Discrete time process algebra

Citation for published version (APA):

Baeten, J. C. M., & Bergstra, J. A. (1992). Discrete time process algebra. In R. Cleaveland (Ed.), *CONCUR'92 (Proceedings 3rd International Conference on Concurrency Theory, Stony Brook NY, USA, August 24-27, 1992)* (pp. 401-420). (Lecture Notes in Computer Science; Vol. 630). Springer. <https://doi.org/10.1007/BFb0084806>

DOI:

[10.1007/BFb0084806](https://doi.org/10.1007/BFb0084806)

Document status and date:

Published: 01/01/1992

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Discrete Time Process Algebra

J.C.M. Baeten

Department of Computing Science, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

J.A. Bergstra

Programming Research Group, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
and

Department of Philosophy, Utrecht University,
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

The axiom system ACP of [BEK84] is extended to ACP_{dt} , which involves discrete time delay, and then to $ACP_{dt} + ATP$, an axiomatisation that adds key features of ATP [NIS90] to ACP. We give an interpretation of all discrete time constructs in the real time theory ACP_{pv1} .

Note: This work was done in the context of ESPRIT Basic Research Action 7166, CONCUR2. The second author received partial support from RACE project 1046, SPECS. This document does not necessarily reflect the views of the SPECS consortium.

1. INTRODUCTION

Process algebra in the form of ACP [BEK84, BAW90], CCS [MIL89] or CSP [BRHR84] describes the main features of imperative concurrent programming without any explicit mention of time. Implicitly, time is present in the interpretation of sequential composition: in $p \cdot q$ (ACP notation) the process p must be executed before q . It may be felt as a weakness of the mentioned process algebra frameworks that time features implicitly only. Indeed a quantitative view on the relation between process execution and progress of time is definitely absent in these calculi. This seems worse than it actually is. Nothing prevents one to introduce timer processes that represent clocks, to have processes interacting with these clocks and to have synchronisations between the clocks in ACP, CCS or CSP.

Of course timing aspects can be described more uniformly if process algebras are given that provide standardised features to incorporate a quantitative view on time. Obvious as this may seem, the more obvious observation is that in a few years so many different formats for timed process algebras have been introduced that this uniformity seems to be further away than before. There are several reasons for the emerging spectrum of timed process algebras. We explain this by considering various options for adding time. The most obvious option is to represent time by means of non-negative reals, and to have time stamps on actions. This is done in [BAB91a] for ACP, in [RER88] for CSP and in [MOT90] for CCS. The timed versions of ACP, CSP and CCS in these papers differ concerning the degree to which time stamping is explicit in the notational format of the proposed process algebras. This in turn influences the form of equations, axiomatisations and the appearance of examples.

Another option is to divide time in slices indexed by natural numbers, to have an implicit or explicit time stamping mechanism that provides each action with the time slice in which it occurs and to have a time order within each slice only. This has been worked out in ATP [NIS90], a process algebra that adds time slicing to a version of ACP based on action prefixing rather than sequential composition (this version was called $aprACP$ in [BAB91b]). Further, [GRO90a] has extended ACP with time slices and [MOT89] have added these features to CCS. We propose to use the phrase *real time process algebra* for algebras that involve explicit or implicit time stamping with real numbers (or any dense subfield of the reals) and *discrete time process algebra* if an enumeration of time slices is used.

The objective of this paper is to extend ACP to a discrete time process algebra. This is done by adding a new process constructor $\sigma_d: P \rightarrow P$. The notation σ has been taken from [HER90], the subscript d draws attention to the discrete time setting. $\sigma_d(x)$ delays the process x till the next time slice. In addition to σ_d four auxiliary functions and an auxiliary sort are used. The booleans serve as auxiliary sort and the conditional operator $\triangleleft _ \triangleright _ : P \times B \times P \rightarrow P$ is used. Further, an operator $1 \gg_d _ : P \rightarrow P$ (unit time shift) reduces a process x to those options left if it idles one step (and to δ if it cannot idle). Then, the operator $\gg_d 1 _ : P \rightarrow P$ restricts a process x to those options that do not involve initial idling. Finally, $D: P \rightarrow B$ determines whether or not a process can idle at all.

For practical application, the primitives of ACP_{dt} are too low level. So we have added the key operators of ATP [NSVR90, NIS90] to ACP_{dt} , thus obtaining $ACP_{dt} + ATP$. These features include: unit delay $[x](y)$ and maximal progress composition $\oplus$. It turns out that these operators can be eliminated in the presence of those of ACP_{dt} . (It should be noted that ATP's $\oplus$ is already present as $+$ in TCCS.) As ACP_{dt} , ATP and TCCS each use strong bisimulation equivalence to obtain a semantic domain, they may be considered as different (and intertranslatable) sets of generators for the same semantic world of processes. Some less important modifications with respect to ATP in the form of [NIS90] have been made. $ACP_{dt} + ATP$ contains time stops (as does [MOT89]), a feature that is rejected on philosophical grounds in [NIS90]. In addition, the cancel mechanism has been omitted because that is covered automatically due to the presence of proper termination in ACP_{dt} .

Both the selection of $ACP_{dt} + ATP$ and the casting of ATP in an ACP framework require some motivation. We have the following remarks on this.

1. ATP contains a combination of features (time outs, time posteriority, mode transfers) which turn it into a nontrivial extension of the untimed (symbolic) case. The features of ATP turn out to be of significant expressive power in comparison to [GRO90a], [GM91]. Moreover we found these features quite pleasant when dealing with examples.
2. The claim of [NIS90] that progress of time by itself should not be allowed to introduce non-determinism is convincing in our view. If time is represented by an action σ then $\sigma \cdot (x + y) = \sigma \cdot x + \sigma \cdot y$ is an appropriate axiom called time factorisation in [NIS90] ([GRO90a], [GM91] do not satisfy time factorisation).
3. ATP does not involve an action that represents the flow of time. This is a conceptual advantage as intuitively, flow of time is not an action, and the progress of an independent clock is not an observable system action.
4. Building ATP on top of ACP has several advantages. First of all, both termination and general sequential composition (which are in ACP but not in ATP) combine very well with ATP's key features. We hope that this is sufficiently illustrated by the protocol verifications in section 5. Secondly, the meta theory of ACP + ATP can be organised just as the meta theory of ACP (see e.g. [BAW90]) which allows to simply copy most of the results and proof techniques.

We design our discrete time process algebra in an incremental way, through stages BPA_{dt} , $BPA_{\delta dt}$, $PA_{\delta dt}$, ACP_{dt} , extensions of ACP_{dt} . This resembles the segmentation of ACP that can be found in [BAW90]. At each step we will motivate our choice and compare it with the sources of discrete time process algebra that we have used: TCCS [MOT89], ACP_1 [GRO90a], TPL [HER90], ATP [NIS90], CCSS [GM91]. Some of the axioms we use already appear in [BAB91b].

2. REAL TIME PROCESS ALGEBRA WITH INITIAL ABSTRACTION OPERATOR

This paper intends to present discrete time process algebra in a clear connection with real time process algebra. To this end, we will first outline the axiom system $ACPP\sqrt{I}$. It describes the real time process algebra of [BAB91a] while integrating absolute and relative time notation using the initial abstraction operator $\sqrt{}$ that was introduced in [BAB92]. For $ACPP\sqrt{I}$, a bisimulation model M_A^* is outlined (here, A denotes the underlying set of atomic actions). This model is introduced in a sketchy way. Its relevance lies in the clear operational meaning it provides for processes. It serves as a conceptual standard model.

Having available M_A^* , the various operators and constants of discrete time process algebra are defined in it. Then, as a subject, discrete time process algebra reduces to an investigation of certain reducts and subalgebras of M_A^* as well as their axiomatic description.

We start with a quick overview of the real time process algebra introduced in [BAB91a]. We follow the presentation of [BAB92]. We also use the initial abstraction operator introduced in [BAB92], which allows to present the absolute time and relative time versions of the theory of [BAB91a] in a unified framework. We give an operational semantics in the style of [BAB91a].

2.1 ATOMIC ACTIONS

We start from a set A of (symbolic) atomic actions. The set A is a parameter of the theory. Further, we have a special constant δ , denoting inaction, the absence of any execution. In particular, δ cannot terminate. We put $A_\delta = A \cup \{\delta\}$. The set of atomic actions with time parameter, AT is

$$AT = \{a(t) \mid a \in A_\delta, t \in \mathbb{R}_{\geq 0}\}.$$

In [BAB91a], we considered the use of locations, in order to express that actions can occur at the same time but at different locations. This led to the introduction of *multi-actions*. In [BAB92], we extended this concept by looking at actions parametrised by a point in real space. As we will not need multi-actions in discrete time process algebra (since within a time slice, $a \parallel b = a \cdot b + b \cdot a + a \mid b$, i.e. interleaving works), we will not consider multi-actions in this paper. The theory can be set up for multi-actions, just like we did in [BAB92], without any problems.

2.2 BASIC PROCESS ALGEBRA

Process algebra (see [BEK84, BAW90]) starts from a given *action alphabet*, here AT. Elements of AT are constants of the sort P of *processes*. The process $a(t)$ can let time progress until t , will then execute action a at time t , and then terminate successfully. The process $\delta(t)$ can also let time progress until t , but then nothing more is possible (in particular, time cannot progress anymore). Therefore, the δ actions are often called *time stops*.

Real Time Basic Process Algebra with Deadlock (BPAP δ) has two binary operators $+, \cdot$: $P \times P \rightarrow P$; $+$ stands for alternative composition and $\cdot$ for sequential composition. Moreover, there is the additional operator $\gg: \mathbb{R}_{\geq 0} \times P \rightarrow P$, the *initialisation operator*. This operator was called the time shift

$X + Y = Y + X$	A1	$a \mid b = b \mid a$	C1
$(X + Y) + Z = X + (Y + Z)$	A2	$a \mid (b \mid c) = (a \mid b) \mid c$	C2
$X + X = X$	A3	$\delta \mid a = \delta$	C3
$(X + Y) \cdot Z = X \cdot Z + Y \cdot Z$	A4	$t \neq s \Rightarrow a(t) \mid b(s) = \delta(\min(t, s))$	ATC1
$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	A5	$a(t) \mid b(t) = (a \mid b)(t)$	ATC2
$X + \delta = X$	A6		
$\delta \cdot X = \delta$	A7	$X \parallel Y = X \parallel Y + Y \parallel X + X \mid Y$	CM1
$a(0) = \delta(0) = \delta$	ATA1	$X = r \gg Z \Rightarrow a(t) \parallel X = (a(t) \gg U(X)) \cdot X$	ATCM2'
$\delta(t) \cdot X = \delta(t)$	ATA2	$Y = r \gg Z \Rightarrow a(t) \cdot X \parallel Y = (a(t) \gg U(Y)) \cdot (X \parallel Y)$	ATCM3'
$t < r \Rightarrow \delta(t) + \delta(r) = \delta(r)$	ATA3	$(X + Y) \parallel Z = X \parallel Z + Y \parallel Z$	CM4
$a(t) + \delta(t) = a(t)$	ATA4	$(a(t) \cdot X) \mid b(r) = (a(t) \mid b(r)) \cdot X$	CM5'
$a(t) \cdot X = a(t) \cdot (t \gg X)$	ATA5	$a(t) \mid (b(r) \cdot X) = (a(t) \mid b(r)) \cdot X$	CM6'
		$(a(t) \cdot X) \mid (b(r) \cdot Y) = (a(t) \mid b(r)) \cdot (X \parallel Y)$	CM7'
$t < r \Rightarrow t \gg a(r) = a(r)$	ATB1	$(X + Y) \mid Z = X \mid Z + Y \mid Z$	CM8
$t \geq r \Rightarrow t \gg a(r) = \delta(t)$	ATB2	$X \mid (Y + Z) = X \mid Y + X \mid Z$	CM9
$t \gg (X + Y) = (t \gg X) + (t \gg Y)$	ATB3		
$t \gg (X \cdot Y) = (t \gg X) \cdot Y$	ATB4	$\partial_H(a) = a$ if $a \notin H$	D1
$r \geq t \Rightarrow a(r) \gg t = \delta(t)$	ATB5	$\partial_H(a) = \delta$ if $a \in H$	D2
$r < t \Rightarrow a(r) \gg t = a(r)$	ATB6	$\partial_H(a(t)) = (\partial_H(a))(t)$	ATD
$(X + Y) \gg t = (X \gg t) + (Y \gg t)$	ATB7	$\partial_H(X + Y) = \partial_H(X) + \partial_H(Y)$	D3
$(X \cdot Y) \gg t = (X \gg t) \cdot Y$	ATB8	$\partial_H(X \cdot Y) = \partial_H(X) \cdot \partial_H(Y)$	D4
$U(a(t)) = t$	ATU1		
$U(X + Y) = \max\{U(X), U(Y)\}$	ATU3		
$U(X \cdot Y) = U(X)$	ATU4		

TABLE 1. ACP ρ .

operator in [BAB91a]. $t \gg X$ denotes the process X starting at time t . This means that all actions that have to be performed at or before time t are turned into deadlocks because their execution has been delayed too long. $\text{BPAP}\delta$ has the axioms A1-7 and ATA1-5, ATB1-4 from table 1 ($a \in A_\delta$).

The letter A in the names of the axioms in table 5 refers to *absolute time* (versions with relative time will be considered in the following). Next we consider parallel composition. Notice that we do not have a counterpart of PA ([BEK84]) in real time process algebra, because merge without communication is not an option here. Indeed, even if a and b do not communicate, the term $a(2) \parallel b(2)$ cannot be evaluated as $a(2) \cdot b(2) + b(2) \cdot a(2)$. Thus, we consider merge with communication straight away.

2.3 ALGEBRA OF COMMUNICATING PROCESSES

In order to formulate communication between processes, we assume we have given a *communication function* $!$ on A_δ . This function $! : A_\delta \times A_\delta \rightarrow A_\delta$ is also a parameter of the theory, and should be commutative, associative and have δ as a neutral element (see axioms C1,2,3). An axiomatization of parallel composition with communication uses the left merge operator $\llcorner$, the communication merge operator $!$, and the encapsulation operator ∂_H of [BEK84]. Moreover, two extra auxiliary operators introduced in [BAB91a] are needed: the ultimate delay operator and the bounded initialization operator.

The *ultimate delay* operator U takes a process expression X , and returns an element of $\mathbb{R}_{\geq 0}$. The intended meaning is that X can idle before $U(X)$, but X can never reach time $U(X)$ or a later time by just idling. The *bounded initialization* operator (or *time out* operator) is also denoted by $\gg$, and is the counterpart of the operator with the same name that we saw in the axiomatization of $\text{BPAP}\delta$. With $X \gg t$ we denote the process X with its behaviour restricted to the extent that its first action must be performed at a time before $t \in \mathbb{R}_{\geq 0}$. The axioms of ACPP are in table 1. There, $H \subseteq A$, $a, b, c \in A_\delta$.

Note that in [BAB91a], axioms ATCM2, ATCM3 appear that coincide with the axioms ATCM2', ATCM3' above except for the absence of the conditions. Because in the setting of ACPP there, we have absolute time notation only, each process X satisfies $X = 0 \gg X$. We have that in the context of ACPP , the new axioms are slightly weaker than the old, unprimed ones, but are in fact equivalent on finite process expressions. We have to add the conditions in order to deal with the following extension to processes that can also involve relative time notation. Notice that by substitution of $0 \gg Z$ for X resp. Y both conditional equations are turned into ordinary equations.

2.4 RELATIVE TIME NOTATION

The following is based on [BAB92]. In some cases, relative time notation is profitable. In [BAB91a], we have introduced a relative time notation as follows: $a[t]$ denotes action a t time units after the previous atomic action or, if such action doesn't exist, after system initialisation. In this paper, we will deal with relative time in a different way. The basis for this approach to relative time notation is the *initial abstraction operator*. Let for $t \in \mathbb{R}_{\geq 0}$, $F(t)$ be a process in P , then $\sqrt{t}.F(t)$ denotes a process that, when started at time r , proceeds as $F(r)$. Semantically, $\sqrt{t}.F(t)$ is just a function f from $\mathbb{R}_{\geq 0}$ into P that satisfies $f(t) = t \gg f(t)$ for all $t \geq 0$. We make things more precise in the following.

2.5 DEFINITION

Let P be the sort of processes. Put $P^* = \{f : \mathbb{R}_{\geq 0} \rightarrow P \text{ such that } f(t) = t \gg f(t)\}$.

We introduce the initial abstraction operator $\sqrt{\cdot} : T \times P^* \rightarrow P^*$, where T is a set of variables ranging over $\mathbb{R}_{\geq 0}$ called *time variables*. In the expression $\sqrt{t}.F$, free occurrences of t in F become bound.

2.6 AXIOMS

All axioms of real time process algebra with integration presented before remain valid on the extended domain. On top of the old axioms, we have the extra axioms in table 2, governing the use of initial abstraction. We intend to axiomatise in such a way that all finite process expressions can be written in a

form $\sqrt{t}.X$, with X not containing initial abstraction. In all axioms in table 2, it is assumed that t is not free in G . Axiom IA0 is the definition of the square bracket notation, axiom IA1 gives α -conversion, axiom IA2 achieves β -conversion, i.e. function application. IA4 gives the embedding of absolute time notation processes in the extended domain. IA5 is an extensionality axiom. We call the theory with axioms in tables 1-2 ACPp $\sqrt{\cdot}$.

$a[r] = \sqrt{t}. a(t + r)$	IA0	$(\sqrt{t}.F) + G = \sqrt{t}.(F + t \gg G)$	IA6
$\sqrt{s}.G = \sqrt{t}.G[\sqrt{s}]$	IA1	$(\sqrt{t}.F) \cdot G = \sqrt{t}.(F \cdot G)$	IA7
$s \gg \sqrt{t}.F = s \gg F[s/t]$	IA2	$(\sqrt{t}.F) \parallel G = \sqrt{t}.(F \parallel t \gg G)$	IA8
$\sqrt{t}.\sqrt{r}.F = \sqrt{t}.F[t/r]$	IA3	$G \parallel (\sqrt{t}.F) = \sqrt{t}.(t \gg G \parallel F)$	IA9
$G = \sqrt{t}.G$	IA4	$(\sqrt{t}.F) \mid G = \sqrt{t}.(F \mid t \gg G)$	IA10
$\forall t \geq 0 \ t \gg X = t \gg Y \Rightarrow X = Y$	IA5	$G \mid (\sqrt{t}.F) = \sqrt{t}.(t \gg G \mid F)$	IA11
		$\partial_H(\sqrt{t}.F) = \sqrt{t}.\partial_H(F)$	IA12
		$(\sqrt{t}.F) \gg r = \sqrt{t}.(F \gg r)$	IA13
		$U(F) = U(0 \gg F)$	IA14

TABLE 2. Axioms for initial abstraction.

2.7 EXAMPLES

1. $a(3) \cdot (\sqrt{t}.\sqrt{r}.a(t + r + 1)) = a(3) \cdot (3 \gg \sqrt{t}.\sqrt{r}.a(t + r + 1)) = a(3) \cdot (3 \gg \sqrt{r}.a(r + 4)) = a(3) \cdot (3 \gg a(7)) = a(3) \cdot a(7)$.
2. $a(t) \cdot b[r] = a(t) \cdot (\sqrt{s}.b(r + s)) = a(t) \cdot (t \gg \sqrt{s}.b(r + s)) = a(t) \cdot (t \gg b(r + t)) = a(t) \cdot b(r + t)$.
3. $a(t) \cdot \partial_H(b[r] \cdot X \parallel Y) = a(t) \cdot \partial_H(b(t+r) \cdot X \parallel Y)$.
4. For all closed terms $s \gg s \gg X = s \gg X$. From this follows: $\sqrt{t}.F = \sqrt{t}.(t \gg F)$.

2.8 RIGIDITY

We call a process X *rigid* iff $X = 0 \gg X$. The rigid processes in P^* correspond to the objects of P , now embedded into P^* by $p \mapsto \lambda t. t \gg p$.

On rigid processes, the axioms ATCM2', ATCM3' reduce to their unprimed versions in ACPp.

2.9 BOOLEANS

A useful feature in the language is the addition of *conditionals* or *guards*. We follow the presentation of [BAB91c]. We use the auxiliary sort **BOOL** with constants *true*, *false* and add the ternary operator

$$\triangleleft, \triangleright : P^* \times \text{BOOL} \times P^* \rightarrow \text{BOOL}.$$

Here, $x \triangleleft b \triangleright y$ should be read as: if b then x else y . On occasion, we also use this operator without the third component, the *guarded command* defined by $b \triangleright x = x \triangleleft b \triangleright \delta$.

Axioms for these operators are straightforward (table 3). In the case of expressions involving the initial abstraction operator, we will also encounter Boolean expressions parametrised by time variables, so we turn the sort **BOOL** into a function domain: $\text{BOOL}^* = \{f : \mathbb{R}_{\geq 0} \rightarrow \text{BOOL}\}$, and we have the abstraction operator $\sqrt{\cdot} : T \times \text{BOOL}^* \rightarrow \text{BOOL}^*$. Boolean functions $\wedge, \vee, \neg$ are defined on BOOL^* pointwise.

The signature has sort BOOL^* , constants *true*, *false* and functions $\neg : \text{BOOL}^* \rightarrow \text{BOOL}^*$, $\wedge, \vee : \text{BOOL}^* \times \text{BOOL}^* \rightarrow \text{BOOL}^*$, $<, >, \leq, \geq, = : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightarrow \text{BOOL}^*$, $\gg : \mathbb{R}_{\geq 0} \times \text{BOOL}^* \rightarrow \text{BOOL}^*$.

$\neg \text{true} = \text{false}$	$\neg \text{false} = \text{true}$
$b \wedge \text{false} = \text{false}$	$b \vee \text{false} = b$
$b \wedge \text{true} = b$	$b \vee \text{true} = \text{true}$
$b \wedge c = c \wedge b$	$b \vee c = c \vee b$
$t \gg \text{true} = \text{true}$	$t \gg \text{false} = \text{false}$
$t \gg \sqrt{r}.b = t \gg b[t/r]$	
$\neg(\sqrt{t}.b) = \sqrt{t}.\neg b$	$(\sqrt{t}.b) \wedge c = \sqrt{t}.(b \wedge (t \gg c))$
$(\sqrt{t}.b) \vee c = \sqrt{t}.(b \vee (t \gg c))$	
$x \triangleleft \text{true} \triangleright y = x$	$x \triangleleft \text{false} \triangleright y = y$
$b \rightarrow x = x \triangleleft b \triangleright \delta$	$x \triangleleft b \triangleright y = (b \rightarrow x) + (\neg b \rightarrow y)$
$(b \vee c) \rightarrow x = (b \rightarrow x) + (c \rightarrow x)$	
$b \rightarrow (c \rightarrow x) = (b \wedge c) \rightarrow x$	
$x \triangleleft \sqrt{t}.b \triangleright y = \sqrt{t}.(t \gg x \triangleleft b \triangleright t \gg y)$	
$t \gg (x \triangleleft b \triangleright y) = t \gg x \triangleleft t \gg b \triangleright t \gg y$	

TABLE 3. Axioms for conditionals.

2.10 EXAMPLE

With the help of Boolean expressions, we can find normal forms for all closed process expressions of the form $\sqrt{t}.P$, with P using atomic actions, $+$, $\cdot$ and conditionals only. Thus, the operators $\parallel, \mathbb{L}, \mathbb{I}$ can be eliminated. We just give two examples to illustrate this:

1. $a(3) = \sqrt{t}.(t < 3) \rightarrow a(3)$.
2. $a(3) \parallel b[2] = \sqrt{t}.(a(3) \parallel b(t+2)) = \sqrt{t}.t \gg (a(3) \parallel b(t+2)) = \sqrt{t}((t < 1) \rightarrow (b(t+2) \cdot a(3)) + (t = 1) \rightarrow (a \mid b)(3) + (t > 1) \rightarrow a(3) \cdot b(t+2))$

2.11 INTEGRATION

An extension of ACPPv (called ACPPvI) that is very useful in applications is the extension with the integral operator, denoting a choice over a continuum of alternatives. That is, if V is a subset of $\mathbb{R}_{\geq 0}$, and v is a variable over $\mathbb{R}_{\geq 0}$, then $\int_{v \in V} P$ denotes the alternative composition of alternatives $P[t/v]$ for $t \in V$ (expression P with nonnegative real t substituted for variable v). In this expression, a free occurrence of v in P becomes bound. Alpha conversion must be allowed to avoid name clashes. For more information, we refer the reader to [BAB91a] and [KLU91].

Of course, we cannot give a complete axiomatisation of general integration because the general theory is undecidable, but the axioms in [BAB91a] are very useful in derivations (for a complete axiomatisation of a subtheory, see [KLU91]).

In closed process expressions with a restricted form of integration, we can eliminate the operators $\parallel, \mathbb{L}, \mathbb{I}, \partial_H, \gg, \gg$. [FOK92] explains in detail how such an elimination theorem can be obtained formally. They use the special case of prefixed integration. Generalisation of their results to a setting with initial abstraction seems unproblematic. This leads to normal forms of the form $\sqrt{t}.P$ with P using $a(t), \delta(t), +, \cdot$, conditionals and prefixed integration.

2.12 SEMANTICS

Suppose we have an unspecified set of *states* S of cardinality $\leq 2^{\aleph_0}$ (when we use structured operational semantics further on, S will consist of closed process expressions plus the special symbol

$\sqrt{}$. S contains a special state $\sqrt{}$ (the terminated state). A *real time transition system* over a set of actions A (not including δ) has domain $S \times \mathbb{R}_{\geq 0}$, has two relations on this domain:

$\text{idle} \subseteq S \times \mathbb{R}_{\geq 0} \times S \times \mathbb{R}_{\geq 0}$, notation $\langle s, t \rangle \rightarrow \langle s', t' \rangle$

$\text{step} \subseteq S \times \mathbb{R}_{\geq 0} \times A \times \mathbb{R}_{\geq 0} \times S \times \mathbb{R}_{\geq 0}$, notation $\langle s, t \rangle \xrightarrow{a(r)} \langle s', t' \rangle$,

and satisfies the following requirements:

0. there is a *root* $\langle s, 0 \rangle$ for some $s \in S$; no transition starts from $\langle s', 0 \rangle$ when $s' \neq s$

1. if $\langle s, t \rangle \rightarrow \langle s', t' \rangle$, then $t < t'$

2. if $\langle s, t \rangle \xrightarrow{a(r)} \langle s', t' \rangle$, then $t < r = t'$

3. if $\langle s, t \rangle \rightarrow \langle s', t' \rangle$ and $t < t^* < t'$, then there is $s^* \in S$ with $\langle s, t \rangle \rightarrow \langle s^*, t^* \rangle \rightarrow \langle s', t' \rangle$

4. if $\langle s, t \rangle \xrightarrow{a(r)} \langle s', t' \rangle$, and $t < t^* < r$, then there is $s^* \in S$ with $\langle s, t \rangle \rightarrow \langle s^*, t^* \rangle \xrightarrow{a(r)} \langle s', t' \rangle$

5. if $\langle s, t \rangle \rightarrow \langle s^*, t^* \rangle \rightarrow \langle s', t' \rangle$, then $\langle s, t \rangle \rightarrow \langle s', t' \rangle$

6. if $\langle s, t \rangle \rightarrow \langle s^*, t^* \rangle \xrightarrow{a(r)} \langle s', t' \rangle$, then $\langle s, t \rangle \xrightarrow{a(r)} \langle s', t' \rangle$

7. no transition starts from a state $\langle \sqrt{}, t \rangle$.

Let us call the set of such transition systems RTTS .

In order to deal with initial abstraction, we have to extend this set to the function space $\text{RTTS}^* = \{f : \mathbb{R}_{\geq 0} \rightarrow \text{RTTS}\}$. We can call elements of RTTS^* *type 1 transition systems*.

2.13 STRUCTURED OPERATIONAL SEMANTICS

Now we interpret the constants and operators of $\text{ACPP}/\sqrt{}$ in the semantic domain of type 1 transition systems. We define the mapping $I^* : P^* \rightarrow \text{RTTS}^*$ by means of a mapping $I : P \rightarrow \text{RTTS}$. We have the identity $I^*(F) = \lambda t. I(t \gg F)$ (t not free in F),

so it remains to define I . The simplest way to define I is by means of structured operational semantics. As set of states we can take the set of closed process expressions plus $\sqrt{}$, the root of $I(x)$ is $\langle x, 0 \rangle$ and the transitions are exactly those that can be derived on the basis of the rules in the following table 4. On the basis of these rules, the operators can also be directly defined on real time transition systems in a straightforward way. We have $t, s, r \in \mathbb{R}_{\geq 0}$, $x, x', y, y' \in S - \{\sqrt{}$, $a, b, c \in A$.

$t < r \Rightarrow \langle a(r), t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle$		
$t < s < r \Rightarrow \langle a(r), t \rangle \rightarrow \langle a(r), s \rangle$	$t < s < r \Rightarrow \langle \delta(r), t \rangle \rightarrow \langle \delta(r), s \rangle$	
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}{\langle x+y, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, \langle y+x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}{\langle x+y, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle, \langle y+x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}$		
$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle}{\langle x+y, t \rangle \rightarrow \langle x+y, r \rangle, \langle y+x, t \rangle \rightarrow \langle y+x, r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}{\langle x \cdot y, t \rangle \xrightarrow{a(r)} \langle x' \cdot y, r \rangle}$	$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle}{\langle x \cdot y, t \rangle \rightarrow \langle x \cdot y, r \rangle}$	$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}{\langle x \cdot y, t \rangle \xrightarrow{a(r)} \langle y, r \rangle}$
$t < r < s \Rightarrow \langle s \gg x, t \rangle \rightarrow \langle s \gg x, r \rangle$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, r > s}{\langle s \gg x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}$	$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle}{\langle s \gg x, t \rangle \rightarrow \langle s \gg x, r \rangle}$	$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle, r > s}{\langle s \gg x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}$
$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle, r < s}{\langle x \gg s, t \rangle \rightarrow \langle x \gg s, r \rangle}$	$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, r < s}{\langle x \gg s, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}$	$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle, r < s}{\langle x \gg s, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}$
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, a \notin H}{\langle \partial_H(x), t \rangle \xrightarrow{a(r)} \langle \partial_H(x'), r \rangle}$	$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle}{\langle \partial_H(x), t \rangle \rightarrow \langle \partial_H(x), r \rangle}$	$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle, a \notin H}{\langle \partial_H(x), t \rangle \xrightarrow{a(r)} \langle \sqrt{}, r \rangle}$

$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, \langle y, t \rangle \rightarrow \langle y, r \rangle}{\langle x \parallel y, t \rangle \xrightarrow{a(r)} \langle x' \parallel y, r \rangle, \langle y \parallel x, t \rangle \xrightarrow{a(r)} \langle y \parallel x', r \rangle, \langle x \mathbb{L} y, t \rangle \xrightarrow{a(r)} \langle x' \mathbb{L} y, r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle, \langle y, t \rangle \rightarrow \langle y, r \rangle}{\langle x \parallel y, t \rangle \xrightarrow{a(r)} \langle y, r \rangle, \langle y \parallel x, t \rangle \xrightarrow{a(r)} \langle y, r \rangle, \langle x \mathbb{L} y, t \rangle \xrightarrow{a(r)} \langle y, r \rangle}$		
$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle, \langle y, t \rangle \rightarrow \langle y, r \rangle}{\langle x \parallel y, t \rangle \rightarrow \langle x \parallel y, r \rangle, \langle x \mathbb{L} y, t \rangle \rightarrow \langle x \mathbb{L} y, r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, \langle y, t \rangle \xrightarrow{b(r)} \langle y', r \rangle, a \mid b = c \neq \delta}{\langle x \parallel y, t \rangle \xrightarrow{c(r)} \langle x' \parallel y', r \rangle, \langle x \mid y, t \rangle \xrightarrow{c(r)} \langle x' \mid y', r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, \langle y, t \rangle \xrightarrow{b(r)} \langle \sqrt{\cdot}, r \rangle, a \mid b = c \neq \delta}{\langle x \parallel y, t \rangle \xrightarrow{c(r)} \langle x', r \rangle, \langle y \parallel x, t \rangle \xrightarrow{c(r)} \langle x', r \rangle, \langle x \mid y, t \rangle \xrightarrow{c(r)} \langle x', r \rangle, \langle y \mid x, t \rangle \xrightarrow{c(r)} \langle x', r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle, \langle y, t \rangle \xrightarrow{b(r)} \langle \sqrt{\cdot}, r \rangle, a \mid b = c \neq \delta}{\langle x \parallel y, t \rangle \xrightarrow{c(r)} \langle \sqrt{\cdot}, r \rangle, \langle x \mid y, t \rangle \xrightarrow{c(r)} \langle \sqrt{\cdot}, r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}{\langle x \triangleleft \text{true} \triangleright y, t \rangle \xrightarrow{a(r)} \langle x', r \rangle, \langle y \triangleleft \text{false} \triangleright x, t \rangle \xrightarrow{a(r)} \langle x', r \rangle}$		
$\frac{\langle x, t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle}{\langle x \triangleleft \text{true} \triangleright y, t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle, \langle y \triangleleft \text{false} \triangleright x, t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle}$		
$\frac{\langle x, t \rangle \rightarrow \langle x, r \rangle}{\langle x \triangleleft \text{true} \triangleright y, t \rangle \rightarrow \langle x \triangleleft \text{true} \triangleright y, r \rangle, \langle y \triangleleft \text{false} \triangleright x, t \rangle \rightarrow \langle y \triangleleft \text{false} \triangleright x, r \rangle}$		
$\frac{\langle x(u), t \rangle \rightarrow \langle x(u), r \rangle, u \in V}{\langle \int_{v \in V} x(v), t \rangle \rightarrow \langle \int_{v \in V} x(v), r \rangle} \quad \frac{\langle x(u), t \rangle \xrightarrow{a(r)} \langle x', r \rangle, u \in V}{\langle \int_{v \in V} x(v), t \rangle \xrightarrow{a(r)} \langle x', r \rangle} \quad \frac{\langle x(u), t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle, u \in V}{\langle \int_{v \in V} x(v), t \rangle \xrightarrow{a(r)} \langle \sqrt{\cdot}, r \rangle}$		

TABLE 4. Structured operational semantics.

All that remains is to define the ultimate delay operator U on RTTS^* . If $f: \mathbb{R}_{\geq 0} \rightarrow \text{RTTS}$, then $U(f) = U(f(0))$, and if $R \in \text{RTTS}$, then $U(R) = \sup\{t \in \mathbb{R}_{\geq 0} : \text{there is } s \in S \text{ with } \text{root}(R) \rightarrow \langle s, t \rangle\}$.

2.14 BISIMULATIONS

A *bisimulation* on RTTS is a binary relation R on $S \times \mathbb{R}_{\geq 0}$ such that ($a \in A$):

- whenever $R(\langle s, t \rangle, \langle s', t' \rangle)$ then $t = t'$
- for each p and q with $R(p, q)$: if there is a step $a(t)$ possible from p to p' , then there is $q' \in S \times \mathbb{R}_{\geq 0}$ such that $R(p', q')$ and there is a step $a(t)$ possible from q to q' .
- for each p and q with $R(p, q)$: if there is a step $a(t)$ possible from q to q' , then there is $p' \in S \times \mathbb{R}_{\geq 0}$ such that $R(p', q')$ and there is a step $a(t)$ possible from p to p' .

- iv. for each p and q with $R(p, q)$: a termination step $a(t)$ to $\langle \sqrt{t}, t \rangle$ is possible from p iff it is possible from q .
- v. for each p and q with $R(p, q)$: if there is an idle step possible from p to p' , then there is $q' \in S \times \mathbb{R}_{\geq 0}$ such that $R(p', q')$ and there is an idle step possible from q to q' .
- vi. for each p and q with $R(p, q)$: if there is an idle step possible from q to q' , then there is $p' \in S \times \mathbb{R}_{\geq 0}$ such that $R(p', q')$ and there is an idle step possible from p to p' .

We say expressions p and q are *bisimilar*, denoted $p \Leftrightarrow q$, if there exists a bisimulation with $R(p, q)$. We can show that bisimulation is a congruence relation on real time transition systems, and that real time transition systems modulo bisimulation determine a model for ACPpI. This model also contains solutions of recursive process definitions, but that is an issue that is outside the scope of the present paper. We obtain a model of ACPpI that we will call M_A .

This definition of bisimulation naturally extends to a definition on type 1 real time transition systems: two elements $f, g \in \text{RTTS}^*$ are bisimilar if for all $t \geq 0$, $f(t)$ and $g(t)$ are bisimilar. This gives us a model M_A^* of type 1 real time transition systems modulo bisimulation.

2.15 STANDARD INITIALISATION AXIOMS

We present some axioms that are useful in the calculations to come, that hold in the model we described, and that can be derived from the theory for all closed process expressions. Thus, these axioms have the same status as the so-called *Standard Concurrency* axioms of [BEK84].

$t \leq r \Rightarrow t \gg (x \gg r) = (t \gg x) \gg r$	$t \geq r \Rightarrow t \gg (x \gg r) = \delta(t)$
$t \leq r \Rightarrow t \gg (r \gg x) = r \gg x$	$t \geq r \Rightarrow (t \gg x) \gg r = \delta(r)$

TABLE 5. Standard initialisation axioms.

3. THE SYNTAX OF DISCRETE TIME PROCESS ALGEBRA

3.1 SIGNATURE

The signature of discrete time process algebra provides constants and operators parametrised by natural numbers rather than real numbers. We add the following ingredients to real time process algebra.

- constants

$a(n)$	a in the n -th time slice ($a \in A, n \in \mathbb{N}$)
$\underline{a}$	a in the current time slice ($= a[1]$)
$\underline{a}[n]$	a in the $(n-1)$ -st time slice after the current one ($n \in \mathbb{N}$)
$\delta(n)$	time stop at the end of the n -th time slice ($n \in \mathbb{N}$)
δ	δ at the end of the current time slice ($= \delta[1]$)
$\delta[n]$	δ at the end of the $(n-1)$ -st time slice after the current one ($n \in \mathbb{N}$)
$\underline{a}$	delayable a ([BAB91a])
$\underline{\delta}$	livelock ([BAB91a])
$\underline{T}$	discrete time step ([GRO90a])
- functions

$\gg_d 1: P^* \rightarrow P^*$	time out in the current time slice
$1 \gg_d: P^* \rightarrow P^*$	initialisation in the following time slice
$\sigma_d: P^* \rightarrow P^*$	discrete time unit delay
$\sigma^t: P^* \rightarrow P^*$	real time delay by time $t \in \mathbb{R}_{\geq 0}$ ([MOT89])
$D: P^* \rightarrow \text{BOOL}^*$	delayability
$\sqrt{d}: \mathbb{N} \times P^* \rightarrow P^*$	discrete time initialisation ([HER90])
$\oplus: P^* \times P^* \rightarrow P^*$	strong choice
$[\cdot, \cdot](\cdot): P^* \times P^* \rightarrow P^*$	unit delay

$\lfloor \cdot \rfloor^n(\cdot): P^* \times P^* \rightarrow P^*$	start delay within n ($n \in \mathbb{N}$)
$\lfloor \cdot \rfloor^\omega: P^* \rightarrow P^*$	unbounded start delay
$\lfloor \cdot \rfloor(\cdot): P^* \times P^* \rightarrow P^*$	execution delay.

The last five operators are taken from ATP [NiS90]. We will call these the Nicollin-Sifakis functions for discrete time process algebra.

3.2 DEFINITIONS

Now we give the definitions of the new signature ingredients in table 6, except for the execution delay $\lfloor \cdot \rfloor(\cdot)$. For $t \in \mathbb{R}$, $\lfloor t \rfloor$ is the floor of t , the largest integer less than or equal to t . We will also on occasion need the largest integer less than t , and define $\lfloor t \rfloor' = \max\{0, \max\{n \in \mathbb{Z} : n < t\}\}$.

$\underline{a} = \int_{t>0} a(t)$	$\underline{\delta} = \int_{t>0} \delta(t)$	
$a(0) = \delta$	$a(n+1) = n \gg (\underline{a} \gg (n+1))$	$\delta(n) = \delta(n)$
$a[0] = \delta$	$\underline{a} = a[1]$	$a[1] = \sqrt{t}. \underline{a} \gg \lfloor t+1 \rfloor$
$\delta[0] = \delta$	$\underline{\delta} = \delta[1]$	$\delta[1] = \sqrt{t}. \underline{\delta} \gg \lfloor t+1 \rfloor$
$T = \sqrt{r}. t(\lfloor r+1 \rfloor)$		
$x \gg_d 1 = \sqrt{t}. (\lfloor t \rfloor \gg x) \gg \lfloor t+1 \rfloor$	$1 \gg_d x = \sqrt{t}. \lfloor t+1 \rfloor' \gg (\lfloor t \rfloor' \gg x)$	
$\sigma_d(x) = \sqrt{t}. \lfloor t+1 \rfloor \gg x$	$\sigma^t(x) = \sqrt{t}. (t+r) \gg x$	
$D(x) = \lambda t. U(\lfloor t \rfloor \gg x) \geq \lfloor t+1 \rfloor$	$\sqrt{d}n. X = \sqrt{t}. X(\lfloor t \rfloor/n)$	
$x \oplus y = \sqrt{t}. (\lfloor t \rfloor \gg x + y \gg \lfloor t+1 \rfloor + \sum_{i=0}^{\infty} \lfloor t+i+1 \rfloor \gg (\lfloor t \rfloor \gg x+y) \gg \lfloor t+i+2 \rfloor$		
	$\triangleleft D(\lfloor t \rfloor \gg x)(\lfloor t+i \rfloor) \wedge D(\lfloor t \rfloor \gg y)(\lfloor t+i \rfloor) \triangleright \underline{\delta}$	
$\lfloor x \rfloor(y) = x \gg_d 1 + \sigma_d(y)$		
$\lfloor x \rfloor^0(y) = y$	$\lfloor x \rfloor^{n+1}(y) = \lfloor x \rfloor(\lfloor x \rfloor^n(y))$	
$\lfloor x \rfloor^\omega = \sqrt{t}. \sum_{i=0}^{\infty} \lfloor t+i \rfloor \gg x \gg \lfloor t+i+1 \rfloor$		

TABLE 6. Discrete time.

3.3 COMMENTS

We will make several comments concerning the definitions in table 6.

1. The definition of T uses a special atomic action t , that satisfies $t \mid t = t$ and $t \mid a = \delta$ for $a \neq t$. T is Groote's time step of [Gro90a]. Thus, Groote's t becomes a 'grote t ', avoiding confusion for Dutchmen.

2. We follow [MOT89] by taking $\underline{\delta}$ (0 in [MOT89]) to be a time stop. $\underline{\delta}$ is like δ in [Gro90a] and like nil in [GM91]. We notice that TPL has no constant that introduces a time stop, but $\underline{\delta}$ seems to be equivalent to an infinite τ -loop in TPL [HER90]. Although time stops do not occur in actual computing, we see no objection to their presence in the calculus. So we do not follow ATP's design rationale, which excludes time stops on a priori grounds. $\underline{a}$ is like atomic actions in ATP.

3. We have a (unit) delay operator σ_d rather than an action representing the progress of time. The name σ for the delay operator has been taken from TPL [HER90] where σ is used to name the action that

represents delay. TCCS [MOT89] has a unary operator for delay just like we have. ATP [NIS90] also introduces delay via an operator, but it is a binary one ($\llbracket \cdot \rrbracket(\cdot)$). $\underline{a}$ models the delayable atoms of TPL.

4. Our $+$ has the interpretation of weak choice in TCCS [MOT89].

5. The notation for the delayability predicate has been taken from TCCS (where it denotes a slightly different operator, however).

3.4 EXECUTION DELAY

In order to give the definition of the execution delay operator $\lceil \cdot \rceil^n(\cdot)$, we first need to define an extra real time operator $\lceil \cdot \rceil_p^1(\cdot)$. We want to stick to the ATP notation for the execution delay operator, and so we use a distinguishing subscript p for the real time variant. We have the axioms in table 7.

$\lceil a(r) \rceil_p^1(x) = a(r) \triangleleft r < t \triangleright t \gg (0 \gg x)$	$\lceil a(r) \cdot x \rceil_p^1(y) = a(r) \lceil r \gg x \rceil_p^1(y) \triangleleft r < t \triangleright t \gg (0 \gg y)$
$\lceil x + y \rceil_p^1(z) = \lceil x \rceil_p^1(z) + \lceil y \rceil_p^1(z)$	$\lceil \sqrt{s} \cdot x \rceil_p^1(y) = \lceil 0 \gg x[0/s] \rceil_p^1(y)$
$\lceil x \rceil_p^0(y) = y$	$\lceil x \rceil^{n+1}(y) = \sqrt{r} \cdot \lceil r \gg x \rceil_p^{r+n+1}(\lceil r+n+1 \rceil \gg y)$

TABLE 7. Execution delay.

3.5 PROPERTIES

1. $n \gg a(m) = a(m) \triangleleft n < m \triangleright \delta(n)$.
2. $\sigma_d(x) + \sigma_d(y) = \sigma_d(x + y)$
3. $\sigma_d(x \cdot y) = \sigma_d(x) \cdot y$.
4. $\lfloor x \rfloor^\omega = \lfloor x \rfloor(\lfloor x \rfloor^\omega)$
5. $x \oplus y = x + y \gg_d 1 + \sigma_d(1 \gg_d x \oplus 1 \gg_d y) \triangleleft D(x) \wedge D(y) \triangleright \delta$.
6. $\sigma_d(x) \gg_d 1 = \delta$.
7. $\lceil a \rceil^{n+1}(x) = a$.
8. $\lceil a \cdot x \rceil^{n+1}(y) = \lceil a \rceil \lceil x \rceil^{n+1}(y)$.
9. $\lceil x + y \rceil^{n+1}(z) = \lceil x \rceil^{n+1}(z) + \lceil y \rceil^{n+1}(z)$.
10. $\lceil \sigma_d(x) \rceil^{n+1}(y) = \sigma_d(\lceil x \rceil^n(y))$.
11. $\lceil \lfloor x \rfloor(y) \rceil^{n+1}(z) = \lfloor \lceil x \rceil^{n+1}(z) \rfloor(\lceil y \rceil^n(z))$.

3.6 LIVENESS

We have that the rigid processes in M_A^* correspond to the real time processes involving absolute time notation only. We can also define a notion that gives the processes that can be specified using relative time notation only by means of the following definition.

We define the predicate $\mathcal{L}: M_A^* \rightarrow \text{BOOL}^*$ as follows: $\mathcal{L}(X) = \lambda t. t \gg X \neq \delta(t)$.

We call a process X *live* if $\mathcal{L}(X) = \lambda t. \text{true}$. We obtain the following properties:

1. $\mathcal{L}(a[0]) = \mathcal{L}(\delta[0]) = \mathcal{L}(a(0)) = \mathcal{L}(\delta(0)) = \lambda t. \text{false}$
2. $\mathcal{L}(a[n+1]) = \mathcal{L}(\delta[n+1]) = \lambda t. \text{true}$
3. $\mathcal{L}(a(n+1)) = \mathcal{L}(\delta(n+1)) = \lambda t. t < n+1$
4. $\mathcal{L}(a(r)) = \mathcal{L}(\delta(r)) = \lambda t. t < r$
5. $\mathcal{L}(x + y) = \lambda t. \mathcal{L}(x)(t) \vee \mathcal{L}(y)(t)$
6. $\mathcal{L}(x \cdot y) = \mathcal{L}(x)$
7. $\mathcal{L}(x \parallel y) = \mathcal{L}(x \ll y) = \mathcal{L}(x \mid y) = \lambda t. \mathcal{L}(x)(t) \wedge \mathcal{L}(y)(t)$
8. $\mathcal{L}(\bigvee_{v \in V} X) = \lambda t. \bigvee_{v \in V} \mathcal{L}(X)(t)$.

In the previous sections, we have sketched a theory $\text{ACPP} \vee I$ and a model M_A^* that contain both real time and discrete time processes. Now we will concentrate on the reduct of the theory and the subalgebra of the model that contain only discrete time processes. For this reason, we introduce the discretisation operator $\mathcal{D}$.

$\mathcal{D}(a(t)) = a(\lfloor t+1 \rfloor)$	
$\mathcal{D}(x + y) = \mathcal{D}(x) + \mathcal{D}(y)$	$\mathcal{D}(x \cdot y) = \mathcal{D}(x) \cdot \mathcal{D}(y)$
$\mathcal{D}(\int_{v \in V} X) = \int_{v \in V} \mathcal{D}(X)$	$\mathcal{D}(\sqrt{t} \cdot X) = \sqrt{t} \cdot \mathcal{D}(X)$

TABLE 8. Discretisation.

3.7 DEFINITION

First of all, we define discretisation on P^* . We have the axioms in table 8.

3.8 PROPERTIES

1. $\mathcal{D}$ does not distribute over $\parallel$, $\mathbb{L}$, $\mathbb{I}$.
2. $\mathcal{D}(\partial_H(x)) = \partial_H(\mathcal{D}(x))$
3. $\lfloor U(x) \rfloor \leq U(\mathcal{D}(x)) \leq \lfloor U(x) + 1 \rfloor$. Here, both extremes are possible.

3.9 DEFINITION

We say that a process x is *discretised*, $\mathcal{DIS}(x)$, if there is a process $y \in P^*$ such that $\mathcal{D}(y) = x$. Also, we define this concept on the model M_A^* . Let $f \in M_A^*$. Then f is discretised, $f \in \mathcal{DIS}M_A^*$, if:

- i. for all $t \geq 0$, all actions of $f(t)$ have a timestamp not in $\mathbb{N}$.
- ii. for all $t \geq 0$ we have the following: let σ be a sequence of actions in $f(t)$ of the form $a_1(r_1) a_2(r_2) \dots a_n(r_n)$ with $r_1 < r_2 < \dots < r_n < r$. Let in $f(t)$ the sequence of actions σ from the root be followed by the step $p \xrightarrow{a(r)} q$. Then:
 - a. for each r' with $r < r' < \lfloor r+1 \rfloor$ there is a q' with $p \xrightarrow{a(r')} q'$ and $q' \Leftrightarrow r' \gg q$.
 - b. for each r' with $\max(r_n, \lfloor r \rfloor) < r' < r$ there is a q' with $p \xrightarrow{a(r')} q'$ and $q \Leftrightarrow r \gg q'$.

3.10 EXAMPLES

1. $\mathcal{DIS}(a(n)), \mathcal{DIS}(a[\lfloor n \rfloor]), \mathcal{DIS}(\delta(n)), \mathcal{DIS}(\delta[\lfloor n \rfloor]), \neg \mathcal{DIS}(a(t))$.
2. $\mathcal{DIS}(x) \Rightarrow \mathcal{DIS}(\partial_H(x)), \mathcal{DIS}(\sigma_d(x)), \mathcal{DIS}(1 \gg_d x), \mathcal{DIS}(x \gg_d 1)$.
3. $\mathcal{DIS}(x) \wedge \mathcal{DIS}(y) \Rightarrow \mathcal{DIS}(x+y), \mathcal{DIS}(x \cdot y), \mathcal{DIS}(x \parallel y), \mathcal{DIS}(x \mathbb{L} y), \mathcal{DIS}(x \mid y)$.
4. for all $v \in V$ $\mathcal{DIS}(X) \Rightarrow \mathcal{DIS}(\bigcup_{v \in V} X)$.
5. $\mathcal{DIS}(x) \wedge \neg D(x) \Rightarrow x = x \gg_d 1$.
6. $\mathcal{DIS}(x) \wedge D(x) \Rightarrow x = x \gg_d 1 + \sigma_d(1 \gg_d x)$.
7. $\mathcal{DIS}(x) \Rightarrow \mathcal{L}(x)$
8. $\mathcal{DIS}(x) \Rightarrow a \mathbb{L} x = a \cdot x$
9. $\mathcal{DIS}(x) \wedge \mathcal{DIS}(y) \Rightarrow \underline{a}x \mathbb{L} y = \underline{a}(x \parallel y)$
10. $\mathcal{DIS}(x) \wedge \mathcal{DIS}(y) \wedge \overline{D}(y) \Rightarrow \sigma_d(x) \mathbb{L} y = \sigma_d(x \mathbb{L} 1 \gg_d y)$.

The remarkable point is that $\mathcal{L}(\underline{\delta})$. Thus, the discretisation of δ is live. The discretisation of δ is certainly not equal to $\delta = \delta(0)$.

3.11 EQUIVALENT MODEL

We can define a model that is conceptually much simpler than $\mathcal{DIS}M_A^*$, but contains an isomorphic copy of it. This is the model $\mathbb{G}/\Leftrightarrow$ that we will use in the following chapter.

We define a set of process graphs as in [BAW90] with labels from $A \cup \{\sigma\}$ satisfying two extra conditions:

- i. every node has *at most one* outgoing σ -labeled edge;
- ii. a σ -labeled edge may not lead to a termination node.

Let $\mathbb{G}$ be the set of such process graphs with cardinality $\leq 2^{\aleph_0}$. To state this precisely, an element of $\mathbb{G}$ is a quadruple $\langle N, E, r, T \rangle$ where N is the set of *nodes*, $E \subseteq N \times A \cup \{\sigma\} \times N$ is the set of *edges*, $r \in N$ is the *root node*, and $T \subseteq N$ is the set of *termination nodes*. We will always have that a termination node has no outgoing edges. A node without outgoing edges that is not a termination node is called a *deadlock node*.

We define a mapping from $\mathcal{DIS}M_A^*$ to $\mathbb{G}/\Leftrightarrow$ by defining a mapping ϕ from real time transition systems to process graphs. Let R be a real time transition system. For simplicity, we assume that R is actually a tree, so each node has at most one incoming transition. If $\langle r, 0 \rangle$ is the root of R ($r \in S$), we take r as the root of $\phi(R)$. The set of states of $\phi(R)$ is S , the set of states of R . Suppose $\langle r, 0 \rangle \xrightarrow{a(t)} \langle s, t \rangle$ is a transition in R from the root. In $\phi(R)$, we take from the root then $\lfloor t \rfloor$ σ -transitions followed by an a -transition to state s . In case of several transitions in R from the root, we share σ -transitions in $\phi(R)$ as much as possible. We proceed likewise for the other states: if $\langle s, u \rangle \xrightarrow{a(t)} \langle s', t \rangle$ is a transition in R , we take from s in $\phi(R)$ $\lfloor t \rfloor - \lfloor u \rfloor$ σ -transitions followed by an a -transition to s' . This describes f . We then obtain a mapping from $\mathcal{DIS}M_A^*$ to $\mathbb{G}/\Leftrightarrow$ by mapping a representative of $f \in \mathcal{DIS}M_A^*$ to $R(f(0))/\Leftrightarrow$.

The inverse mapping can be defined along the same lines. We identify the domain of $\mathbb{G}$ with some subset of $\mathbb{S}$, the fixed domain of states that was mentioned in 2.12. If r is the root of $g \in \mathbb{G}$, and $r \xrightarrow{\sigma} \dots \xrightarrow{a} s$ a sequence of transitions with n σ -steps, add transitions $\langle r, 0 \rangle \xrightarrow{a(t)} \langle s, t \rangle$ for each t between n and $n+1$. Next, if node s of g corresponds to node $\langle s, u \rangle$, and $s \xrightarrow{a} s'$ is a transition in g , add transitions $\langle s, u \rangle \xrightarrow{a(t)} \langle s', t \rangle$ for each t between u and $\lfloor u+1 \rfloor$. On the other hand, if $s \xrightarrow{\sigma} \dots \xrightarrow{a} s'$ is a sequence of transitions in g with n σ -steps, $n \geq 1$, add transitions $\langle s, u \rangle \xrightarrow{a(t)} \langle s', t \rangle$ for each t between $\lfloor u+n \rfloor$ and $\lfloor u+n+1 \rfloor$.

We leave the verification that this indeed defines an isomorphism to the reader.

4. DISCRETE TIME PROCESS ALGEBRA

Now we will incrementally define axiomatisations for discrete time subalgebras of our theory.

4.1 BPA_{dt}

We start from:

- a unary operator for (unit) delay like in TCCS [MOT89];
- time factorisation as in ATP [NIS90], TCCS [MOT89], TPCCS [HAN91] and TPL [HER90];
- the interpretation of $+$ as the weak choice of TCCS;
- a significant difference with TPL [HER90] because visible actions cannot idle (here, we follow TCCS, ACP_t and ATP).

The signature of BPA_{dt} has constants $\underline{a}$ ($a \in A$), functions $+, \cdot: P \times P \rightarrow P$ and the discrete time unit delay $\sigma_d: P \rightarrow P$. The axioms of BPA_{dt} are A1-5 and DT1,2 in table 9. Axioms DT1,2 appear in [BAB91b]. They can be derived from the theory in the previous section.

4.2 INACTION

BPA_{δdt} is obtained by introducing δ as a constant representing inaction (time stop at the end of the current time slice). The axioms for δ are standard (table 9).

Next, we add parallel composition. We start by describing a system with merge without communication, a so-called free merge.

4.3 PA_{δdt}

We start by adding to the signature of BPA_{δdt} the extra sort **BOOL** with constants and functions as 2.9. We add the initialisation in the following time slice, $1 \gg_d: P \rightarrow P$, and the delayability predicate $D: P \rightarrow \text{BOOL}$. Axioms for this additional syntax are in table 3 (those without the $\sqrt{}$ operator). They already appear in [BAB91b]. Then, the merge with its auxiliary operator left merge is introduced. Axioms for this theory are UTB1-4, DEL1-4, CM1 without third summand and CM2-4 in table 9. Axiom DTM1 is from [BAB91b].

Our definition of merge is similar to the definitions in TCCS, ATP, ACP_t and TPL (the correspondence with the TPL definition holds in the absence of communication; in the presence of communication TPL's merge will give priority to internal communications). So it follows that without communication the relation between merge and discrete time is not controversial. We depart from ATP by allowing time stops. Indeed, in our setting $\sigma_d(\underline{a} \cdot \underline{b}) \parallel \delta$ is unable to perform a σ -step. This is a matter of taste. The presence of time stops seems not to spoil the algebra at all.

4.4 COMMUNICATION

If we add communication, we have a given *communication function* $! : A_\delta \times A_\delta \rightarrow A_\delta$ as an extra parameter of the theory. This communication function is required to be commutative, associative and has δ as a neutral element (axioms C1,2,3 of table 1 in 2.3). Furthermore, in order to axiomatise the merge, we have an additional auxiliary operator $\mid$ (communication merge). The DTM axioms are from

[BAB91b]. Encapsulation $\partial_H: P \rightarrow P$ for each $H \subseteq A$ has the same equations as in ACP plus the additional axiom DTM7.

At this point, some further motivation is needed. The merge of ACP_{dt} works just as the merge of TCCS [MOT89] (taking weak choice of TCCS for $+$). It is also equivalent to the merge of ATP. It differs from merge in [GRO90a], set up in ACP_1 . In fact, ACP_1 contains axioms $\sigma \parallel x = \delta$ and $\sigma x \parallel y = \delta$. Our main objection against these axioms (which occur in ATP as well) is that they render it impossible to injectively embed ACP_1 into $ACPP$ of [BAB91a].

It follows that the only disagreement in the literature is about the proper semantics of left merge and communication merge. Therefore, our choice for the merge operator itself seems reasonably well motivated.

$x + y = y + x$	A1	$\underline{a} \mid \underline{b} = \underline{c}$	if $a \mid b = c$	C4
$(x + y) + z = x + (y + z)$	A2			
$x + x = x$	A3	$x \parallel y = x \parallel y + y \parallel x + x \mid y$		CM1
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\underline{a} \parallel x = \underline{a} \cdot x$		CM2
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$(\underline{a} \cdot x) \parallel y = \underline{a} \cdot (x \parallel y)$		CM3
$x + \underline{\delta} = x$	A6	$(x + y) \parallel z = x \parallel z + y \parallel z$		CM4
$\underline{\delta} \cdot x = \underline{\delta}$	A7	$\sigma_d(x) \parallel y = \sigma_d(x \parallel (1 \gg_d y)) \triangleleft D(y) \triangleright \underline{\delta}$		DTM1
		$(\underline{a} \cdot x) \mid \underline{b} = (\underline{a} \mid \underline{b}) \cdot x$		CM5
$\sigma_d(x) + \sigma_d(y) = \sigma_d(x + y)$	DT1	$\underline{a} \mid (\underline{b} \cdot x) = (\underline{a} \mid \underline{b}) \cdot x$		CM6
$\sigma_d(x) \cdot y = \sigma_d(x \cdot y)$	DT2	$(\underline{a} \cdot x) \mid (\underline{b} \cdot y) = (\underline{a} \mid \underline{b}) \cdot (x \parallel y)$		CM7
		$(x + y) \mid z = x \mid z + y \mid z$		CM8
$1 \gg_d \underline{a} = \underline{\delta}$	UTB1	$x \mid (y + z) = x \mid y + x \mid z$		CM9
$1 \gg_d (x + y) = 1 \gg_d x + 1 \gg_d y$	UTB2	$\underline{a} \mid \sigma_d(x) = \underline{\delta}$		DTM2
$1 \gg_d (x \cdot y) = (1 \gg_d x) \cdot y$	UTB3	$\sigma_d(x) \mid \underline{a} = \underline{\delta}$		DTM3
$1 \gg_d \sigma_d(x) = x$	UTB4	$(\underline{a} \cdot x) \mid \sigma_d(y) = \underline{\delta}$		DTM4
		$\sigma_d(x) \mid (\underline{a} \cdot y) = \underline{\delta}$		DTM5
$D(\underline{a}) = \text{false}$	DEL1	$\sigma_d(x) \mid \sigma_d(y) = \sigma_d(x \mid y)$		DTM6
$D(x + y) = D(x) \vee D(y)$	DEL2			
$D(x \cdot y) = D(x)$	DEL3	$\partial_H(\underline{a}) = \underline{a}$ if $a \notin H$		D1
$D(\sigma_d(x)) = \text{true}$	DEL4	$\partial_H(\underline{a}) = \underline{\delta}$ if $a \in H$		D2
		$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
		$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4
		$\partial_H(\sigma_d(x)) = \sigma_d(\partial_H(x))$		DTM7

TABLE 9. ACP_{dt} .

4.6 STRUCTURED OPERATIONAL SEMANTICS

The semantic domain for BPA_{dt} consists of process graphs as defined in 3.11. We will map every closed term to a graph in $\mathbb{G}$ by using a structured operational semantics. We can give a transition system specification by means of the rules in table 10 ($a \in A$). The operational meaning of $\underline{\delta}$ is a process that allows no step whatsoever.

Now we can show that these rules give rise to a unique transition relation on closed terms. This can be done with the theory of [GRO90b].

Bisimulation is defined as usual, so a symmetric binary relation R on process terms is a bisimulation iff the following transfer conditions hold:

- i. if $R(p, q)$ and $p \xrightarrow{u} p'$ ($u \in A \cup \{\sigma\}$), then there is q' such that $q \xrightarrow{u} q'$ and $R(p', q')$;
- ii. if $R(p, q)$, then $p \xrightarrow{a} \sqrt{} \ (a \in A)$ iff $q \xrightarrow{a} \sqrt{} \ .$

Two terms p, q are bisimilar, $p \approx q$, if there exists a bisimulation relating them. We have that the set of process terms modulo bisimulation is a model for ACP_{dt} , and moreover, the axiomatisation of ACP_{dt} is sound and complete for the set of transition graphs modulo bisimulation.

$\underline{a} \xrightarrow{a} \sqrt{} \$	$\sigma_d(x) \xrightarrow{\sigma} x$	
$\frac{x \xrightarrow{a} x'}{x \cdot y \xrightarrow{a} x' \cdot y}$	$\frac{x \xrightarrow{a} \sqrt{} \ }{x \cdot y \xrightarrow{a} y}$	$\frac{x \xrightarrow{\sigma} x'}{x \cdot y \xrightarrow{\sigma} x' \cdot y}$
$\frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x', y + x \xrightarrow{a} x'}$	$\frac{x \xrightarrow{a} \sqrt{} \ }{x + y \xrightarrow{a} \sqrt{} \ , y + x \xrightarrow{a} \sqrt{} \ }$	
$\frac{x \xrightarrow{\sigma} x', y \xrightarrow{\sigma} y'}{x + y \xrightarrow{\sigma} x' + y'}$	$\frac{x \xrightarrow{\sigma} x', y \xrightarrow{\sigma} \sqrt{} \ }{x + y \xrightarrow{\sigma} x', y + x \xrightarrow{\sigma} x'}$	
$\frac{x \xrightarrow{a} x'}{x \parallel y \xrightarrow{a} x' \parallel y, y \parallel x \xrightarrow{a} y \parallel x', x \ll y \xrightarrow{a} x' \ll y}$	$\frac{x \xrightarrow{a} \sqrt{} \ }{x \parallel y \xrightarrow{a} y, y \parallel x \xrightarrow{a} y, x \ll y \xrightarrow{a} y}$	
$\frac{x \xrightarrow{\sigma} x', y \xrightarrow{\sigma} y'}{x \parallel y \xrightarrow{\sigma} x' \parallel y', x \ll y \xrightarrow{\sigma} x' \ll y', x \mid y \xrightarrow{\sigma} x' \mid y'}$	$\frac{x \xrightarrow{a} x', y \xrightarrow{b} y', a \mid b = c}{x \parallel y \xrightarrow{c} x' \parallel y', x \mid y \xrightarrow{c} x' \mid y'}$	
$\frac{x \xrightarrow{a} x', y \xrightarrow{b} \sqrt{} \ , a \mid b = c}{x \parallel y \xrightarrow{c} x', x \mid y \xrightarrow{c} x', y \parallel x \xrightarrow{c} x', y \mid x \xrightarrow{c} x'}$	$\frac{x \xrightarrow{a} \sqrt{} \ , y \xrightarrow{b} \sqrt{} \ , a \mid b = c}{x \parallel y \xrightarrow{c} \sqrt{} \ , x \mid y \xrightarrow{c} \sqrt{} \ }$	
$\frac{x \xrightarrow{\sigma} y, y \xrightarrow{a} z}{1 \gg_d x \xrightarrow{a} z}$	$\frac{x \xrightarrow{\sigma} y, y \xrightarrow{a} \sqrt{} \ }{1 \gg_d x \xrightarrow{a} \sqrt{} \ }$	$\frac{x \xrightarrow{\sigma} y, y \xrightarrow{\sigma} z}{1 \gg_d x \xrightarrow{\sigma} z}$
$\frac{x \xrightarrow{a} x', a \notin H}{\partial_H(x) \xrightarrow{a} \partial_H(x')}$	$\frac{x \xrightarrow{a} \sqrt{} \ , a \notin H}{\partial_H(x) \xrightarrow{a} \sqrt{} \ }$	$\frac{x \xrightarrow{\sigma} x'}{\partial_H(x) \xrightarrow{\sigma} \partial_H(x')}$

TABLE 10. Operational semantics of ACP_{dt} .

4.7 SYNTAX

The syntax of ACP_{dt} is not tailored to practical application. The operators are too low level. We will extend ACP_{dt} with ATP operators in order to extend the expressive power of the language. In addition, we need an auxiliary operator $\gg_d 1$. We have functions

$$\gg_d 1: P \rightarrow P$$

time out in the current time slice

$\oplus: P \times P \rightarrow P$	strong choice
$\lfloor \cdot \rfloor(.): P \times P \rightarrow P$	unit delay
$\lfloor \cdot \rfloor^n(.): P \times P \rightarrow P$	start delay within n ($n \in \mathbb{N}$)
$\lfloor \cdot \rfloor^\omega: P \rightarrow P$	unbounded start delay
$\lfloor \cdot \rfloor^n(.): P \times P \rightarrow P$	execution delay.

The last five functions are the Nicollin-Sifakis functions.

4.8 AXIOMS

Axioms for these auxiliary operators are in table 11. Axioms UTB5-8 and ATP1,2 are taken from [BAB91b]. For ATP1,3,6 we refer to table 7, the other axioms can be shown to hold in the theory of section 3.

$\underline{a} \gg_d 1 = \underline{a}$	UTB5	$(x \cdot y) \gg_d 1 = (x \gg_d 1) \cdot y$	UTB7
$(x + y) \gg_d 1 = x \gg_d 1 + y \gg_d 1$	UTB6	$\sigma_d(x) \gg_d 1 = \underline{\delta}$	UTB8
$\lfloor x \rfloor(y) = x \gg_d 1 + \sigma_d(y)$			ATP1
$x \oplus y = x + y \gg_d 1 + \sigma_d(1 \gg x \oplus 1 \gg y) \triangleleft D(x) \wedge D(y) \triangleright \underline{\delta}$			ATP2
$\lfloor x \rfloor^0(y) = y$	ATP3	$\lfloor x \rfloor^{n+1}(y) = \lfloor x \rfloor(\lfloor x \rfloor^n(y))$	ATP4
$\lfloor x \rfloor^\omega = \lfloor x \rfloor(\lfloor x \rfloor^\omega)$	ATP5	$\lfloor x \rfloor^0(y) = y$	ATP6
$\lfloor \underline{a} \rfloor^{n+1}(x) = \underline{a}$	ATP7	$\lfloor x + y \rfloor^{n+1}(z) = \lfloor x \rfloor^{n+1}(z) + \lfloor y \rfloor^{n+1}(z)$	ATP9
$\lfloor \underline{a} x \rfloor^{n+1}(y) = \underline{a} \lfloor x \rfloor^{n+1}(y)$	ATP8	$\lfloor \sigma_d(x) \rfloor^{n+1}(y) = \sigma_d(\lfloor x \rfloor^n(y))$	ATP10

TABLE 11. Additional axioms for Nicollin-Sifakis functions.

4.9 OPERATIONAL SEMANTICS

For the new operators, we have the following operational rules (table 12).

4.10. ABSTRACTION

An advantage of concentrating on the subtheory $ACP_{dt} + ATP$ is that it is easier to add the silent step τ . In real time, this is still under investigation (see [KLU92]), but on our discrete time subtheory, it is relatively straightforward. We will not interpret the constant τ in real time. Clearly, that is an option, but it requires real time abstraction to be worked out. (Besides [KLU92], other approaches exist.)

We add a constant $\underline{\tau}$ ($\tau \notin A$), and for each $l \subseteq A$ an operator $\tau_l: P \rightarrow P$. Further, we have an operator τ_σ that substitutes τ for time steps. All axioms in tables xxx are now valid for $a \in A \cup \{\delta, \tau\}$. Additional axioms are shown in table 13 (where also $a \in A \cup \{\delta, \tau\}$). Here, we follow the presentation of the silent step according to *branching bisimulation semantics* (see [BAW90]). Alternatively, it is possible to present an axiomatisation of the silent step in *weak bisimulation semantics* (see [MIL89]).

4.11 SEMANTICS

The operational semantics is obtained by using the rules of tables xxx also in case $a = \tau$, but dividing out a different congruence relation. Let E be the set of process terms enriched with the extra element $\checkmark$. Write $x \Rightarrow y$ for elements of E if either $x=y$ or there is path from x to y (generated by the operational rules) that only contains τ -labels.

A *branching bisimulation* is a symmetric binary relation R on E with the following transfer property:

if $R(x, y)$ and $x \xrightarrow{a} x'$ ($a \in A \cup \{\tau\}$), then either $a=\tau$ and $R(x', y)$, or there are y^*, y' in E such that $y \Rightarrow y^* \xrightarrow{a} y'$ and $R(x, y^*), R(x', y')$.

$\frac{x+y \xrightarrow{a} z}{x \oplus y \xrightarrow{a} z}$	$\frac{x+y \xrightarrow{a} \checkmark}{x \oplus y \xrightarrow{a} \checkmark}$	$\frac{x \xrightarrow{\sigma} x', y \xrightarrow{\sigma} y'}{x \oplus y \xrightarrow{\sigma} x' \oplus y'}$
$\frac{}{[x](y) \xrightarrow{\sigma} y}$	$\frac{x \xrightarrow{a} x'}{[x](y) \xrightarrow{a} x'}$	$\frac{x \xrightarrow{a} \checkmark}{[x](y) \xrightarrow{a} \checkmark}$
$\frac{y \xrightarrow{a} y'}{[x]^0(y) \xrightarrow{a} y'}$	$\frac{y \xrightarrow{a} \checkmark}{[x]^0(y) \xrightarrow{a} \checkmark}$	$\frac{y \xrightarrow{\sigma} y'}{[x]^0(y) \xrightarrow{\sigma} y'}$
$\frac{}{[x]^{n+1}(y) \xrightarrow{\sigma} [x]^n(y)}$	$\frac{x \xrightarrow{a} x'}{[x]^{n+1}(y) \xrightarrow{a} x'}$	$\frac{x \xrightarrow{a} \checkmark}{[x]^{n+1}(y) \xrightarrow{a} \checkmark}$
$\frac{}{[x]^\omega \xrightarrow{\sigma} [x]^\omega}$	$\frac{x \xrightarrow{a} x'}{[x]^\omega \xrightarrow{a} x'}$	$\frac{x \xrightarrow{a} \checkmark}{[x]^\omega \xrightarrow{a} \checkmark}$
$\frac{y \xrightarrow{a} y'}{[x]^0(y) \xrightarrow{a} y'}$	$\frac{y \xrightarrow{a} \checkmark}{[x]^0(y) \xrightarrow{a} \checkmark}$	$\frac{y \xrightarrow{\sigma} y'}{[x]^0(y) \xrightarrow{\sigma} y'}$
$\frac{x \xrightarrow{a} x'}{[x]^{n+1}(y) \xrightarrow{a} [x']^{n+1}(y)}$	$\frac{x \xrightarrow{a} \checkmark}{[x]^{n+1}(y) \xrightarrow{a} \checkmark}$	$\frac{x \xrightarrow{\sigma} x'}{[x]^{n+1}(y) \xrightarrow{\sigma} [x']^{n+1}(y)}$

TABLE 12. Operational semantics of Nicollin-Sifakis functions.

$x \tau = x$	B1	$\tau \mid \underline{a} = \underline{\delta}$	C4
$x(\tau(y + z) + y) = x(y + z)$	B2		
$\tau_\sigma(\underline{a}) = \underline{a}$	TS1	$\tau_1(\underline{a}) = \underline{a}$ if $a \notin I$	TI1
$\tau_\sigma(x + y) = \tau_\sigma(x) + \tau_\sigma(y)$		$\tau_1(\underline{a}) = \tau$ if $a \in I$	TI2
if $D(x) \vee D(y) = \text{false}$	TS2	$\tau_1(x + y) = \tau_1(x) + \tau_1(y)$	TI3
$\tau_\sigma(x \cdot y) = \tau_\sigma(x) \cdot \tau_\sigma(y)$	TS3	$\tau_1(x \cdot y) = \tau_1(x) \cdot \tau_1(y)$	TI4
$\tau_\sigma(\sigma_d(x)) = \tau \tau_\sigma(x)$	TS4	$\tau_1(\sigma_d(x)) = \sigma_d(\tau_1(x))$	TI8

TABLE 13. Axioms of ACP_{dt}^τ .

We say that two process terms are *branching bisimilar*, $x \approx_b y$, if there is a branching bisimulation R on E with $R(x, y)$.

Now let R be a branching bisimulation on E . We say that x, y *satisfy the root condition* if we have:

if $R(x, y)$ and $x \xrightarrow{a} x'$ ($a \in A \cup \{\tau\}$), then there is y' in E such that $y \xrightarrow{a} y'$ and $R(x', y')$.

Then, we say that two process terms are *rooted branching bisimilar*, $x \approx_{rb} y$, if there exists a branching bisimulation R such that:

- i. x, y satisfy the root condition;
- ii. whenever $R(x', y')$ and there is a path from x to x' with the last transition a σ -transition, then x', y' satisfy the root condition.

4.12 FAIR ABSTRACTION

Besides KFAR and CFAR (see [BAW90]), there is a rule for time:

$\frac{x = \sigma_d(x) + y}{\tau \cdot \tau_\sigma(x) = \tau \cdot \tau_\sigma(y)}$	σFAR
---	--------------------

TABLE 14. Time abstraction.

Using Nicollin-Sifakis functions, the fair abstraction principle can be formulated as follows:

$$\frac{x = \lfloor y \rfloor(x)}{\tau \cdot \tau_\sigma(x) = \tau \cdot \tau_\sigma(y)}, \text{ or alternatively: } \tau \cdot \tau_\sigma(\lfloor x \rfloor^\omega) = \tau \cdot \tau_\sigma(x) \quad \sigma\text{FAR.}$$

5. EXAMPLE.

We describe the alternating bit protocol with time out of [NIS90]. Put $B = \{0, 1\}$.

FIGURE 1.

Sender: $S = S_0 \cdot S_1 \cdot S$

$$S_b = \sum_{d \in D} \lfloor r1(d) \rfloor^\omega \cdot S_{b_d} \quad S_{b_d} = s2(db) \cdot \lfloor r6(b) + r6(1-b) \cdot S_{b_d} \rfloor (S_{b_d}) \quad (b=0,1, d \in D).$$

$$\text{Unreliable data transmission channel: } K = \sum_{f \in D \times B} \lfloor r2(f) \rfloor^\omega \cdot (i \cdot K + i \cdot s3(f) \cdot K).$$

$$\text{Unreliable acknowledgement transmission channel: } L = \sum_{b \in B} \lfloor r5(b) \rfloor^\omega \cdot (i \cdot L + i \cdot s6(b) \cdot L).$$

Receiver: $R = R_1 \cdot R_0 \cdot R$

$$R_b = \sum_{d \in D} \lfloor r3(db) \rfloor^\omega \cdot s5(b) \cdot R_b + \sum_{d \in D} \lfloor r3(d(1-b)) \rfloor^\omega \cdot s4(d) \cdot s5(1-b).$$

Then the protocol is described by: $ABP = \tau_\sigma \circ \tau_I \circ \partial_H(S \parallel K \parallel L \parallel R)$, where

$$H = \{ri(x), si(x) \mid i \in \{2,3,5,6\}, x \in \{0,1\} \cup D \times \{0,1\}\}, I = \{ci(x) \mid i \in \{2,3,5,6\}, x \in \{0,1\} \cup D \times \{0,1\}\} \cup \{i\}.$$

We can derive that the process $X = \partial_H(S \parallel K \parallel L \parallel R)$ satisfies the following recursive specification:

$$\begin{aligned} X &= \sum_{d \in D} \lfloor r1(d) \rfloor^\omega \cdot X_d^1 & Y &= \sum_{d \in D} \lfloor r1(d) \rfloor^\omega \cdot Y_d^1 \\ X_d^1 &= c2(d0) \cdot (i \cdot \sigma_d(X_d^1) + i \cdot c3(d0) \cdot s4(d) \cdot X_d^2) & Y_d^1 &= c2(d1) \cdot (i \cdot \sigma_d(Y_d^1) + i \cdot c3(d1) \cdot s4(d) \cdot Y_d^2) \end{aligned}$$

$$X_d^2 = c5(0) \cdot (i \cdot \sigma_d(X_d^3) + i \cdot c6(0) \cdot Y)$$

$$Y_d^2 = c5(1) \cdot (i \cdot \sigma_d(Y_d^3) + i \cdot c6(1) \cdot X)$$

$$X_d^3 = c2(d0) \cdot (i \cdot \sigma_d(X_d^3) + i \cdot c3(d0) \cdot X_d^2)$$

$$Y_d^3 = c2(d1) \cdot (i \cdot \sigma_d(Y_d^3) + i \cdot c3(d1) \cdot Y_d^2)$$

Due to the fact that a complete cycle takes place in one time slice, the sender never receives a wrong bit, and so the specification could be simplified. We see that this protocol is more a PAR protocol (Positive Acknowledgement with Retransmission) than an ABP protocol.

After abstraction from internal actions, and using RSP, we obtain the following specification:

$$\tau_1(X) = \sum_{d \in D} [r1(d)] \cdot \tau_d^1$$

$$Z_d^1 = \tau \cdot \sigma_d(Z_d^1) + \tau \cdot s4(d) \cdot Z_d^2$$

$$Z_d^2 = \tau \cdot \sigma_d(Z_d^3) + \tau \cdot \tau_1(X)$$

$$Z_d^3 = \tau \cdot \sigma_d(Z_d^3) + \tau \cdot Z_d^2$$

Abstraction from time and using fair abstraction gives the desired result:

$$ABP = \tau \circ \tau_1(X) = \tau \cdot \sum_{d \in D} r1(d) \cdot s4(d) \cdot ABP$$

Notice that we need a generalisation of the fair abstraction rule in order to obtain this result.

6. CONCLUDING REMARKS

We have proposed a theory on discrete time process algebra that extends ACP with the features of ATP in a setting consistent with that of ACPp. Many options for further work remain. We mention some:

- i. All extensions of ACP that have been developed can be integrated in the discrete time setting, such as state operators, process creation, signals, priorities, asynchronous communication (see [BAW90] for these ACP extensions).
- ii. The proof theory of ACP_{dt} with initialisation abstraction requires further work. Interesting are principles dealing with infinite behaviour as RDP, RSP, AIP. In particular, the special case of regular processes needs attention because regularity requires a new definition in the presence of discrete time.
- iii. The analysis of ACP_{dt} and its extensions as a term rewriting system may be worth attention.
- iv. Like in the case of ACP, many more models than the bisimulation model are possible. Investigation of failures, ready, trace models as well as projective limit constructions will be worth while.
- v. A theory that explains how discrete time process algebra can be used to develop simulation techniques for real time processes would be very interesting. We think that along this line the best perspective for application of discrete time process algebra can be found.

REFERENCES

- [BAB91a] J.C.M. BAETEN & J.A. BERGSTRA, *Real time process algebra*, Formal Aspects of Computing 3 (2), 1991, pp. 142-188.
- [BAB91b] J.C.M. BAETEN & J.A. BERGSTRA, *A survey of axiom systems for process algebras*, report P9111, Programming Research Group, University of Amsterdam 1991.
- [BAB91c] J.C.M. BAETEN & J.A. BERGSTRA, *Process algebra with signals and conditions*, report CS-R9103, CWI, Amsterdam 1991. To appear in Proc. NATO Summer School, Marktoberdorf 1990 (M. Broy, ed.), Springer Verlag.
- [BAB92] J.C.M. BAETEN & J.A. BERGSTRA, *Real space process algebra*, report CSN 92/03, Dept. of Comp. Sci., Eindhoven University of Technology 1992.

- [BAW90] J.C.M. BAETEN & W.P. WEIJLAND, *Process algebra*, Cambridge Tracts in Theor. Comp. Sci. 18, Cambridge University Press 1990.
- [BEK84] J.A. BERGSTRA & J.W. KLOP, *Process algebra for synchronous communication*, Inf. & Control 60, 1984, pp. 109-137.
- [BRHR84] S.D. BROOKES, C.A.R. HOARE & A.W. ROSCOE, *A theory of communicating sequential processes*, JACM 31 (3), 1984, pp. 560-599.
- [GRO90a] J.F. GROOTE, *Specification and verification of real time systems in ACP*, in: Proc. 10th Symp. on Protocol Specification, Testing and Verification, Ottawa (L. Logrippo, R.L. Probert & H. Ural, eds.), North-Holland, Amsterdam 1990, pp. 261-274.
- [GRO90b] J.F. GROOTE, *Transition system specifications with negative premises*, in: Proc. CONCUR'90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), LNCS 458, Springer 1990, pp. 332-341.
- [GM91] D.P. GRUSKA & A. MAGGIOLO-SCHETTINI, *A timed process description language based on CCS*, report TR-9/91, Università di Pisa 1991.
- [HAN91] H. HANSSON, *Time and probabilities in formal design of distributed systems*, Ph.D. thesis, report DoCS 91/27, Uppsala University, 1991.
- [HER90] M. HENNESSY & T. REGAN, *A temporal process algebra*, report 2/90, University of Sussex 1990.
- [KLU91] A.S. KLUSENER, *Completeness in real time process algebra*, report CS-R9106, CWI Amsterdam 1991. Extended abstract in Proc. CONCUR'91, Amsterdam (J.C.M. Baeten & J.F. Groote, eds.), Springer LNCS 527, 1991, pp. 376-392.
- [KLU92] A.S. KLUSENER, *Abstraction in real time process algebra*, to appear in Proc. REX Workshop on Real Time: Theory in Practice (J.W. de Bakker, C. Huizing, W.P. de Roever & G. Rozenberg, eds.), Mook 1991, Springer LNCS.
- [MIL89] R. MILNER, *Communication and concurrency*, Prentice-Hall 1989.
- [MOT89] F. MOLLER & C. TOFTS, *A temporal calculus of communicating systems*, report LFCS-89-104, University of Edinburgh 1989.
- [MOT90] F. MOLLER & C. TOFTS, *A temporal calculus of communicating systems*, in: Proc. CONCUR'90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), LNCS 458, Springer 1990, pp. 401-415.
- [NIS90] X. NICOLLIN & J. SIFAKIS, *The algebra of timed processes ATP: theory and application*, report RT-C26, IMAG, Grenoble 1990.
- [NSVR90] X. NICOLLIN, J.-L. RICHIER, J. SIFAKIS & J. VOIRON, *ATP: an algebra for timed processes*, in Proc. IFIP TC2 Conf. on Progr. Concepts & Methods, Sea of Gallilee, Israel 1990.
- [RER88] G.M. REED & A.W. ROSCOE, *A timed model for communicating sequential processes*, TCS 58, 1988, pp. 249-261.