\epsilon}$ );
- discrete time process  $\underline{a}.x$  is mapped to the real time process that allows any delay from the current point of time up to the end of the current time slice followed by the execution of  $a$  followed by the translation of  $x$ .

However, in order to define this embedding formally, we need to know the position of the current point of time in the current time slice, so that we can calculate the upper bound of the possible delay. This requires knowledge of absolute timing. This point was already noted in [12], section 4.3.2. Note that it is an advantage of absolute timing over relative timing, that such an embedding can be written down explicitly. There, such an embedding is eventually achieved by extending the theory first to parametric timing, a variant in which relative and absolute timing coexist. We refrain from doing this here, and just remark a similar approach works in the present setting.

## 5.3 Embedding Untimed Process Algebras with Explicit Termination

**Minimal Process Algebra** We compare the real-time process algebra with relative timing  $\text{MPT}_{\text{rt}}$  with the process algebra MPA from [13]. The signature of the process algebra MPA consists of constants for deadlock  $\delta$  and termination  $\epsilon$ , action prefix operators  $a.$  (for  $a \in A$ ), and alternative composition  $+$ . In this process algebra, where timing is only implicitly available in the interpretation of the action prefix operators, deadlock is a unit for alternative composition. The axioms of MPA are given in Table 14.

One embedding of MPA is where  $a.$  is mapped to  $\tilde{a}.$ , and where  $\delta, \epsilon$  are mapped to  $\tilde{\delta}, \tilde{\epsilon}$ . This is an embedding as all axioms remain valid. But it is a not very interesting embedding as any untimed process is interpreted as a timed process where all behaviour takes place at time 0. In such a setting, one might as well stick with the time-free theory. For this reason, we will not consider embeddings based on this interpretation anymore.

**Table 14.** Axioms of MPA

|                             |    |
|-----------------------------|----|
| $x + y = y + x$             | A1 |
| $(x + y) + z = x + (y + z)$ | A2 |
| $x + x = x$                 | A3 |
| $x + \delta = x$            | A6 |

Thus, this leads us to interpret a process  $a.x$  as a process where action  $a$  can be executed at any moment of time, i.e.  $a$  is executed after an arbitrary delay. Operationally, this means we have:

$$\Delta_t(a.x) \quad a.x \xrightarrow{a}_t \dots$$

for any  $t \in R^{\geq 0}$ .

Now it is still to be decided what remains of process  $a.x$  after  $a$  is executed. There are two possibilities.

1. In the timed setting, the process  $a.x$  executes the action  $a$  after an arbitrary delay and upon the execution of  $a$  the process  $x$  starts executing immediately.
2. In the timed setting, the process  $a.x$  executes the action  $a$  after an arbitrary delay and upon the execution of  $a$  the process  $x$  starts after an arbitrary delay.

Both of these interpretations yield embeddings, but we have a strong preference to use the first possibility and not the second. This can be motivated as follows.

Having realised the embedding, we want to use the arbitrary delay prefix  $a..$  and the urgent action prefix  $\tilde{a}..$  both together in the description of processes. This will be apparent in the examples further on. Thus, we will have terms of the form  $a.b.x$  and terms of the form  $a.\tilde{b}.x$ . In the first term, execution of  $b$  is some time after the execution of  $a$  (after an arbitrary delay), whereas in the second term, we intend to say that execution of  $b$  is immediately after the execution of  $a$  (after no delay). This forces us to use the first interpretation, so we have

$$\Delta_t(a.x) \quad a.x \xrightarrow{a}_t x$$

for any  $t \in R^{\geq 0}$ .

Note that the process algebra  $\text{MPA}_{drt}$  from [13] also uses this interpretation.

It remains now to give the embedding of the constants  $\delta, \epsilon$ . An important operator in process algebra is the encapsulation operator  $\partial_H$ , that blocks the execution of actions in  $H$  (usually to enforce communication). If  $a \in H$ , the time-free theory contains the axiom  $\partial_H(a.x) = \delta$  in cases where  $a \in H$  as it limits choices of embedding the time-free theory in the real-time theory. As the action prefix operator allows an initial arbitrary delay, we are forced to also

interpret  $\delta$  as having an initial arbitrary delay, i.e. the process  $\delta$  cannot execute any action, but does not block time. For this reason, this process can be called livelock. For more details, see [13]. Next,  $x + \delta = x$  is an axiom of the time-free theory (A6), i.e. this axiom holds for all time-free processes. Thus, it holds in particular for  $\epsilon$ , and thus, also the process  $\epsilon$  should allow an initial arbitrary delay. It turns out the time-free processes are characterized by the fact that before any action execution and before any termination, an arbitrary delay can occur.

In order to make things explicit, let us extend the real-time process algebra  $\text{MPT}_{\text{srt}}$  with the arbitrary delay prefix operator  $\sigma^* \dots$ . The process  $\sigma^*.x$  describes an arbitrary delay before  $x$  starts executing. The deduction rules for the arbitrary delay prefix operator are given in Table 15 and axioms can be found in Table 16.

**Table 15.** Deduction rules for arbitrary delay prefix operator ( $t, s \in R^{\geq 0}$ )

|  |  |                            |
|--|--|----------------------------|
| $\frac{x \downarrow_t}{\sigma^*.x \downarrow_{t+s}} [1]$ | $\frac{x \xrightarrow{a}_t x'}{\sigma^*.x \xrightarrow{a}_{t+s} x'} [2]$ | $\Delta_t(\sigma^*.x) [3]$ |
|--|--|----------------------------|

**Table 16.** Axioms of arbitrary delay prefix operator ( $t \in R^{\geq 0}$ )

|  |
|--|
| $\sigma^*.\sigma^*.x = \sigma^*.x$ $\sigma^*.\sigma^t.x = \sigma^t.\sigma^*.x$ $\sigma^*.x + \sigma^t.x = \sigma^*.x$ $\sigma^*.x + \sigma^*.y = \sigma^*.(x + y)$ $\sigma^*.x \cdot y = \sigma^*.(x \cdot y)$ $\partial_H(\sigma^*.x) = \sigma^*.\partial_H(x)$ |
|--|

Thus, we now have the following definitional extension:

- $a.x = \sigma^*.\tilde{a}.x$
- $\delta = \sigma^*.\tilde{\delta}$
- $\epsilon = \sigma^*.\tilde{\epsilon}$ .

In order to show we have an embedding, we need to derive the axioms of MPA for closed MPA-terms. We proceed to do this. The axioms A1-A3 are also part of  $\text{MPT}_{\text{srt}}$  and need therefore not be considered. For axiom A6 we use induction on the structure of closed MPA-terms:

- $x \equiv \epsilon$ . Then  $x + \delta = \epsilon + \delta = \sigma^*.\tilde{\epsilon} + \sigma^*.\tilde{\delta} = \sigma^*.\tilde{(\epsilon + \delta)} = \sigma^*.\tilde{\epsilon} = \epsilon = x$ .
- $x \equiv \delta$ . Then  $x + \delta = \delta + \delta = \delta = x$ .
- $x = a.x'$  for some  $a \in A$  and closed MPA-term  $x'$ . Then  $x + \delta = a.x' + \delta = \sigma^*.\tilde{a}.x' + \sigma^*.\tilde{\delta} = \sigma^*.\tilde{(a.x' + \delta)} = \sigma^*.\tilde{a}.x' = a.x'$ .
- $x \equiv x_1 + x_2$ . By induction we have  $x_1 + \delta = x_1$  and  $x_2 + \delta = x_2$ . Then  $x + \delta = (x_1 + x_2) + \delta = x_1 + (x_2 + \delta) = x_1 + x_2 = x$ .

**Sequential process algebra** Next, we briefly consider the embedding of SPA into  $\text{TSP}_{\text{srt}}$ . The process algebra SPA [13], is the extension of MPA with sequential composition. It has the axioms of Table 17 in addition to those from MPA.

**Table 17.** Axioms of SPA

|   |     |
|---|-----|
| $(x + y) \cdot z = x \cdot z + y \cdot z$   | A4  |
| $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ | A5  |
| $\delta \cdot x = \delta$                   | A7  |
| $\epsilon \cdot x = x$                      | A8  |
| $x \cdot \epsilon = x$                      | A9  |
| $a.x \cdot y = a.(x \cdot y)$               | A10 |

The axioms A4 and A5 are also part of  $\text{TSP}_{\text{srt}}$  and need therefore not be considered to show we have an embedding. Axiom A7 is easily derived as follows:

$$\delta \cdot x = (\sigma^*.\tilde{\delta}) \cdot x = \sigma^*.\tilde{(\delta \cdot x)} = \sigma^*.\tilde{\delta} = \delta.$$

The proof for axiom A6, again for both interpretations, is extended with the case that  $x \equiv x_1 \cdot x_2$  as follows, using the fact that axiom A7 can be derived for closed SPA-terms:

$$x + \delta = x_1 \cdot x_2 + \delta = x_1 \cdot x_2 + \delta \cdot x_2 = (x_1 + \delta) \cdot x_2 = x_1 \cdot x_2 = x.$$

Axiom A10 is derived as follows:

$$a.x \cdot y = \sigma^*.\tilde{a}.x \cdot y = \sigma^*.\tilde{(a.x \cdot y)} = \sigma^*.\tilde{a}.(x \cdot y) = a.(x \cdot y),$$

Next, the equality  $\sigma^*.x = x$  can be derived for closed SPA-terms by induction. The proof goes as follows:

- $x \equiv \epsilon$ . Then  $\sigma^*.x = \sigma^*.\epsilon = \sigma^*.\sigma^*.\tilde{\epsilon} = \sigma^*.\tilde{\epsilon} = \epsilon = x$ .
- $x \equiv \delta$ . Then  $\sigma^*.x = \sigma^*.\delta = \sigma^*.\sigma^*.\tilde{\delta} = \sigma^*.\tilde{\delta} = \delta = x$ .
- $x \equiv a.x'$  for some  $a \in A$  and closed SPA-term  $x'$ . Then  $\sigma^*.x = \sigma^*.a.x' = \sigma^*.\sigma^*.\tilde{a}.x' = \sigma^*.\tilde{a}.x' = a.x' = x$ .

- $x \equiv x_1 + x_2$  for some closed SPA-terms  $x_1$  and  $x_2$ . Then, using the induction hypotheses  $\sigma^*.x_1 = x_1$  and  $\sigma^*.x_2 = x_2$ , we have  $\sigma^*.x = \sigma^*.x_1 + \sigma^*.x_2 = x_1 + x_2 = x$ .
- $x \equiv x_1 \cdot x_2$  for some closed SPA-terms  $x_1$  and  $x_2$ . Then, using the induction hypothesis  $\sigma^*.x_1 = x_1$ , we have  $\sigma^*.x = \sigma^*.x_1 \cdot x_2 = x_1 \cdot x_2 = x$ .

Then, Axiom A8 is derived for closed SPA-terms  $x$  as follows:

$$\epsilon \cdot x = \sigma^*.\tilde{\epsilon} \cdot x = \sigma^*.(\tilde{\epsilon} \cdot x) = \sigma^*.x = x$$

Finally, Axiom A9 is obtained by induction on the structure of closed SPA-term  $x$  as follows.

- $x \equiv \epsilon$ . Then  $x \cdot \epsilon = \epsilon \cdot \epsilon = \sigma^*.\tilde{\epsilon} \cdot \epsilon = \sigma^*.(\tilde{\epsilon} \cdot \epsilon) = \sigma^*.\epsilon = \epsilon = x$ .
- $x \equiv \delta$ . Then  $x \cdot \epsilon = \delta \cdot \epsilon = \sigma^*.\tilde{\delta} \cdot \epsilon = \sigma^*.(\tilde{\delta} \cdot \epsilon) = \sigma^*.\tilde{\delta} = \delta = x$ .
- $x \equiv a.x'$  for some  $a \in A$  and closed SPA-term  $x'$ . Then using the induction hypothesis  $x' \cdot \epsilon = x'$ , we get  $x \cdot \epsilon = a.x' \cdot \epsilon = \sigma^*.\tilde{a}.x' \cdot \epsilon = \sigma^*.(\tilde{a}.x' \cdot \epsilon) = \sigma^*.\tilde{a} \cdot (x' \cdot \epsilon) = \sigma^*.\tilde{a}.x' = a.x' = x$ .
- $x \equiv x_1 + x_2$  for some closed SPA-terms  $x_1$  and  $x_2$ . Then, using the induction hypotheses  $x_1 \cdot \epsilon = x_1$  and  $x_2 \cdot \epsilon = x_2$ , we have  $x \cdot \epsilon = (x_1 + x_2) \cdot \epsilon = x_1 \cdot \epsilon + x_2 \cdot \epsilon = x_1 + x_2 = x$ .
- $x \equiv x_1 \cdot x_2$  for some closed SPA-terms  $x_1$  and  $x_2$ . Then, using the induction hypothesis  $x_2 \cdot \epsilon = x_2$ , we have  $x \cdot \epsilon = (x_1 \cdot x_2) \cdot \epsilon = x_1 \cdot (x_2 \cdot \epsilon) = x_1 \cdot x_2 = x$ .

**Interaction Process Algebra** The extension of SPA with parallel composition and communication, called IPA, from [13] cannot be embedded into  $\text{TCP}_{\text{srt}}$ . The reason for this, perhaps unexpected, result is that the intended meaning of the auxiliary operators  $\llbracket \_ \rrbracket$  and  $\llbracket \_ \rrbracket$  in these theories is fundamentally different. We conjecture that the subalgebras that are obtained by removing these auxiliary operators from the signature, allow for embeddings similar to the embeddings from MPA and SPA into  $\text{MPT}_{\text{drt}}$  and  $\text{TSP}_{\text{drt}}$ .

**Basic Process Algebra** We consider the process algebra  $\text{BPA}_{\delta\epsilon}$  of [29], which is similar to SPA with the difference that it does not have action prefix operators, but action constants instead. The axioms of  $\text{BPA}_{\delta\epsilon}$  are A1-A9.

The same reasoning as above leads us to interpret  $\delta$  as  $\sigma^*.\tilde{\delta}$  and  $\epsilon$  as  $\sigma^*.\tilde{\epsilon}$ . But now, we have to interpret the constant  $a$ . If we interpret  $a$  as  $\sigma^*.\tilde{a} \cdot \tilde{\epsilon}$ , the axiom  $x \cdot \epsilon = x$  cannot be derived as  $a \cdot \epsilon = \sigma^*.\tilde{a} \cdot \sigma^*.\tilde{\epsilon} \neq \sigma^*.\tilde{a} = a$ . On the other hand, if we interpret  $a$  as  $\sigma^*.\tilde{a} \cdot \sigma^*.\tilde{\epsilon}$ , then we run into problems with the interpretation of terms of the form  $a \cdot \tilde{b} \cdot x$ .

This observation has been the reason to redesign the hierarchy of ACP-related process algebras replacing action constants by action prefix in [13], [30].

#### 5.4 Embedding $\text{MPA}_{\text{drt}}$ into $\text{MPT}_{\text{drt}}$

In this section, we formalize the intuition that the process algebra  $\text{MPA}_{\text{drt}}$  from Section 2.1 can be embedded into the process algebra  $\text{MPT}_{\text{drt}}$  from Section 2.2.

We interpret, as mentioned before, the next time slice prefix operator from  $\text{MPA}_{drt}$  as the relative delay prefix operator with relative delay 1. Thereto we add the relative delay prefix operator to  $\text{MPT}_{drt}$  by means of the following explicit definition:

$$\sigma.x = \sigma^1.x.$$

Using Theorem 23, we only need to derive the axioms of  $\text{MPA}_{drt}$ , for closed  $\text{MPA}_{drt}$ -terms, in this definitional extension of  $\text{MPT}_{drt}$ . The only axiom of  $\text{MPA}_{drt}$  that is not also an axiom of  $\text{MPT}_{drt}$  is axiom DRTD from Table 1. It can be derived as follows:

$$\sigma.x + \sigma.y = \sigma^1.x + \sigma^1.y = \sigma^1.(x + y) = \sigma.(x + y).$$

### 5.5 Embedding Timed Process Algebras without Explicit Termination

In this section, we compare the process algebra  $\text{TSP}_{drt}$  with a relative timing process algebra from the literature that is very similar to  $\text{TSP}_{drt}$ , but does not contain a constant for undelayable termination. One of the few examples of such process algebras is the process algebra  $\text{BPA}_{-}^{drt} - \text{ID}$  from [12]. The signature of this process algebra consists of the urgent actions  $\underline{a}$  (for  $a \in A$ ), undelayable deadlock  $\underline{\delta}$ , alternative and sequential composition, and the unary relative delay operator  $\sigma^n(-)$  (for  $n \in N$ ). The axioms of this process algebra are given in Table 18.

**Table 18.** Axioms of  $\text{BPA}_{-}^{drt} - \text{ID}$  ( $m, n \in N$ )

|   |    |  |       |
|---|----|--|-------|
| $x + y = y + x$                             | A1 | $\sigma_{\text{rel}}^0(x) = x$   | DRT1  |
| $(x + y) + z = x + (y + z)$                 | A2 | $\sigma_{\text{rel}}^m(\sigma_{\text{rel}}^n(x)) = \sigma_{\text{rel}}^{m+n}(x)$     | DRT2  |
| $x + x = x$                                 | A3 | $\sigma_{\text{rel}}^n(x) + \sigma_{\text{rel}}^n(y) = \sigma_{\text{rel}}^n(x + y)$ | DRT3  |
| $(x + y) \cdot z = x \cdot z + y \cdot z$   | A4 | $\sigma_{\text{rel}}^n(x) \cdot y = \sigma_{\text{rel}}^n(x \cdot y)$                | DRT4  |
| $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ | A5 | $\underline{a} + \underline{\delta} = \underline{a}$                                 | A6DRa |
|   |    | $\sigma_{\text{rel}}^{n+1}(x) + \underline{\delta} = \sigma_{\text{rel}}^{n+1}(x)$   | A6DRb |
|   |    | $\underline{\delta} \cdot x = \underline{\delta}$                                    | A7DR  |

The undelayable actions  $\underline{a}$  (for  $a \in A$ ) and the unary relative delay operator  $\sigma^n$  of  $\text{BPA}_{-}^{drt} - \text{ID}$  can be defined explicitly on  $\text{TSP}_{drt}$  by the equations  $\underline{a} = \underline{a}.\epsilon$  and  $\sigma^n(x) = \sigma^n.x$ , respectively.

We show that, for closed terms over the signature of  $\text{BPA}_{-}^{drt} - \text{ID}$ , the axioms of  $\text{BPA}_{-}^{drt} - \text{ID}$  are derivable from this definitional extension of  $\text{TSP}_{drt}$ .

There to, we have to consider each of the axioms of  $\text{BPA}_{\text{dr}}^{\text{dr}} - \text{ID}$ . The axioms A1-A5, and A7DR of  $\text{BPA}_{\text{dr}}^{\text{dr}} - \text{ID}$  are also axioms of  $\text{TSP}_{\text{dr}}$  and need therefore not be considered. For the other axioms we have the following derivations (for  $n, m \in \mathbb{N}$ ):

$$\text{(DRT1)} \quad \sigma^0(x) = \sigma^0.x = x,$$

$$\text{(DRT2)} \quad \sigma^n(\sigma^m(x)) = \sigma^n.\sigma^m.x = \sigma^{n+m}.x = \sigma^{n+m}(x),$$

$$\text{(DRT3)} \quad \sigma^n(x) + \sigma^n(y) = \sigma^n.x + \sigma^n.y = \sigma^n.(x + y) = \sigma^n(x + y),$$

$$\text{(DRT4)} \quad \sigma^n(x) \cdot y = (\sigma^n.x) \cdot y = \sigma^n.(x \cdot y) = \sigma^n(x \cdot y),$$

$$\text{(A6DRa)} \quad \underline{a} + \underline{\delta} = \underline{a},$$

$$\text{(A6DRb)} \quad \sigma^{n+1}(x) + \underline{\delta} = \sigma^{n+1}(x).$$

Then, as a consequence of Theorem 23, we have that  $\text{BPA}_{\text{dr}}^{\text{dr}} - \text{ID}$  can be embedded in  $\text{TSP}_{\text{srt}}$  using  $\tilde{h}$  defined by  $\tilde{h}(\underline{\delta}) = \underline{\delta}$ ,  $\tilde{h}(\underline{a}) = \underline{a}.\underline{\epsilon}$ ,  $\tilde{h}(x_1 + x_2) = \tilde{h}(x_1) + \tilde{h}(x_2)$ ,  $\tilde{h}(x_1 \cdot x_2) = \tilde{h}(x_1) \cdot \tilde{h}(x_2)$ ,  $\tilde{h}(\sigma^t(x)) = \sigma^t.\tilde{h}(x)$ .

## 6 Time-free projection, Abstraction and Recursion, $\Sigma$ -notations

In this section, we discuss some additional operators that will be used in the examples we discuss in the following section.

### 6.1 Time-free projection

First of all, we discuss an operator that abstracts from all timing information. This means a process is turned into a process in time-free process algebra. As we saw in the previous section, in time-free process algebra, there are arbitrary delays between all activities. We introduce the *time-free projection* operator  $\pi_{\text{tf}}$ , with the axioms in Table 19. The process  $\pi_{\text{tf}}(x)$  has the same action and termination behaviour as the process  $x$ , but it also allows arbitrary time steps in between.

**Table 19.** Axioms for time-free projection ( $a \in A$ ,  $t \in \mathbb{R}^{\geq 0}$ )

|  |
|--|
| $\pi_{\text{tf}}(\tilde{\delta}) = \delta$                         |
| $\pi_{\text{tf}}(\tilde{\epsilon}) = \epsilon$                     |
| $\pi_{\text{tf}}(\tilde{a}.x) = a.\pi_{\text{tf}}(x)$              |
| $\pi_{\text{tf}}(x + y) = \pi_{\text{tf}}(x) + \pi_{\text{tf}}(y)$ |
| $\pi_{\text{tf}}(\sigma^t.x) = \pi_{\text{tf}}(x)$                 |

## 6.2 Abstraction

In the calculations to come, abstraction from internal activity will play an important role. In time-free process algebra, abstraction and internal steps are often approached by use of the notion of *branching bisimulation*, see [31]. This notion can be formulated by analogy in timed theories also, see e.g. [12]. We will refrain from doing this here, however, because we will have reason to change this notion in the sequel. Instead, at this point we just give an axiomatization of timed branching bisimulation, adapted from [12]. This axiomatization uses an auxiliary operator  $\nu$  (“now”), that strips away any possible initial delay behaviour of a process. This operator is used in the timed variants of the branching axiom (the first two axioms), in order to ensure that a term starting with a delay is not split.

The axioms are given in Table 20. Obviously, the now operator and the abstraction operator can easily be eliminated from every closed term.

**Table 20.** Axioms for abstraction ( $a \in A_\tau$ ,  $I \subseteq A$ ,  $t \in R^{\geq 0}$ ,  $u \in R^{>0}$ )

|  |
|--|
| $\tilde{a} . (\tilde{\tau} . (\nu(x) + y) + \nu(x)) = \tilde{a} . (\nu(x) + y)$ $\tilde{a} . (\tilde{\tau} . (\nu(x) + y) + y) = \tilde{a} . (\nu(x) + y)$ $\tilde{a} . (\sigma^u . \tilde{\tau} . x + \nu(y)) = \tilde{a} . (\sigma^u . x + \nu(y))$<br>$\nu(\tilde{\delta}) = \tilde{\delta}$ $\nu(\tilde{\epsilon}) = \tilde{\epsilon}$ $\nu(\tilde{a} . x) = \tilde{a} . x$ $\nu(x + y) = \nu(x) + \nu(y)$ $\nu(\sigma^u . x) = \tilde{\delta}$<br>$\tau_I(\tilde{\delta}) = \tilde{\delta}$ $\tau_I(\tilde{\epsilon}) = \tilde{\epsilon}$ $\tau_I(\tilde{a} . x) = \tilde{\tau} . \tau_I(x) \quad \text{if } a \in I$ $\tau_I(\tilde{a} . x) = \tilde{a} . \tau_I(x) \quad \text{if } a \notin I$ $\tau_I(x + y) = \tau_I(x) + \tau_I(y)$ $\tau_I(\sigma^t . x) = \sigma^t . \tau_I(x)$ |
|--|

### 6.3 Recursion

A recursive specification over  $\text{TCP}_{\text{srt}}$  is a set of recursive equations  $E = \{X = t_X \mid X \in V\}$  where  $V$  is a set of variables and each  $t_X$  is a  $\text{TCP}_{\text{srt}}$ -term that only contains variables from  $V$ .

A recursive specification is called a *completely guarded* recursive specification if all occurrences of variables in terms  $t_X$  (for  $X \in V$ ) occur in the context of an action prefix operator or a time prefix operator with positive delay. A recursive specification is called a *guarded* recursive specification if it can be rewritten into a completely guarded recursive specification using the axioms of  $\text{TCP}_{\text{srt}}$  and the equations from the recursive specification.

We can assume that we work in a setting where all guarded recursive specifications have unique solutions. The necessary theory underpinning this assumption can be found e.g. in [12].

### 6.4 Generalized alternative composition

For a finite set  $D$  we define  $\sum_{d \in D} p$  recursively as follows: for  $v \notin D$

$$\sum_{d \in \emptyset} p = \tilde{\delta} \quad \sum_{d \in D \cup \{v\}} p = p[d := v] + \sum_{d \in D} p.$$

For simplicity, we only consider the case here where we use the sum notation for a finite set of data, or a finite set of time units. A generalization to an infinite set of data, or even an uncountable set of time points, will require additional theory.

## 7 Some Simple Calculations: Communicating Buffers

In this section, we give some simple calculations in order to illustrate the use of our relative-time theory.

In the calculations to come we use the so called *standard communication function*. Suppose we have given two finite sets, the set of messages or data  $D$ , and the set of ports  $P$ . For each  $d \in D$  and  $i \in P$ , we have atomic actions  $s_i(d)$ ,  $r_i(d)$  and  $c_i(d)$  (denoting send, receive and communication of datum  $d$  along port  $i$ ) and the only defined communications are

$$\gamma(s_i(d), r_i(d)) = \gamma(r_i(d), s_i(d)) = c_i(d).$$

for any  $d \in D$  and  $i \in P$ .

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

$$B^{ij} = \sum_{d \in D} r_i(d).s_j(d).B^{ij}.$$

A straightforward computation shows that the parallel composition of two such buffers placed in sequence (connecting the output port of the first with the input port of the second) with the corresponding encapsulation of internal send and receive events and abstraction from internal communication events results in a two-place buffer:

$$\tau_I \circ \partial_H(B^{ij} \parallel B^{jk}) = B_2^{ik},$$

where  $H = \{s_j(d), r_j(d) \mid d \in D\}$  and  $I = \{c_j(d) \mid d \in D\}$ .

In the remainder of this section we consider timed versions of such buffers. It is possible to use the discrete time theory, as all points of interest can be discussed. The unit of time is chosen in such a way that at most one input per time slice is possible.

In time-free process algebra, combining two one-place buffers yields a two-place buffer. This is the case when arbitrary delays are added at each point. Adding timing constraints in different ways can yield instead a one-place or a two-place buffer in different ways, as we will see.

### 7.1 No-delay One-place Buffers

The first type of one-place buffer with input port  $i$  and output port  $j$  we consider allows at most one input in every time slice, and outputs with no delay:

$$C^{ij} = \sum_{d \in D} r_i(d). \underline{\underline{s_j(d)}}. \sigma. C^{ij},$$

or equivalently

$$C^{ij} = \sum_{d \in D} \underline{\underline{r_i(d)}}. \underline{\underline{s_j(d)}}. \sigma. C^{ij} + \sigma. C^{ij}.$$

If we consider the time-free projection of  $C^{ij}$ , i.e.,  $\pi_{\text{tf}}(C^{ij})$  we obtain

$$\pi_{\text{tf}}(C^{ij}) = \sum_{d \in D} r_i(d). s_j(d). \pi_{\text{tf}}(C^{ij})$$

which satisfies the specification of the time-free one-place buffer. Hence, this timed one-place buffer can be considered an implementation of the time-free one-place buffer.

With  $H = \{s_j(d), r_j(d)\}$  we can derive

$$\begin{aligned} & \partial_H(C^{ij} \parallel C^{jk}) \\ &= \partial_H(C^{ij} \parallel C^{jk}) + \partial_H(C^{jk} \parallel C^{ij}) + \partial_H(C^{ij} \mid C^{jk}) \\ &= \sum_{d \in D} \underline{\underline{r_i(d)}}. \partial_H(\underline{\underline{s_j(d)}}. \sigma. C^{ij} \parallel C^{jk}) + \sigma. \partial_H(C^{ij} \parallel C^{jk}) \\ &= \sum_{d \in D} \underline{\underline{r_i(d)}}. \underline{\underline{c_j(d)}}. \partial_H(\sigma. C^{ij} \parallel \underline{\underline{s_k(d)}}. \sigma. C^{jk}) \\ &= \sum_{d \in D} \underline{\underline{r_i(d)}}. \underline{\underline{c_j(d)}}. \underline{\underline{s_k(d)}}. \partial_H(\sigma. \underline{\underline{C^{ij}}} \parallel \sigma. C^{jk}) \\ &= \sum_{d \in D} \underline{\underline{r_i(d)}}. \underline{\underline{c_j(d)}}. \underline{\underline{s_j(d)}}. \sigma. \partial_H(C^{ij} \parallel C^{jk}) \end{aligned}$$

Next, abstracting from the internal communication actions over port 2 using  $I = \{c_j(d) \mid d \in D\}$  gives:

$$\begin{aligned}
& \tau_I \circ \partial_H(C^{ij} \parallel C^{jk}) \\
&= \tau_I(\sum_{d \in D} r_i(d). \underline{c_j(d)}. \underline{s_k(d)}. \sigma. \partial_H(C^{ij} \parallel C^{jk})) \\
&= \sum_{d \in D} r_i(d). \underline{\tau_I}. \underline{s_k(d)}. \sigma. \tau_I \circ \partial_H(C^{ij} \parallel C^{jk}) \\
&= \sum_{d \in D} r_i(d). \underline{s_k(d)}. \sigma. \tau_I \circ \partial_H(C^{ij} \parallel C^{jk})
\end{aligned}$$

Observe that the composition again behaves as a no-delay channel, with input port  $i$  and output port  $k$ , i.e.,

$$\tau_I \circ \partial_H(C^{ij} \parallel C^{jk}) = C^{ik}.$$

We see that this is very different behaviour from the time-free case.

## 7.2 Unit-delay One-place Buffers

Consider

$$D^{ij} = \sum_{d \in D} r_i(d). \sigma. \underline{s_j(d)}. D^{ij}.$$

This specification describes a buffer with capacity one and a delay between input and output of one time unit. Again, if we consider the time-free projection of  $D^{ij}$ , i.e.,  $\pi_{\text{tf}}(D^{ij})$  we obtain

$$\pi_{\text{tf}}(D^{ij}) = \sum_{d \in D} r_i(d). s_j(d). \pi_{\text{tf}}(D^{ij})$$

which satisfies the specification of the time-free one-place buffer. Hence, this timed one-place buffer can be considered as a different implementation of the time-free one-place buffer. Now define  $X = \partial_H(D^{ij} \parallel D^{jk})$  and  $X_d = \partial_H(D^{ij} \parallel \sigma. \underline{s_k(d)}. D^{jk})$  for  $d \in D$ . Then the following recursive specification can be derived for variables  $X, X_d$ :

$$\begin{aligned}
& X \\
&= \partial_H(D^{ij} \parallel D^{jk}) \\
&= \partial_H(D^{ij} \parallel D^{jk}) + \partial_H(D^{jk} \parallel D^{ij}) + \partial_H(D^{ij} \parallel D^{jk}) \\
&= \sum_{d \in D} \underline{r_i(d)}. \partial_H(\sigma. \underline{s_j(d)}. D^{ij} \parallel D^{jk}) + \sigma. \partial_H(D^{ij} \parallel D^{jk}) \\
&= \sum_{d \in D} r_i(d). \sigma. \partial_H(\underline{s_j(d)}. D^{ij} \parallel D^{jk}) \\
&= \sum_{d \in D} r_i(d). \sigma. \underline{c_j(d)}. \partial_H(D^{ij} \parallel \sigma. \underline{s_k(d)}. D^{jk}) \\
&= \sum_{d \in D} r_i(d). \sigma. \underline{c_j(d)}. X_d
\end{aligned}$$

and, for  $d \in D$

$$\begin{aligned}
& X_d \\
&= \partial_H(D^{ij} \parallel \sigma.\underline{s_k}(d).D^{jk}) \\
&= \sum_{e \in D} \underline{r_i}(e).\partial_H(\sigma.\underline{s_j}(e).D^{ij} \parallel \sigma.\underline{s_k}(d).D^{jk}) + \sigma.\partial_H(D^{ij} \parallel \underline{s_k}(d).D^{jk}) \\
&= \sum_{e \in D} \underline{r_i}(e).\sigma.\partial_H(\underline{s_j}(e).D^{ij} \parallel \underline{s_k}(d).D^{jk}) + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\partial_H(\sigma.\underline{s_j}(e).D^{ij} \parallel \underline{s_k}(d).D^{jk}) + \underline{s_k}(d).\partial_H(D^{ij} \parallel D^{jk})) \\
&= \sum_{e \in D} \underline{r_i}(e).\sigma.\underline{s_k}(d).\partial_H(\underline{s_j}(e).D^{ij} \parallel D^{jk}) + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\partial_H(\sigma.\underline{s_j}(e).D^{ij} \parallel D^{jk}) + \underline{s_k}(d).X) \\
&= \sum_{e \in D} \underline{r_i}(e).\sigma.\underline{s_k}(d).c_j(e).\partial_H(D^{ij} \parallel \sigma.\underline{s_k}(e).D^{jk}) + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\sigma.\partial_H(\underline{s_j}(e).D^{ij} \parallel D^{jk}) + \underline{s_k}(d).X) \\
&= \sum_{e \in D} \underline{r_i}(e).\sigma.\underline{s_k}(d).c_j(e).\underline{X_e} + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\sigma.c_j(e).\partial_H(D^{ij} \parallel \sigma.\underline{s_k}(e).D^{jk}) + \underline{s_k}(d).X) \\
&= \sum_{e \in D} \underline{r_1}(e).\sigma.\underline{s_k}(d).c_2(e).\underline{X_e} + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\sigma.c_j(e).\underline{X_e} + \underline{s_k}(d).X)
\end{aligned}$$

Abstraction from the internal communications gives:

$$\begin{aligned}
\tau_I(X) &= \sum_{d \in D} \underline{r_i}(d).\sigma.\underline{\tau}.\tau_I(X_d) \\
&= \sum_{d \in D} \underline{r_i}(d).\sigma.\tau_I(X_d) \\
\tau_I(X_d) &= \sum_{e \in D} \underline{r_i}(e).\sigma.\underline{s_k}(d).\underline{\tau}.\tau_I(X_e) + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\sigma.\underline{\tau}.\tau_I(X_e) + \underline{s_k}(d).\tau_I(X)) \\
&= \sum_{e \in D} \underline{r_i}(e).\sigma.\underline{s_k}(d).\tau_I(X_e) + \\
&\quad \sigma.(\sum_{e \in D} \underline{r_i}(e).\underline{s_k}(d).\sigma.\tau_I(X_e) + \underline{s_k}(d).\tau_I(X))
\end{aligned}$$

Observe that this denotes a two-place buffer with a delay of two time units. Thus, in this case we do obtain a result in accordance with the time-free theory. It can be claimed that this is a more faithful timed variant of the time-free buffer as in this version, each transportation from input to output takes one unit of time.

### 7.3 Unit-delay One-place buffers II

To conclude this section, consider yet another variant of a one-place buffer.

$$E^{ij} = \sum_{d \in D} \underline{r_i}(d).\sigma.(\underline{s_j}(d).\underline{\epsilon} \parallel E^{ij})$$

Again, transportation from input to output take one unit of time, but now, a new input can be accepted in every time slice. Thus, it can be said that this implementation is *input-enabled*.

The composition  $\tau_I(\partial_H(E^{ij} \parallel E^{jk}))$  now satisfies the following recursive specification:

$$\begin{aligned} Y &= \sum_{d \in D} r_i(d) \cdot \sigma \cdot Y'_d \\ Y'_d &= \sigma \cdot (\underline{s_k(d)} \cdot Y + \sum_{e \in D} \underline{r_i(e)} \cdot \underline{s_k(d)} \cdot \sigma \cdot Y'_e) + \sum_{e \in D} \underline{r_i(e)} \cdot \sigma \cdot Y''_{de} \\ Y''_{de} &= \underline{s_k(d)} \cdot Y'_e + \sum_{f \in D} \underline{r_i(f)} \cdot \underline{s_k(d)} \cdot \sigma \cdot Y''_{ef} \end{aligned}$$

We see that it is possible that three data elements reside in this composition at the same time.

The time-free abstraction of this process gives:

$$\begin{aligned} \pi_{\text{tf}}(Y) &= \sum_{d \in D} r_i(d) \cdot \pi_{\text{tf}}(Y'_d) \\ \pi_{\text{tf}}(Y'_d) &= s_k(d) \cdot X + \sum_{e \in D} r_i(e) \cdot s_k(d) \cdot \pi_{\text{tf}}(Y'_e) + \sum_{e \in D} r_i(e) \cdot \pi_{\text{tf}}(Y''_{de}) \\ \pi_{\text{tf}}(Y''_{de}) &= s_k(d) \cdot \pi_{\text{tf}}(Y'_e) + \sum_{f \in D} r_i(f) \cdot s_k(d) \cdot \pi_{\text{tf}}(Y''_{ef}) \end{aligned}$$

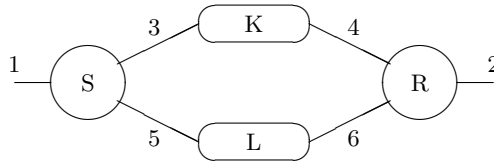
## 8 Case Study: PAR Protocol

In the following example, we describe a simple communication protocol. A communication protocol concerns the transmission of data through an unreliable channel, such that no data will get lost. The example shows that with the basic concepts for timing introduced so far, we are able to describe the protocol in a satisfactory way. This is not possible with the concepts around which ACP (without timing) has been set up.

### 8.1 Description of the PAR Protocol

We consider a simple version of the communication protocol known as the *PAR* (Positive Acknowledgement with Retransmission) protocol.

The configuration of the PAR protocol is shown in Fig. 1 by means of a connection diagram. The sender waits for an acknowledgement before a new



**Fig. 1.** Connection diagram for PAR protocol

datum is transmitted. If an acknowledgement is not received within a complete protocol cycle, the old datum is retransmitted. In order to avoid duplicates due to retransmission, data are labeled with an alternating bit from  $B = \{0, 1\}$ .

We have a sender process  $S$ , a receiver process  $R$ , and two channels  $K$  and  $L$ . The process  $S$  waits until a datum  $d$  is offered at an external port (port 1). When a datum, say  $d$ , is offered at this port,  $S$  consumes it, packs it with an alternating bit  $b$  in a frame  $(d, b)$ , and then delivers the frame at an internal port used for sending (port 3). Next,  $S$  waits until an acknowledgement  $ack$  is offered at an internal port used for receiving (port 5). When the acknowledgement does not arrive within a certain time period,  $S$  delivers the same frame again and goes back to waiting for an acknowledgement. When the acknowledgement arrives within that time period,  $S$  goes back to waiting for a datum. The process  $R$  waits until a frame with a datum and an alternating bit  $(d, b)$  is offered at an internal port used for receiving (port 4). When a frame is offered at this port,  $R$  consumes it, unpacks it, and then delivers the datum  $d$  at an external port (port 2) if the alternating bit  $b$  is the right one and in any case delivers an acknowledgement  $ack$  at an internal port used for sending (port 6). After that,  $R$  goes back to waiting for a frame, but the right bit changes to  $(1 - b)$  if the alternating bit was the right one. The processes  $K$  and  $L$  pass on frames from an internal port of  $S$  to an internal port of  $R$  and acknowledgements from an internal port of  $R$  to an internal port of  $S$ , respectively. Because the channels are supposed to be unreliable, they may produce an error instead of passing on frames or acknowledgements. The times  $t_1, \dots, t_4$  are the times that it takes the different processes to pack and deliver, to unpack and deliver or simply to deliver what they consume. The time  $t'_1$  is the time-out time of the sender, i.e., the time after which it retransmits a datum in case it is still waiting for an acknowledgement. The time  $t'_2$  is the time that it takes the receiver to produce and deliver an acknowledgement.

We assume a finite set of data  $D$ . Let  $F = D \times B$  be the set of frames. For  $d \in D$  and  $b \in B$ , we write  $d, b$  for the frame  $(d, b)$ .

Again, we use the standard communication function, so assuming a finite set of data  $D$ , the communication function is defined such that

$$\gamma(s_i(d), r_i(d)) = \gamma(r_i(d), s_i(d)) = c_i(d)$$

for all  $d \in D$ , and it is undefined otherwise. The recursive specification of the sender consists of the following equations: for every  $b \in B$  and  $d \in D$

$$\begin{aligned} S &= S_0 \\ S_b &= \sum_{d \in D} r_1(d) \cdot \sigma^{t_1} \cdot SF_{d,b} \\ SF_{d,b} &= \underline{s_3(d, b)} \cdot \left( \sum_{k < t'_1} \sigma^k \cdot \underline{r_5(ack)} \cdot S_{1-b} + \sigma^{t'_1} \cdot SF_{d,b} \right) \end{aligned}$$

The recursive specification of the receiver consists of the following equations: for every  $b \in B$

$$\begin{aligned} R &= R_0 \\ R_b &= \sum_{d \in D} r_4(d, b) \cdot \sigma^{t_2} \cdot \underline{s_2(d)} \cdot \sigma^{t'_2} \cdot \underline{s_6(ack)} \cdot R_{1-b} \\ &\quad + \sum_{d \in D} r_4(d, 1-b) \cdot \sigma^{t'_2} \cdot \underline{s_6(ack)} \cdot R_b \end{aligned}$$

Each of the channels is recursively defined by a single equation:

$$\begin{aligned} K &= \sum_{f \in F} r_3(f). \left( \sigma^{t_3}. \underline{s_4(f)}. K + \sum_{k \leq t_3} \sigma^k. \underline{error}. K \right) \\ L &= r_6(ack). \left( \sigma^{t_4}. \underline{s_5(ack)}. L + \sum_{k \leq t_4} \sigma^k. \underline{error}. L \right). \end{aligned}$$

The whole system is described by the following term:

$$\partial_H(S \parallel K \parallel L \parallel R),$$

where

$$\begin{aligned} H &= \{s_i(f) \mid i \in \{3, 4\}, f \in F\} \cup \{r_i(f) \mid i \in \{3, 4\}, f \in F\} \\ &\quad \cup \{s_i(ack) \mid i \in \{5, 6\}\} \cup \{r_i(ack) \mid i \in \{5, 6\}\}. \end{aligned}$$

This protocol is only correct if the time-out time  $t'_1$  is longer than a complete protocol cycle, i.e., if  $t'_1 > t_2 + t'_2 + t_3 + t_4$ . If the time-out time is shorter than a complete protocol cycle, the time-out is called premature. In that case, while an acknowledgement is still on the way, the sender will retransmit the current frame. When the acknowledgement finally arrives, the sender will treat this acknowledgement as an acknowledgement of the retransmitted frame. However, an acknowledgement of the retransmitted frame may be on the way. If the next frame transmitted gets lost and the latter acknowledgement arrives, no retransmission of that frame will follow and the protocol will fail.

There have been attempts to describe this protocol using process algebra without timing. These attempts are unsatisfactory. In most attempts the premature time-out of an acknowledgement is excluded by inhibiting a time-out so long as it does not lead to deadlock! In [32], two ways in which that can be established are described: by means of a priority operator (not treated here) and by means of communication between three processes. In other attempts, the premature time-out of an acknowledgement is not excluded at all.

## 8.2 Analysis of the PAR Protocol

We now have all we need to analyze the PAR protocol.

We use the expansion theorem in order to derive a recursive specification of the process  $\partial_H(S \parallel K \parallel L \parallel R)$ .

We introduce some auxiliary variables in order to facilitate expansion. We rewrite the recursive specifications of  $S$ ,  $R$ ,  $K$  and  $L$  as follows. We refrain from mentioning after each equation schema that there is an instance for every  $d \in D$

and/or  $b \in B$ .

$$\begin{aligned}
S &= S_0, \\
S_b &= \sum_{d \in D} r_1(d).S'_{d,b}, \\
S'_{d,b} &= \sigma^{t_1}.\underline{s_3(d,b)}.S''_{d,b}, \\
S''_{d,b} &= \sum_{k < t'_1} \sigma^k.\underline{r_5(ack)}.S_{1-b} + \sigma^{t'_1}.\underline{s_3(d,b)}.S''_{d,b}, \\
R &= R_0, \\
R_b &= \sum_{d \in D} r_4(d,b).R'_{d,b} + \sum_{d \in D} r_4(d,1-b).R''_b, \\
R'_{d,b} &= \sigma^{t_2}.\underline{s_2(d)}.R''_{1-b}, \\
R''_b &= \sigma^{t'_2}.\underline{s_6(ack)}.R_b, \\
K &= \sum_{(d,b) \in D \times B} \underline{r_3(d,b)}.K'_{d,b} + \sigma.K, \\
K'_{d,b} &= \sigma^{t_3}.\underline{s_4(d,b)}.K + \sum_{k \leq t_3} \sigma^k.\underline{error}.K, \\
L &= \underline{r_6(ack)}.L' + \sigma.L, \\
L' &= \sigma^{t_4}.\underline{s_5(ack)}.L + \sum_{k \leq t_4} \sigma^k.\underline{error}.L.
\end{aligned}$$

Secondly, we expand the term  $\partial_H(S_b \parallel K \parallel L \parallel R_b)$  by repeated application of the expansion theorem. We remove in each step immediately those alternatives that are known to be equal to  $\sigma^n(\underline{\delta})$  (for some  $n \geq 0$ ) because of incapability to communicate, encapsulation or timing conflict, provided the removal is justified by the fact that  $\sigma^m(t) + \sigma^n(\underline{\delta}) = \sigma^m(t)$  is derivable for all closed terms  $t$  and for all  $m \geq n$ . In the expansion, we will use the following abbreviation for every  $d \in D$ ,  $b \in B$  and  $t > 0$ :

$$S''_{d,b,t} \text{ for } \sum_{k < t} \sigma^k.\underline{r_5(ack)}.S_{1-b} + \sigma^t.\underline{s_3(d,b)}.S''_{d,b}.$$

Again, we refrain from mentioning after each equation schema that there is an instance for every  $d \in D$  and/or  $b \in B$ .

$$\begin{aligned}
&\partial_H(S_b \parallel K \parallel L \parallel R_b) \\
&= \sum_{d \in D} r_1(d).\partial_H(S'_{d,b} \parallel K \parallel L \parallel R_b),
\end{aligned}$$

$$\partial_H(S'_{d,b} \parallel K \parallel L \parallel R_b) = \sigma^{t_1}.\underline{c_3(d,b)}.\partial_H(S''_{d,b} \parallel K'_{d,b} \parallel L \parallel R_b),$$

$$\begin{aligned}
&\partial_H(S''_{d,b} \parallel K'_{d,b} \parallel L \parallel R_b) \\
&= \sigma^{t_3}.\underline{c_4(d,b)}.\partial_H(S''_{d,b,t'_1-t_3} \parallel K \parallel L \parallel R'_{d,b}) \\
&\quad + \sum_{k \leq t_3} \sigma^k.\underline{error}.\partial_H(S''_{d,b,t'_1-k} \parallel K \parallel L \parallel R_b),
\end{aligned}$$

$$\begin{aligned}
&\partial_H(S''_{d,b,t} \parallel K \parallel L \parallel R'_{d,b}) = \sigma^{t_2}.\underline{s_2(d)}.\partial_H(S''_{d,b,t-t_2} \parallel K \parallel L \parallel R''_{1-b}) \\
&\text{(for every } t > t_2),
\end{aligned}$$

$$\begin{aligned}
& \partial_H(S''_{d,b,t} \parallel K \parallel L \parallel R''_{1-b}) \\
&= \sigma^{t'_2} \cdot \underline{c_6(ack)} \cdot \partial_H(S''_{d,b,t-t'_2} \parallel K \parallel L' \parallel R_{1-b}) \\
& \text{(for every } t > t'_2),
\end{aligned}$$

$$\begin{aligned}
& \partial_H(S''_{d,b,t} \parallel K \parallel L' \parallel R_{1-b}) \\
&= \sigma^{t_4} \cdot \underline{c_5(ack)} \cdot \partial_H(S_{1-b} \parallel K \parallel L \parallel R_{1-b}) \\
& \quad + \sum_{k \leq t_4} \sigma^k \cdot \underline{error} \cdot \partial_H(S''_{d,b,t-k} \parallel K \parallel L \parallel R_{1-b}) \\
& \text{(for every } t > t_4),
\end{aligned}$$

$$\begin{aligned}
& \partial_H(S''_{d,b,t} \parallel K \parallel L \parallel R_b) = \sigma^t \cdot \underline{c_3(d,b)} \cdot \partial_H(S''_{d,b} \parallel K'_{d,b} \parallel L \parallel R_b) \\
& \text{(for every } t > 0),
\end{aligned}$$

$$\begin{aligned}
& \partial_H(S''_{d,b,t} \parallel K \parallel L \parallel R_{1-b}) \\
&= \sigma^t \cdot \underline{c_3(d,b)} \cdot \partial_H(S''_{d,b} \parallel K'_{d,b} \parallel L \parallel R_{1-b}) \\
& \text{(for every } t > 0),
\end{aligned}$$

$$\begin{aligned}
& \partial_H(S''_{d,b} \parallel K'_{d,b} \parallel L \parallel R_{1-b}) \\
&= \sigma^{t_3} \cdot \underline{c_4(d,b)} \cdot \partial_H(S''_{d,b,t'_1-t_3} \parallel K \parallel L \parallel R''_{1-b}) \\
& \quad + \sum_{k \leq t_3} \sigma^k \cdot \underline{error} \cdot \partial_H(S''_{d,b,t'_1-k} \parallel K \parallel L \parallel R_{1-b}).
\end{aligned}$$

If the terms on the left-hand sides of these equations include all unexpanded terms on the right-hand sides, we have that the terms on the left-hand sides make up a solution of the guarded recursive specification obtained by replacing all occurrences of these terms in the equations by occurrences of corresponding variables. It is easy to see that this is the case iff  $t'_1 > t_2 + t'_2 + t_3 + t_4$ . Hence, we derive that  $\partial_H(S_b \parallel K \parallel L \parallel R_b)$  is the solution for its corresponding variable of this guarded recursive specification. The guarded recursive specification concerned can easily be rewritten, using its equations and the axioms of process algebra, to the following sender-oriented guarded recursive specification:

$$X_b = \sum_{d \in D} r_1(d) \cdot \sigma^{t_1} \cdot Y_{d,b},$$

$$\begin{aligned}
Y_{d,b} = \underline{c_3(d,b)} \cdot \left( \sigma^{t_3} \cdot \underline{c_4(d,b)} \cdot \sigma^{t_2} \cdot \underline{s_2(d)} \cdot \sigma^{t'_2} \cdot \underline{c_6(ack)} \cdot Z_{d,b} \right. \\
\left. + \sum_{k \leq t_3} \sigma^k \cdot \underline{error} \cdot \sigma^{t'_1-k} \cdot Y_{d,b} \right),
\end{aligned}$$

$$Z_{d,b} = \sigma^{t_4} \cdot \underline{c_5(ack)} \cdot X_{1-b} + \sum_{k \leq t_4} \sigma^k \cdot \underline{error} \cdot \sigma^{t'_1-(t_2+t'_2+t_3+k)} \cdot U_{d,b},$$

$$U_{d,b} = \underline{c_3(d,b)} \cdot \left( \sigma^{t_3} \cdot \underline{c_4(d,b)} \cdot \sigma^{t'_2} \cdot \underline{c_6(ack)} \cdot V_{d,b} + \sum_{k \leq t_3} \sigma^k \cdot \underline{error} \cdot \sigma^{t'_1 - k} \cdot U_{d,b} \right),$$

$$V_{d,b} = \sigma^{t_4} \cdot \underline{c_5(ack)} \cdot X_{1-b} + \sum_{k \leq t_4} \sigma^k \cdot \underline{error} \cdot \sigma^{t'_1 - (t'_2 + t_3 + k)} \cdot U_{d,b}.$$

This recursive specification shows that, if we abstract from all actions other than the send and receive actions at the external ports 1 and 2 and in addition from the timing of actions, the whole PAR protocol is a buffer with capacity 1 as described in the previous section.

In the previous calculations, we obtained useful results by just applying the branching laws of Table 20. If we consider the PAR protocol, things become much more difficult, however. We want to hide the actions in the set

$$I = \{error, c_5(ack), c_6(ack)\} \cup \{c_i(d,b) \mid i \in \{3, 4\}, d \in D, b \in \{0, 1\}\}.$$

Now we can proceed in different ways. First of all, we can focus on functional correctness. This means that we abstract from all timing of actions by means of the time free projection operator of Sect. 6 before we abstract from internal actions. In that case, we can apply the abstraction operator in the theory without timing. Starting from the specification of  $\partial_H(S_b \parallel K \parallel L \parallel R_b)$  just given, we can easily calculate that  $\pi_{\text{tf}}(\partial_H(S_b \parallel K \parallel L \parallel R_b))$  is the solution of the guarded recursive specification that consists of the following equations:

$$\begin{aligned} X'_b &= \sum_{d \in D} r_1(d) \cdot Y'_{d,b}, \\ Y'_{d,b} &= c_3(d,b) \cdot (c_4(d,b) \cdot s_2(d) \cdot c_6(ack) \cdot Z'_{d,b} + error \cdot Y'_{d,b}), \\ Z'_{d,b} &= c_5(ack) \cdot X'_{1-b} + error \cdot U'_{d,b}, \\ U'_{d,b} &= c_3(d,b) \cdot (c_4(d,b) \cdot c_6(ack) \cdot V'_{d,b} + error \cdot U'_{d,b}), \\ V'_{d,b} &= c_5(ack) \cdot X'_{1-b} + error \cdot U'_{d,b}. \end{aligned}$$

We see immediately that  $Z'_{d,b} = V'_{d,b}$ . The branching law as given in [31] is in this case not sufficient to obtain, starting from this specification, a guarded recursive specification of  $\tau_I(\pi_{\text{tf}}(\partial_H(S_b \parallel K \parallel L \parallel R_b)))$ , as this process can get into performing cycles of silent steps, and a fair abstraction rule like KFAR [29] is needed. However, it is straightforward to exhibit a branching bisimulation between the process  $\tau_I(\pi_{\text{tf}}(\partial_H(S_b \parallel K \parallel L \parallel R_b)))$  and the buffer with capacity one recursively specified by the equation

$$B = \sum_{d \in D} r_1(d) \cdot s_2(d) \cdot B.$$

Thus, we see the PAR protocol is functionally correct. We want to stress that, in order to achieve this result, it was necessary to calculate first the time-dependent

behavior of the whole protocol, because the PAR protocol is only correct if the timing parameters are set correctly. A complete verification in process algebra without timing is not possible without resorting to artificial tricks such as excluding the premature time-out of an acknowledgement by inhibiting a time-out so long as it does not lead to deadlock (see e.g., [32]).

Next, we can have a look at the timing aspects. Starting from the specification of  $\partial_H(S_b \parallel K \parallel L \parallel R_b)$  obtained, we can easily calculate that  $\tau_I(\partial_H(S_b \parallel K \parallel L \parallel R_b))$  is the solution of the guarded recursive specification that consists of the following equations:

$$\begin{aligned} X'' &= \sum_{d \in D} r_1(d) \cdot \sigma^{t_1} \cdot Y_d'', \\ Y_d'' &= \sigma^{t_3} \cdot \underline{\tau} \cdot \sigma^{t_2} \cdot \underline{s_2}(d) \cdot \sigma^{t'_2} \cdot Z'' + \sum_{k \leq t_3} \sigma^k \cdot \underline{\tau} \cdot \sigma^{t'_1 - k} \cdot Y_d'', \\ Z'' &= \sigma^{t_4} \cdot \underline{\tau} \cdot X'' + \sum_{k \leq t_4} \sigma^k \cdot \underline{\tau} \cdot \sigma^{t'_1 - (t_2 + t'_2 + t_3 + k)} \cdot U'', \\ U'' &= \sigma^{t_3} \cdot \underline{\tau} \cdot \sigma^{t'_2} \cdot V'' + \sum_{k \leq t_3} \sigma^k \cdot \underline{\tau} \cdot \sigma^{t'_1 - k} \cdot U'', \\ V'' &= \sigma^{t_4} \cdot \underline{\tau} \cdot X'' + \sum_{k \leq t_4} \sigma^k \cdot \underline{\tau} \cdot \sigma^{t'_1 - (t'_2 + t_3 + k)} \cdot U''. \end{aligned}$$

Not many simplifications can be achieved, mainly because branching bisimulation does not allow us to leave out silent steps that occur in between delays. In effect, all internal choices made, e.g., whether or not a channel forwards a datum correctly, remain visible. More research is needed in this matter. For some initial observations concerning this matter, we refer to [33]. In this paper, a more distinguishing equivalence is investigated, which is similar to rooted branching tail bisimulation equivalence, but treats silent steps in the midst of time steps under all circumstances as redundant.

Indeed, as a result the following extra  $\tau$ -law can be applied:

$$\underline{\tau} \cdot \sigma \cdot x = \sigma \cdot \underline{\tau} \cdot x.$$

With the help of this law, the specification above can be simplified to the following:

$$\begin{aligned} A &= \sum_{d \in D} r_1(d) \cdot \sigma^{t_1 + t_2 + t_3} \cdot B_d, \\ B_d &= \underline{s_2}(d) \cdot \sigma^{t'_2} \cdot C + \sigma^{t'_1} \cdot B_d, \\ C &= \sigma^{t_4} \cdot (A + \sigma^{t'_1 - t_2 - t_4} \cdot D), \\ D &= \sigma^{t_4} \cdot (A + \sigma^{t'_1 - t_4} \cdot D). \end{aligned}$$

Based on this final specification, we can see that the protocol takes at least  $t_1 + t_2 + t_3$  time slices between consumption and delivery of a datum, and in general, between consumption and delivery we have  $t_1 + t_2 + t_3 + n \cdot t'_1$  time slices, where  $n \geq 0$ . After delivery, at least  $t'_2 + t_4$  time slices must pass before the next datum can be consumed, and in general, we have  $t'_2 + t_4$  or  $t'_2 + t_4 + m \cdot t'_1 - t_2$  time slices, where  $m > 0$ . Thus, we have a complete throughput analysis of the protocol.

## 9 Related Work

In [18], an attempt to introduce the undelayable empty process into the realm of process algebra with discrete relative timing has been described. Three design goals were formulated for the extension: (i) the undelayable empty process should be a unit element w.r.t. both sequential composition and parallel composition; (ii) commutativity and associativity of parallel composition should remain valid; (iii) the time-determinism property should be maintained.

Despite the elaborate and rather convincing discussions presented in [18] regarding the unavoidability of strange intuitions involving the undelayable empty process, we have presented such an extension that avoids the unintuitive examples described in [18] while realizing the three design goals formulated before. We contribute this success to the design decision to only capture synchronization aspects of parallel processes (such as termination and time-synchronization) in the communication merge and not in the left-merge.

In the theory of [18], the following derivations can be made (presented in the syntax used in this paper):

$$\begin{aligned}
& - (\underline{a.\underline{\epsilon}} + \underline{\epsilon}) \parallel \underline{b.\underline{\epsilon}} = \underline{a.\underline{b.\underline{\epsilon}}} + \underline{b.(\underline{a.\underline{\epsilon}} + \underline{\epsilon})} \\
& - (\underline{\sigma.\underline{a.\underline{\epsilon}}} + \underline{\epsilon}) \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} = \underline{\sigma.\underline{a.\underline{\epsilon}}} \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} \\
& - (\underline{a.\underline{\epsilon}} + \underline{\epsilon}) \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} = \underline{a.\underline{\sigma.\underline{b.\underline{\epsilon}}}} + \underline{\sigma.\underline{b.\underline{\epsilon}}}
\end{aligned}$$

In the theory presented in this paper we obtain the following equalities:

$$\begin{aligned}
& - (\underline{a.\underline{\epsilon}} + \underline{\epsilon}) \parallel \underline{b.\underline{\epsilon}} = \underline{a.(\underline{\epsilon} \parallel \underline{b.\underline{\epsilon}})} + \underline{b.(\underline{\epsilon} \parallel (\underline{a.\underline{\epsilon}} + \underline{\epsilon}))} = \underline{a.\underline{b.\underline{\epsilon}}} + \underline{b.(\underline{a.\underline{\epsilon}} + \underline{\epsilon})} \\
& - (\underline{\sigma.\underline{a.\underline{\epsilon}}} + \underline{\epsilon}) \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} = \underline{\sigma.\underline{a.\underline{\epsilon}}} \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} + \underline{\sigma.\underline{b.\underline{\epsilon}}} \\
& - (\underline{a.\underline{\epsilon}} + \underline{\epsilon}) \parallel \underline{\sigma.\underline{b.\underline{\epsilon}}} = \underline{a.\underline{\sigma.\underline{b.\underline{\epsilon}}}} + \underline{\sigma.\underline{b.\underline{\epsilon}}}
\end{aligned}$$

## 10 Conclusion

Using our theory, we can replace the delay operators of timed ACP by delay prefixing, thus simplifying axioms and calculations. We have a clear separation of action execution and termination in our operational rules. All states in transition systems correspond to process terms. In order to ensure that the axiom of weak time-determinism holds in our operational model, we use a different set of relations generating the transition systems. This avoids the use of bisimulations relating sets of states and allows to use the results that are based on the format of the SOS rules. This is an improvement in the treatment of time steps.

On the basis of the material presented here, we can extend also the other timed process algebras of the framework of [12] with explicit termination. To reach this goal, first define a variant of the present theory using absolute timing instead of relative timing. Both absolute and relative timing can be integrated using parametric timing ([9, 10]). Then, absolute-time discrete-time process algebra is a subtheory of absolute-time dense-time process algebra as in [12]. Finally, relative-time discrete-time process algebra and parametric time discrete-time process algebra can be developed.

The combination of the empty process and discrete, relative time process algebra has been studied in [18, 19]. Among the conclusions in [18, 19] are the following two: the empty process cannot be straightforwardly combined with the deadlocked process of [9, 10, 34], and the behaviour of the empty process is not always into accordance with one's first intuition. In this paper we present a combination of the empty process with standard real-time process algebra in relative timing with the deadlocked process in which the behaviour of the empty process is clear in all cases.

## References

1. Baeten, J., Bergstra, J.: Real time process algebra. *Formal Aspects of Computing* **3** (1991) 142–188
2. Hennessy, M., Regan, T.: A process algebra for timed systems. *Information and Computation* **177** (1995) 221–239
3. Moller, F., Tofts, C.: A temporal calculus of communicating systems. In Baeten, J., Klop, J., eds.: *CONCUR'90 - Theories of Concurrency: Unification and Extension*. Volume 458 of *Lecture Notes in Computer Science.*, Amsterdam, Springer-Verlag (1990) 401–415
4. Nicollin, X., Sifakis, J.: The algebra of timed processes, ATP: Theory and application. *Information and Computation* **114** (1994) 131–178
5. Quemada, J., de Frutos, D., Azcorra, A.: TIC: A TImed calculus. *Formal Aspects of Computing* **5** (1993) 224–252
6. Bos, S., Reniers, M.: The  $I^2C$ -bus in discrete-time process algebra. *Science of Computer Programming* **29** (1997) 235–258
7. Schneider, S., Davies, J., Jackson, D., Reed, G., Reed, J., Roscoe, A.: Timed CSP: Theory and practice. In de Bakker, J., Huizing, C., de Roever, W., Rozenberg, G., eds.: *Real Time: Theory and Practice*. Volume 600 of *Lecture Notes in Computer Science.*, Springer-Verlag (1991) 640–675
8. Groote, J.: The syntax and semantics of timed  $\mu$ CRL. Technical Report SEN-R9709, CWI, Amsterdam (1997)
9. Baeten, J., Bergstra, J.: Discrete time process algebra. *Formal Aspects of Computing* **8** (1996) 188–208
10. Baeten, J., Bergstra, J.: Discrete time process algebra: absolute time, relative time and parametric time. *Fundamenta Informaticae* **29** (1997) 51–76
11. Groote, J.: *Process Algebra and Structured Operational Semantics*. PhD thesis, University of Amsterdam (1991)
12. Baeten, J., Middelburg, C.: *Process Algebra with Timing*. Springer Verlag (2002)
13. Baeten, J.: Embedding untimed into timed process algebra: the case for explicit termination. *Mathematical Structures in Computer Science* **13** (2003) 589–618
14. Baeten, J., Reniers, M.: Explicit termination in timed process algebra with relative-timing. *Formal Aspects of Computing* (2004) To appear.
15. Koymans, C., Vrancken, J.: Extending process algebra with the empty process  $\varepsilon$ . Technical Report Logic Group Preprint Series 1, University Utrecht, Department of Philosophy (1985)
16. Baeten, J., Glabbeek, R.v.: Merge and termination in process algebra. In Nori, K., ed.: *Foundations of Software Technology and Theoretical Computer Science VII*. Volume 287 of *Lecture Notes in Computer Science.*, Pune, Springer-Verlag (1987) 153–172

17. Vrancken, J.: The algebra of communicating processes with empty process. *Theoretical Computer Science* **177** (1997) 287–328
18. Vereijken, J.: Discrete-time process algebra. PhD thesis, Eindhoven University of Technology (1997)
19. Baeten, J., Vereijken, J.: Discrete-time process algebra with empty process. In Bruné, M., van Deursen, A., Heering, J., eds.: *Dat is dus heel interessant*, CWI (1997) 5–24 Liber Amicorum dedicated to Paul Klint.
20. Hoare, C.: *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall International (1985)
21. Bos, V., Kleijn, J.: Formalisation of a production system modelling language: the operational semantics of  $\chi$  Core. *Fundamenta Informaticae* **41** (2000) 367–392
22. Reniers, M.: *Message Sequence Chart: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology (1999)
23. Groote, J.: Transition system specifications with negative premises. In Baeten, J., Klop, J., eds.: *CONCUR'90 - Theories of Concurrency: Unification and Extension*. Volume 458 of *Lecture Notes in Computer Science*., Amsterdam, Springer-Verlag (1990) 332–341
24. Verhoef, C.: A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal of Computing* **2** (1995) 274–302
25. Fokkink, W.: The tyft/tyxt format reduces to tree rules. In Hagiya, M., Mitchell, J., eds.: *Proceedings 2nd International Symposium in Theoretical Aspects of Computer Software (TACS'94)*. Volume 789 of *Lecture Notes in Computer Science*., Springer-Verlag (1994) 440–453
26. Baeten, J., Reniers, M.: Termination in timed process algebra. Technical Report CSR 00-13, Eindhoven University of Technology, Department of Computing Science (2000)
27. Baeten, J., Verhoef, C.: A congruence theorem for structured operational semantics with predicates. In Best, E., ed.: *CONCUR'93, International Conference on Concurrency Theory*. Volume 715 of *Lecture Notes in Computer Science*., Springer-Verlag (1993) 477–492
28. Baeten, J., Verhoef, C.: Concrete process algebra. In Abramsky, S., Gabbay, D.M., Maibaum, T., eds.: *Semantic Modelling*. Volume 4 of *Handbook of Logic in Computer Science*. Oxford University Press (1995) 149–268
29. Baeten, J., Weijland, W.: *Process Algebra*. Volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press (1990)
30. Baeten, J., Basten, T., Reniers, M.: *Algebra of Communicating Processes*. Cambridge University Press (2004) To appear.
31. van Glabbeek, R., Weijland, W.: Branching time and abstraction in bisimulation semantics. *Journal of the ACM* **43** (1996) 555–600
32. Vaandrager, F.: Two simple protocols. In Baeten, J., ed.: *Applications of process algebra*. Volume 17 of *Cambridge Tracts in Theoretical Computer Science*., Cambridge University Press (1990) 23–44
33. Baeten, J., Middelburg, C., Reniers, M.: A new equivalence for processes with timing. Technical Report CSR 02-10, Eindhoven University of Technology, Department of Computing Science (2002)
34. Baeten, J., Bergstra, J., Reniers, M.: Discrete time process algebra with silent step. In Plotkin, G., Stirling, C., Tofte, M., eds.: *Proof, Language, and Interaction: Essays in Honour of Robin Milner*. *Foundations of Computing Series*. MIT Press (2000) 535–569