Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.F. Groote

Specification and verification of real time systems in ACP

Computer Science/Department of Software Technology Report CS-R9015 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Specification and Verification of Real Time Systems in ACP

Jan Friso Groote

Centre for Mathematics and Computer Science

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract

A language, based on ACP (Algebra of Communicating Processes), is proposed to describe and verify real time systems using a discrete time scale. This language is called $\mathrm{ACP}^t_{\tau\epsilon}$. $\mathrm{ACP}_{\tau\epsilon}$ is the timeless variant of $\mathrm{ACP}^t_{\tau\epsilon}$. Contrary to many other proposals for real time languages $\mathrm{ACP}^t_{\tau\epsilon}$ is simple, but powerful. It unifies real time, abstraction, parallelism and communication in one algebraic framework.

We give a set of axioms describing the properties of $\mathrm{ACP}^t_{\tau\epsilon}$, together with a corresponding operational semantics. Furthermore, we define an interpretation of $\mathrm{ACP}_{\tau\epsilon}$ in $\mathrm{ACP}^t_{\tau\epsilon}$, thereby fixing an intuition behind time in $\mathrm{ACP}_{\tau\epsilon}$. Some examples show the use of $\mathrm{ACP}^t_{\tau\epsilon}$.

Key Words & Phrases: Real Time, Process Algebra, Concurrency, Real Time Translation.
1985 Mathematics Subject Classification: 68Q60
1982 CR Categories: D.3.1, F.3.1, J.7
Note: The author is supported by the European Communities under RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS). However, this document does not necessarily reflect the view of the project.

1 Introduction

Distributed computer systems working on a real time basis find an increasing number of applications, especially in industry. Therefore, it is worthwhile to develop formal techniques in order to construct reliable and correct real time computer systems. We call any system a real time system if its interactions with the environment must satisfy certain time constraints. These constraints can be of various kinds, e.g. maximal reaction times, minimal delay times etc.

The importance of real time is illustrated by the huge amount of papers that have been published in the last decade. To mention just a few: [1, 11, 18, 20, 24, 25, 27, 28], most of them introducing real time into process algebra in different ways. It is our feeling that most existing approaches are complicated, which decreases their use in practical applications. Therefore, we start from a simple idea, namely that the proceeding of a time unit is represented by an action, which we denote by t and which we call a *time step*, and elaborate on it in the well developed setting of ACP [3, 6] obtaining $\mathrm{ACP}^t_{\tau\epsilon}$. Technically, our approach is closest to the work of RICHIER, SIFAKIS and VOIRON [28].

The idea turns out to work well, both for the development of theory and in examples. Because we can use the techniques available in ACP we have parallelism and abstraction almost for free in our framework. We present an operational model in Plotkin style together with a complete axiom system for weak bisimulation semantics [22]. Weak bisimulation is arbitrarily chosen. The theory could as well be developed using for instance branching bisimulation [16, 13], ready trace semantics, failure semantics etc. We introduce several delay operators and we illustrate their use by an example. Finally, we show how $\mathrm{ACP}_{\tau\epsilon}$ processes can be interpreted in $\mathrm{ACP}^t_{\tau\epsilon}$ preserving existing identities between $\mathrm{ACP}_{\tau\epsilon}$ expressions.

The idea that the proceeding of time is modeled by an action, is criticized for the following reason. From an operational perspective it is not natural to take time to be an action, because actions can

be prevented from happening. This means that time can be blocked, which is impossible in reality. However, we think that blocking time is rather useful for analysis of real time systems. For instance, when two systems are composed in parallel in such a way that one system *must* synchronize at times the other is not available, analysis will reveal that time cannot proceed from a certain point onwards without violating the behavior of one of the subsystems. In this case a *time deadlock* indicates the time inconsistency in the composition. It is of course clear that only systems without time deadlocks can be implemented.

2 The language and its axioms

$\mathrm{ACP}_{\tau\epsilon}^{t}$ is an algebraic theory about real time processes that has two parameters, *Act* and γ. *Act* is a (possibly infinite) set of *atomic actions* representing the different basic activities of these processes. These actions are considered to be pointwise in time, i.e. their execution does not take time. $\gamma : Act \times Act \rightarrow Act$ is a *communication function*. It is a partial, associative and commutative function that defines how actions in *Act* can synchronize in order to obtain interactions between processes.

The signature of $\mathrm{ACP}_{\tau\epsilon}^{t}$ consists of a constant name for all actions in *Act*. Furthermore, it contains a constant t, the *time step*, that represents the progress of one time unit. In this article a time unit corresponds to a physical amount of time, such as for instance a millisecond or a minute. A constant δ is included, called *inaction*, which cannot perform any activity. δ cannot even take part in the proceeding of time. A constant ϵ stands for the *empty process*, which can do nothing but terminate immediately, and a constant τ represents an *internal action*. Just as the actions in *Act*, τ is also pointwise in time.

The signature contains the binary operators $+$ (*alternative composition*), $\cdot$ (*sequential composition*) and $\|$ (*parallel composition*). All these operators behave as in $\mathrm{ACP}_{\tau\epsilon}$ with as differences that the action t must synchronously happen in both sides of the parallel operator and the execution of the sequential operator does not take time. In contrast with other approaches [1, 18, 24, 25] the $+$ is not time deterministic, i.e. in $t \cdot p + t \cdot q$ a choice is made between $t \cdot p$ and $t \cdot q$ by selecting an initial t. The function name for sequential composition is often omitted in process expressions. As usual, to give a finite axiomatization of the parallel operator [23], the *left-merge* ($\between$) and the *communication-merge* ($|$) are introduced. In the left-merge the first action must come from the left hand side. This action may not be a t action. The first action of the communication-merge must be a synchronization of two actions, each from one component. These actions may be internal steps. $\cdot$ binds stronger than $\|$, $\between$ and $|$ which in turn bind stronger than $+$.

The *encapsulation operator* (∂_H, $H \subseteq Act$) and the *abstraction operator* (τ_I, $I \subseteq Act$) are the unary operators in $\mathrm{ACP}_{\tau\epsilon}^{t}$. The encapsulation operator renames actions in H to δ and the abstraction operator renames actions in I to τ. These operators are not allowed to rename t and τ.

The signature of $\mathrm{ACP}_{\tau\epsilon}^{t}$ is summarized in table 1.

Example 2.1. We will try to give the reader some idea about the properties of the language $\mathrm{ACP}_{\tau\epsilon}^{t}$. Consider the process expressions:

1. $a \cdot t \cdot b + c \cdot t \cdot t \cdot d$,
2. $a \cdot b$,
3. $a^{\omega} = a \cdot a \cdot a \cdot \ldots$.

The first process must immediately perform either an a or a c action. If it starts with an a action, then after exactly one time unit it must perform a b. If it initially selects a c action, then two time units later, it must do a d. The second process must perform an a, instantaneously followed by a b; a and b happen at the same time, only b depends causally on a. The last process illustrates an imperfection in the current approach. Here a^{ω} is the process that can do an infinite number of a actions in no time.

$\Sigma(\mathrm{ACP}^t_{\tau\epsilon})$:	constants	a	for any atomic action $a \in Act$
		δ	inaction
		τ	silent action
		ϵ	empty process
		t	time step
	unary operators:	∂_H	encapsulation, for any $H \subseteq Act$
		τ_I	hiding, for any $I \subseteq Act$
	binary operators:	$+$	alternative composition
		$\cdot$	sequential composition (sum)
		$\|\|$	parallel composition (merge)
		$\lfloor\!\lfloor$	left-merge
		$\|$	communication-merge

Table 1.

Later we will give constructs that enable us to define a^ω precisely. Intuitively, such a process is not very appealing and, although we can get rid of it by introduction of some constraints on processes, we do not feel that this kind of process definitions will yield any problem when using our algebra.

Equalities between process terms can be calculated using the normal inference rules of equational logic. These rules are summarized in table 3. In the table $p, q, r, p_1, p_2, q_1, q_2$ are open terms over the signature of $\mathrm{ACP}^t_{\tau\epsilon}$. O_1 stands for ∂_H or τ_I and O_2 may be replaced by $+, \cdot, \|, \lfloor\!\lfloor$ or $|$. σ is a substitution assigning a process term to each variable. Substitutions can be applied to terms in the standard way. A set T of axioms proves an equation $p = q$, notation $T \vdash p = q$, if there is an equational logic proof tree with root $p = q$ from the axioms in T in the usual sense.

The set of axioms of $\mathrm{ACP}^t_{\tau\epsilon}$ in table 2 characterizes the basic properties of the function symbols in the signature. In this table a, b range over $Act_\delta = Act \cup \{\delta\}$ and x, y, z are variables. The axiom system is sound and complete for recursion free closed process expressions with respect to an operational model, which will be given in the sequel. The axioms mainly have the usual form of ACP axioms [3, 5]. However, there are a few differences with previous work concerning the axioms of the auxiliary operators. The differences are due to the empty process and real time in combination with the communication-merge. The most remarkable new axioms are EM6, TIM1 and EM13. EM6 says that two terminated processes in a communication-merge are indistinguishable from the empty process. Note that with this axiom we can easily derive that $\epsilon \parallel \epsilon = \epsilon$. TIM1 expresses that time cannot proceed in only one component of the merge. Axiom EM13 is an axiom with no clear intuition. Our choice is mainly motivated by theorem 6.6, which would not hold if our axioms were chosen as usual. As we will see by introducing the operational model, no unwanted identities between processes without auxiliary function names ($\lfloor\!\lfloor, |$) are introduced. Therefore, we feel that the use of EM13 is justified.

$\mathrm{ACP}_{\tau\epsilon}$ is the theory obtained by leaving out all references to t. This means that the signature of $\mathrm{ACP}_{\tau\epsilon}$ is the signature of $\mathrm{ACP}^t_{\tau\epsilon}$ without t. The axioms can be found in table 2 by leaving out T1t,T3t,TIM1-TIM5,TID and TIT. All definitions that will be given for $\mathrm{ACP}^t_{\tau\epsilon}$ also apply to $\mathrm{ACP}_{\tau\epsilon}$.

$\mathrm{ACP}^t_{\tau\epsilon}$ can be used to define finite processes, directly. We specify infinite processes by means of (sets of) equations. For example, the equation $x = a \cdot x$ specifies the process a^ω ($=a \cdot a \cdot$ ... infinitely many a's) because it is the only solution of this equation.

Not all equations have unique solutions, e.g. any process satisfies the equation $x = x$. Therefore, we introduce a *guarded recursive specification* as a set of guarded recursive equations. We will formulate two principles, RDP and RSP, together stating that any guarded recursive specification defines exactly one process.

Definition 2.2. Let p be an open term over the signature $\mathrm{ACP}^t_{\tau\epsilon}$. An occurrence of a variable x in

$\mathrm{ACP}_{\tau\epsilon}^{t}$

$x+(y+z)=(x+y)+z$	A1	$a\tau=a$	T1a
$x+y=y+x$	A2	$\tau\tau=\tau$	T1b
$x+x=x$	A3	$t\tau=t$	T1t
$(x+y)z=xz+yz$	A4	$\tau\cdot x+x=\tau\cdot x$	T2
$(xy)z=x(yz)$	A5	$a(\tau x+y)=a(\tau x+y)+ax$	T3
$x+\delta=x$	A6	$t(\tau x+y)=t(\tau x+y)+tx$	T3t
$\delta x=\delta$	A7		
$\epsilon x=x$	A8	$a \mid b=\gamma(a,b)$ if $\gamma(a,b)$ defined	
$x\epsilon=x$	A9	$a \mid b=\delta$ if $\gamma(a,b)$ undefined	
$x \parallel y=x\between y+y\between x+x \mid y$	EM1	$\tau x\between y=\tau(x \parallel y)$	EM10
$\epsilon\between x=\delta$	EM2	$tx\between y=\delta$	TIM1
$ax\between y=a(x \parallel y)$	EM3	$\epsilon \mid tx=\delta$	TIM2
$(x+y)\between z=x\between z+y\between z$	EM4	$\epsilon \mid \tau x=\epsilon \mid x$	EM11
$x \mid y=y \mid x$	EM5	$\tau x \mid ay=x \mid ay$	EM12
$\epsilon \mid \epsilon=\epsilon$	EM6	$\tau x \mid \tau y=\tau(x \parallel y)$	EM13
$\epsilon \mid ax=\delta$	EM7	$tx \mid ty=t(x \parallel y)$	TIM3
$ax \mid by=(a \mid b)(x \parallel y)$	EM8	$tx \mid ay=\delta$	TIM4
$(x+y) \mid z=x \mid z+y \mid z$	EM9	$tx \mid \tau y=tx \mid y$	TIM5
$\partial_H(\tau)=\tau$	DT	$\tau_I(\tau)=\tau$	TI1
$\partial_H(t)=t$	TID	$\tau_I(t)=t$	TIT
$\partial_H(\epsilon)=\epsilon$	DE	$\tau_I(\epsilon)=\epsilon$	TE
$\partial_H(a)=a$ if $a\notin H$	D1	$\tau_I(a)=a$ if $a\notin I$	TI2
$\partial_H(a)=\delta$ if $a\in H$	D2	$\tau_I(a)=\tau$ if $a\in I$	TI3
$\partial_H(x+y)=\partial_H(x)+\partial_H(y)$	D3	$\tau_I(x+y)=\tau_I(x)+\tau_I(y)$	TI4
$\partial_H(xy)=\partial_H(x)\partial_H(y)$	D4	$\tau_I(xy)=\tau_I(x)\tau_I(y)$	TI5

Table 2.

$p=p$ $\quad\frac{p=q}{q=p}\quad\frac{p=q\ \ q=r}{p=r}$	$\sigma(p)=\sigma(q)$ if $p=q$ is an axiom
$\frac{p=q}{O_1(p)=O_1(q)}$	$\frac{p_1=q_1\ \ p_2=q_2}{O_2(p_1,p_2)=O_2(q_1,q_2)}$

Table 3.

p is *guarded* if it occurs in a subterm $a \cdot p'$ of p and p does not contain the τ_I operator. p is *guarded* if all variables occur guarded in p.

Definition 2.3. A set E of equations of the form $\{x_i = p_i | i \in I\}$ over the signature $\mathrm{ACP}_{\tau\epsilon}^t$, with I an index set, is called a *recursive specification over* $\mathrm{ACP}_{\tau\epsilon}^t$ if for every variable x in p_i $(i \in I)$, $x \equiv x_j$ for some $j \in I$. E is called a *guarded recursive specification over* $\mathrm{ACP}_{\tau\epsilon}^t$ if E is a recursive specification and if every p_i $(i \in I)$ is guarded.

Principle 2.4. (Recursive Definition Principle, RDP). Every recursive specification has at least one solution.

Principle 2.5. (Recursive Specification Principle, RSP). Every guarded recursive specification has at most one solution.

Using both principles, we can specify real time processes by guarded recursive specifications.

Example 2.6. A typical real time device is a stopwatch. It can be used to measure time durations. Suppose the stopwatch has as visible actions $go, stop, reset$ and $read_n$ for all time values $n \in \mathbf{N}$ with their obvious meanings. The stopwatch is given by the following infinite system of guarded recursive equations. Here $Stopwatch(i)$ and $\overline{Stopwatch(i)}$ are variables for every $i \in \mathbf{N}$. The process that we want to specify is the (unique) solution for $Stopwatch(0)$.

$$\begin{aligned}
Stopwatch(i) &= stop \cdot \overline{Stopwatch(i)} + \\
&\quad read_i \cdot Stopwatch(i) + \\
&\quad t \cdot Stopwatch(i+1), \\
\overline{Stopwatch(i)} &= go \cdot Stopwatch(i) + \\
&\quad reset \cdot \overline{Stopwatch(0)} + \\
&\quad read_i \cdot \overline{Stopwatch(i)} + \\
&\quad t \cdot \overline{Stopwatch(i)}.
\end{aligned}$$

The stopwatch can either be measuring time (represented by $Stopwatch(i)$), in which case its internal counter is incremented every time unit or its internal counter can be stopped, which is indicated by the barred variant of the stopwatch. Note that it is not possible that the action go and $reset$ take place when the internal counter is counting while $stop$ cannot happen when the internal counter is stopped. The counter contains also other internal design choices. One of them is that go and $stop$ can succeed each other infinitely fast, which may be hard to build in a real system.

The last example shows that it is useful to have a notation for the solution of a guarded recursive specification. We therefore introduce the following angular bracket notation.

Notation 2.7. Let E be a guarded recursive specification in which a variable x occurs. The (unique) solution for x in E is written as $< x|E >$.

The construct $< x|E >$ may occur as a process term. It will be treated as a constant. So the process term $a \cdot a \cdot < x|x = a\,x >$ is a valid term, representing of course a^ω. Any term not containing these constants is called *recursion free*. The fact that $< x|E >$ is the solution of x in E is expressed by the following axiom:

$$\text{REC:} \quad < x|E > = < p_x|E > \quad \text{if } x = p_x \in E.$$

$< p_x|E >$ is an abbreviation for the term p_x where every occurrence of a variable y in p_x is replaced by $< y|E >$.

With the constants $< x|E >$ RDP is not needed in the sequel. Therefore, we will not consider RDP any further. RSP, however, will play an important role. Using the notation for the solution of a guarded recursive specification, we can state RSP more precisely:

Principle 2.8. (RSP). Let E be a guarded recursive specification containing the variable x and let σ be a substitution. The following inference rule scheme is a rephrasing of RSP:

$$\text{RSP: } \frac{\sigma(E)}{\sigma(x) =< x|E >}$$

Here $\sigma(E)$ is an abbreviation for $\{\sigma(x) = \sigma(p_x)|x = p_x \in E\}$.

This rule must be read as follows. Suppose we can prove the equations in $\sigma(E)$, which means that we can show that $\sigma(x)$ is a solution for x in E. Then, RSP says that $\sigma(x)$ is equal to *the* unique solution for x in E, i.e. $< x|E >$.

We remark that RSP is not an axiom, but an inference rule scheme. For every guarded recursive specification E, substitution σ and variable x in E, there is a variant of RSP. If E consists of an infinite set of equations, then the inference rule has an infinite number of premises. Inference rules generated by RSP can occur in proofs, beside the normal rules of equational logic. If this is the case, this is written as RSP $\vdash p = q$.

Example 2.9. Suppose we have the following processes, where a^n is used as an abbreviation for n sequential occurrences of a:

$$p =< x|x = a^2x >,$$

$$q =< y|y = a^3y > .$$

We can prove using REC and RSP that $p = q$ by showing that both p and q satisfy the recursive equation $z = a^6z$. From the definitions of p and q we can derive by REC $p =< x|x = a^2x >= a^2 < x|x = a^2x >= a^4 < x|x = a^2x >= a^6 < x|x = a^2x >= a^6p$ and in the same way $q =< y|x = a^3y >= a^3 < y|x = a^3y >= a^6 < y|x = a^3y >= a^6q$. Now it can be concluded by RSP that $p =< z|z = a^6z >$ and $q =< z|z = a^6z >$. Therefore, RSP + REC $\vdash p = q$.

3 Delays

In this section we introduce several delay processes and study some of their properties. Delay processes can wait an indefinite (but sometimes bounded) amount of time before terminating. They can be used to specify that actions must happen in certain time intervals.

The *(general) delay*, which we denote by Δ, is defined as follows:

$$\Delta =< x|x = t \cdot x + \epsilon > .$$

The process Δ can always terminate or wait one time unit.

The general delay must not be confused with delays that occur elsewhere in the literature. In synchronous calculi [21, 29] Δ (also used as operator and written as $\delta(p)$ and $\Delta(p)$) is the process that can do an arbitrary number of 1 actions before terminating (or performing p). In [7, 8] Δ is used to represent divergence, i.e. the possibility to perform unbounded internal activity.

The general delay immediately suggests a variant, the *bounded delay* Δ_n. The process Δ_n can wait maximal n time units before terminating. It is defined by:

$$\Delta_n =< x_n|E >$$

where E is a recursive specification containing the equations ($n \in \mathbf{N}$):

$$\begin{aligned} x_0 &= \epsilon, \\ x_{n+1} &= t \cdot x_n + \epsilon. \end{aligned}$$

Example 3.1. We can now specify the following processes

1. $\Delta a \Delta b \Delta$,
2. $\Delta a \Delta_{15} b \Delta$.

The first process can once do an action a and then time can pass. Possibly, after some time action b can happen, but this is not necessary. In the second case action b *must* happen after the occurrence of a within at most 15 time units.

Lemma 3.2. *We have the following identities concerning* Δ *and* Δ_n*, provable using the axioms* $\mathrm{ACP}_{\tau\epsilon}^t$, REC *and* RSP*:*

1. $\Delta = \Delta + \epsilon$,
2. $\Delta\Delta = \Delta$,
3. $\tau\Delta = \Delta\tau\Delta$,
4. $\Delta_n\Delta = \Delta$.

Proof.

1. $\Delta = t\Delta + \epsilon = t\Delta + \epsilon + \epsilon = \Delta + \epsilon$.
2. Take as an equation: $X = tX + \Delta + \epsilon$. Then $\Delta\Delta$ and Δ are both solutions of this equation.

   $$\Delta\Delta = t\Delta\Delta + \Delta = t\Delta\Delta + \Delta + \epsilon$$

   $$\Delta = \Delta + \Delta = t\Delta + \epsilon + \Delta$$

   Hence, with RSP, $\Delta\Delta = \Delta$.
3. We show that $\tau\Delta$ and $\Delta\tau\Delta$ are both solutions of $X = tX + \tau\Delta$.

   $$\tau\Delta = \Delta + \tau\Delta = t\Delta + \epsilon + \tau\Delta = t\Delta + \tau\Delta = t\tau\Delta + \tau\Delta$$

   $$\Delta\tau\Delta = (t\Delta + \epsilon)\tau\Delta = t\Delta\tau\Delta + \tau\Delta$$

4. By induction on n. ($n = 0$) $\Delta_n\Delta = \epsilon\Delta = \Delta$. ($n \geq 0$) $\Delta_{n+1}\Delta = t\Delta_n\Delta + \Delta = t\Delta + \epsilon + \Delta = \Delta + \Delta = \Delta$.

□

We remark that the identity $\Delta_n \cdot \Delta_m = \Delta_{n+m}$ is not derivable. Consider for instance the case where $m = n = 1$. Then $\Delta_m \cdot \Delta_n = t(t + \epsilon) + t + \epsilon$. On the other hand $\Delta_2 = t(t + \epsilon) + \epsilon$. In the operational model, which is given in section 5, these two processes are not equal.

Lemma 3.3. *With the axioms in* $\mathrm{ACP}_{\tau\epsilon}^t$ *and* REC *we can show that* Δ *distributes over* $+$*, prefixed actions and the empty process.*

1. $(x + y) \between \Delta = x \between \Delta + y \between \Delta$,

2. $ax \parallel \Delta = a(x \parallel \Delta)$ $a \in Act_{\tau t\delta}$,

3. $\epsilon \parallel \Delta = \epsilon$.

Proof.

1. $x+y \parallel \Delta = (x+y)\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}(x+y)+(x+y) \mid \Delta = x\mathbin{\mathbb{L}}\Delta+y\mathbin{\mathbb{L}}\Delta+t\Delta\mathbin{\mathbb{L}}(x+y)+\epsilon\mathbin{\mathbb{L}}(x+y)+x \mid \Delta+y \mid \Delta = x\mathbin{\mathbb{L}}\Delta+x \mid \Delta+y\mathbin{\mathbb{L}}\Delta+y \mid \Delta = x\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}x+x \mid \Delta+\Delta\mathbin{\mathbb{L}}y+y\mathbin{\mathbb{L}}\Delta+x \mid \Delta = x \parallel \Delta+y \parallel \Delta$.

2. $ax \parallel \Delta = ax\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}ax+ax \mid t\Delta+ax \mid \epsilon = a(x \parallel \Delta) = a(x \parallel \Delta)$ for $a \in Act$,
$tx \parallel \Delta = tx\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}tx+tx \mid t\Delta = t(x \parallel \Delta)$,
$\delta x \parallel \Delta = \delta\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}\delta+\delta \mid \Delta = \delta \mid t\Delta+\delta \mid \epsilon = \delta = \delta x$,
$\tau x \parallel \Delta = \tau x\mathbin{\mathbb{L}}\Delta+\Delta\mathbin{\mathbb{L}}\tau x+\Delta \mid \tau x = \tau(x\mathbin{\mathbb{L}}\Delta)+t\Delta \mid \tau x+\epsilon \mid \tau x = \tau(x \parallel \Delta)+t\Delta \mid x+\epsilon \mid \tau x = \tau(x \parallel \Delta)+\Delta \mid x = \tau(x \parallel \Delta)$.

3. $\epsilon \parallel \Delta = \epsilon\mathbin{\mathbb{L}}\Delta+\epsilon\mathbin{\mathbb{L}}\epsilon+t\Delta\mathbin{\mathbb{L}}\epsilon+\epsilon \mid t\Delta+\epsilon \mid \epsilon = \epsilon \mid \epsilon = \epsilon$.

□

Δ can always be delayed or terminated, controlled by its environment. For instance, Δa in $\partial_{\{a,\bar{a}\}}(\Delta a \parallel \bar{a})$ with $\gamma(a,\bar{a}) = a^*$ behaves as a and therefore, the delay in Δa is forced to terminate immediately. Sometimes, one wants to describe delays that can independently of the context decide to wait or to terminate. The context has to adapt itself in this case. For this purpose *autonomous delays* Γ_0, Γ_1 and Γ_2 are introduced:

$$\begin{aligned} \Gamma_0 &= < x|x = \tau tx+\tau > \\ \Gamma_1 &= < x|x = \tau tx+\epsilon > \\ \Gamma_2 &= < x|x = tx+\tau > \end{aligned}$$

The τ's in Γ_i $(i = 0,1,2)$ indicate that by some internal activity, options to terminate or to proceed can be lost. In Γ_1, for instance, an internal τ-step makes immediate termination impossible.

The following theorem gives a number of derivable facts about the autonomous delays.

Lemma 3.4. $\mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash$

$$\begin{array}{rclcrclcrcl} \Gamma_0 &=& \Gamma_0+\tau & & \Gamma_1 &=& \Gamma_1+\epsilon & & \Gamma_2 &=& \Gamma_2+\tau \\ \Gamma_0\Gamma_0 &=& \tau\Gamma_0 & & \Gamma_1\Gamma_1 &=& \Gamma_1 & & \Gamma_2\Gamma_2 &=& \tau\Gamma_2 \\ \tau\Gamma_0 &=& \Gamma_0\tau\Gamma_0 & & \tau\Gamma_1 &=& \Gamma_1\tau\Gamma_1 & & \tau\Gamma_2 &=& \Gamma_2\tau\Gamma_2 \\ \Delta \parallel \Gamma_0 &=& \Gamma_0 & & \Delta \parallel \Gamma_1 &=& \Gamma_1 & & \Delta \parallel \Gamma_2 &=& \Gamma_2 \end{array}$$

Proof. The proofs have the same structure as the proof of lemma 3.2 □

It is possible to give the finite versions of the autonomous delays also. Here, we only define Γ^n_0 as the finite variant of Γ_0 which will be used in the next example.

$$\Gamma^n_0 = < x_n|E >$$

where E consists of the equations $(i \in \mathbf{N})$:

$$\begin{aligned} x_0 &= \epsilon, \\ x_{i+1} &= \tau t x_i+\tau. \end{aligned}$$

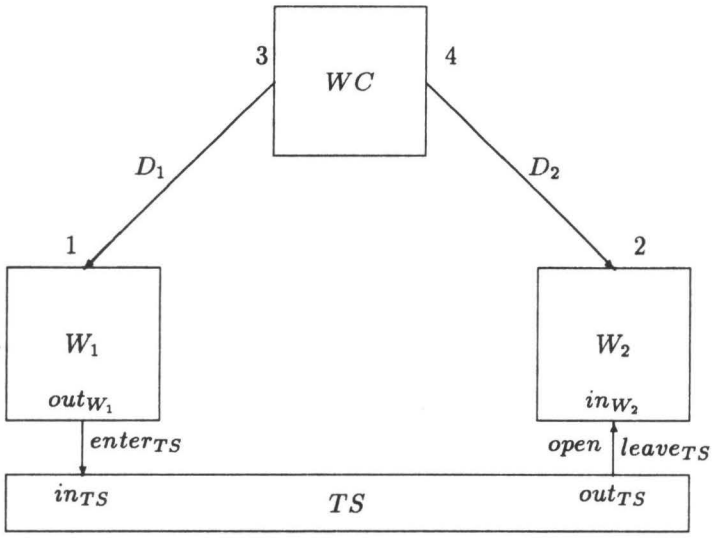

Figure 1.

4 Example

In this section a *manufacturing workcell* is presented that shows how time can be used to describe and verify timed systems. In the system it is important that certain actions happen just because time has passed. Further, there are actions that cannot be guaranteed to happen at predefined times. In the system a datalink can deliver messages in time, but the exact time the delivery takes is not known.

The workcell is structured as described in [9]. It consists of two *workstations*, called W_1 and W_2, which actually produce products, and a *transport system*, TS, that transports products from workstation W_1 to W_2. In this case TS is thought to be a simple conveyor belt which can only carry one product at a time. The last main component of the workcell is a *workcell controller* (WC) that coordinates all activities in the workcell.

Workcell W_1 needs one time unit to produce a product. Then the transport system takes exactly three time units to transfer this product to workcell W_2. Products that arrive at W_2 can enter W_2 by a small gate that can be opened and closed by W_2. Whenever a product arrives, it should always be open. It is assumed that this gate closes automatically when a product has entered W_2. This is not explicitly specified. The workcell controller is connected with the workstations by two asynchronous point to point datalinks D_1 and D_2 through which it can send coordinating messages to the workcells. D_1 connects WC with W_1 and D_2 connects WC with W_2. Datalink D_2 delivers its messages exactly in one time unit. Datalink D_1 autonomously decides to do it in in 0,1 or 2 time units. The workcell controller first sends a message to W_1 to say that it must send a product to W_2 using the transport system. After some time it sends a message to W_2 saying that it must open its gate because a product is arriving. This is done in such a way that the gate is only open as short as possible. As the control system of this workcell is fairly primitive and there are uncertainties in the delivery time of datalink D_1, the transport system is not optimally used.

The system is given as the solution of the following equations. See also figure 1. Capitals are used as variables for readability.

$$S = \partial_H(W_1 \parallel W_2 \parallel WC \parallel TS \parallel D_1 \parallel D_2)$$

with $H = \{r_i, s_i | i = 1,2,3,4\} \cup \{out_{W_1}, in_{W_2}, in_{TS}, out_{TS}\}$ and $\gamma(r_i, s_i) = c_i$, $\gamma(out_{W_1}, in_{TS}) = enter_{TS}$, $\gamma(out_{TS}, in_{W_2}) = leave_{TS}$. The actions r_i, s_i and c_i describe respectively a receive, a send

and a resulting communication at port i (see the numbers in figure 1). The other action names are self-explaining.

$$\begin{aligned} W_1 &= \Delta r_1 t\, out_{W_1} W_1 \\ W_2 &= \Delta r_2 t\, open \Delta in_{W_2} W_2 \\ WC &= s_3 t^2 s_4 t^4 WC \\ TS &= \Delta in_{TS} t^3 out_{TS} TS \\ D_1 &= \Delta r_3 \Gamma_0^2 s_1 D_1 \\ D_2 &= \Delta r_4 t\, s_2 D_2 \end{aligned}$$

The behavior of the whole system is expected to be the following: The workcell controller sends a message to W_1. After receipt of this message a product is put on the conveyor belt. While it is being transported to W_2 the workcell controller sends a message to W_2 saying that it must open the gate. Then, the product arrives at W_2 and the whole process starts over again. We are especially interested how long the gate is open before the arrival of a product. This can be studied by hiding all actions in S except *open* and $leave_{TS}$. Hence, we are interested in the behavior of

$$\tau_I(S)$$

where $I = \{c_i | i = 1,2,3,4\} \cup \{enter_{TS}\}$. We will only write down the result of the verification. The verification itself is straightforward. $\tau_I(S)$ is a solution for U_1 in the following set of guarded equations.

$$\begin{aligned} U_1 &= \tau \cdot (\tau \cdot U_3 + \tau \cdot U_4), \\ U_2 &= leave_{TS} \cdot U_1 + \tau \cdot leave_{TS} \cdot U_3 + \tau \cdot leave_{TS} \cdot U_4, \\ U_3 &= t \cdot (\tau t^3 \cdot open \cdot t^2 \cdot U_2 + \tau \cdot t^3 \cdot open \cdot t \cdot leave_{TS} \cdot t \cdot U_1), \\ U_4 &= t^4 \cdot open \cdot leave_{TS} \cdot t^2 \cdot U_1. \end{aligned}$$

We see that the system does not contain any deadlocks and the gate is open for at most two time units. But the behavior of the system is more complicated than expected. Only if one looks precisely at the specification, one can see that there is a good reason for this. Consider the case where delivery of a message in D_1 takes two time units. Then the system arrives at U_2 via U_3. In U_2 the situation is that a product is ready to enter W_2 and WC has just issued a new message via D_1 to W_1. Now there are three actions that can happen. The product can enter W_2 or D_1 can decide to deliver the message in 0 or in more than 0 time units. In the last case it is known that the next product will arrive after more than four time units (second option in U_2) or after exactly four time units (third option) when $leave_{TS}$ takes place. Therefore, U_2 cannot be identified with $leave_{TS} \cdot U_1$ and thus all equations are necessary. We need the axiom $abx + aby = a(bx + by)$ valid in ready trace semantics [13] to prove $\tau_I(S)$ equal to S_2, defined by the equation:

$$S_2 = \tau(t^4 open\, leave_{TS} t^2 + t^4 open\, t\, leave_{TS} t + t^4 open\, t^2 leave_{TS})\, S_2.$$

The ready trace axiom is not valid in weak bisimulation semantics which will be introduced in the next section, but it respects deadlock behavior.

We can try to improve the performance of the workcell by letting the workcell controller issue its commands faster. Define a new workcell controller and a workcell S' by:

$$\begin{aligned} S' &= \partial_H(W_1 \parallel W_2 \parallel WC' \parallel TS \parallel D_1 \parallel D_2), \\ WC' &= s_3 t^2 s_4 t^2 WC'. \end{aligned}$$

H, γ are the same as above. We are now only interested in the time consistency of WC'. Therefore all actions are hidden. It turns out that $\tau_J(WC')$ ($J = I \cup \{leave_{TS}, open\}$) is the solution for X of the following equations (again the law $abx + aby = a(bx + by)$ is used).

$$\begin{aligned} X &= \tau \cdot t^4 \cdot (X + Y), \\ Y &= \tau \cdot (t \cdot \delta + t^4 \cdot X). \end{aligned}$$

A time deadlock occurs in Y. This means that the time constraints of the components of S' were incompatible. And indeed, in S' it is possible that W_1 must put a product on the conveyor belt, while the transport system TS still needs one time unit before it is capable to accept this product from W_1. As in any implementation time cannot be blocked, the new workcell cannot be built.

5 An operational semantics for $\mathrm{ACP}^t_{\tau\epsilon}$

We give an operational semantics to $\mathrm{ACP}^t_{\tau\epsilon}$ which corresponds to a way $\mathrm{ACP}^t_{\tau\epsilon}$-terms can be executed. In this way we show directly how $\mathrm{ACP}^t_{\tau\epsilon}$-terms can be seen as processes. The completeness of the axioms with respect to the operational semantics will be shown. Thus, there is a direct correspondence between the two. This means that the axioms indeed capture the idea of processes.

The semantics of an $\mathrm{ACP}^t_{\tau\epsilon}$-term is given using a transition relation $\longrightarrow$ that defines the behavior of terms. If process p can perform action a then this is denoted by a transition $p \xrightarrow{a} q$. The action a is called the *label* of the transition and the term q represents the resulting behavior. The transition relation is defined by the *Transition System Specification* (TSS) [17] in table 4. The rules R1-R12.3 are inference rules. All transitions between closed $\mathrm{ACP}^t_{\tau\epsilon}$-terms that are derivable using these rules are in the transition relation $\longrightarrow$.

Labels in the transition relation are chosen from $Act_{t\tau\surd} = Act \cup \{t, \tau, \surd\}$. The label $\surd$ denotes termination, t represents the proceeding of a time unit and τ the occurrence of an internal action. In table 4 a and b range over $Act_{t\tau\surd}$ and c ranges over Act unless explicitly stated otherwise. x, y, z are variables.

Only some rules are explained here. For the others we refer to [12, 17] in particular for rules 12.2 and 12.3. Rule 7.2 is introduced to allow $p \mid q$ to terminate if both p and q can terminate, and to make it possible that if τ's can occur as initial steps in both p and q, then $p \mid q$ can perform an initial τ-step. This is introduced for the proof of lemma 6.6. Note that this only influences the behavior of the communication merge, which is just an auxiliary operator. Rule 5.4 indicates that t-actions in both sides of the parallel operator can communicate.

The behavior of a term is now given by the transitions it can perform, whereby the names of intermediate processes are of no interest. However, this is not yet satisfying. Consider for example the transitions belonging to the expressions a and $a\epsilon + a$, depicted in figure 2. The forgotten names of the intermediate states are put between brackets. We can see that the transition system (TS) of $a + a\epsilon$ is shaped as two transition systems for a. Operationally, it does not make any difference whether the a-side or the $a\epsilon$-side is executed. Therefore, we would like to equate the TS of $a + a\epsilon$ such that it becomes equal to the TS for a.

In order to obtain this, we consider strong bisimulation equivalent process terms as equal. Strong bisimulation equivalence is chosen as it is the coarsest relation that does not alter the operational behavior of a transition system. Due to the rules 12.1-12.3 this captures exactly weak bisimulation.

Definition 5.1. A relation $R \subseteq$ between $\mathrm{ACP}^t_{\tau\epsilon}$-terms is called a *bisimulation relation* if it satisfies the *transfer property*, i.e.:

1. if $p\,R\,q$ and $p \xrightarrow{a} p'$ for $a \in Act_{t\tau\surd}$ then $\exists q'$ such that $q \xrightarrow{a} q'$ and $p'\,R\,q'$,
2. if $p\,R\,q$ and $q \xrightarrow{a} q'$ for $a \in Act_{t\tau\surd}$ then $\exists p'$ such that $p \xrightarrow{a} p'$ and $p'\,R\,q'$.

ϵ :	R1:	$\epsilon \xrightarrow{\surd} \delta$		
$a \in Act_{t\tau}$:	R2:	$a \xrightarrow{a} \epsilon$		
$+$:	R3.1:	$\frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}$	R3.2:	$\frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$
$\cdot$:	R4.1:	$\frac{x \xrightarrow{a} x'}{xy \xrightarrow{a} x'y}$ if $a \in Act_{t\tau}$	R4.2:	$\frac{x \xrightarrow{\surd} x' \quad y \xrightarrow{a} y'}{xy \xrightarrow{a} y'}$
$\|$:	R5.1:	$\frac{x \xrightarrow{a} x'}{x\|y \xrightarrow{a} x'\|y}$ if $a \in Act_{\tau}$	R5.2:	$\frac{y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x\|y'}$ if $a \in Act_{\tau}$
	R5.3:	$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x\|y \xrightarrow{c} x'\|y'}$ if $\gamma(a,b) = c$	R5.4:	$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x'\|y'}$ if $a = t, \surd$
$\lfloor\!\lfloor$:	R6:	$\frac{x \xrightarrow{a} x'}{x \lfloor\!\lfloor y \xrightarrow{a} x'\|y}$ if $a \in Act_{\tau}$		
$\|$:	R7.1:	$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{b} y'}{x\|y \xrightarrow{c} x'\|y'}$ if $\gamma(a,b) = c$	R7.2:	$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x\|y \xrightarrow{a} x'\|y'}$ if $a = t, \tau, \surd$
∂_H :	R9:	$\frac{x \xrightarrow{a} x'}{\partial_H(x) \xrightarrow{a} \partial_H(x')}$ if $a \notin H$		
τ_I :	R10.1	$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{a} \tau_I(x')}$ if $a \notin I$	R10.2:	$\frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}$ if $a \in I$
recursion:	R11:	$\frac{<p_x\|E> \xrightarrow{a} y}{<x\|E> \xrightarrow{a} y}$ if $x = p_x \in E$		
τ-laws:	R12.1:	$a \xrightarrow{a} \tau$ if $a \in Act_{t\tau}$	R12.2:	$\frac{x \xrightarrow{\tau} y \quad y \xrightarrow{a} z}{x \xrightarrow{a} z}$
	R12.3:	$\frac{x \xrightarrow{a} y \quad y \xrightarrow{\tau} z}{x \xrightarrow{a} z}$		

Table 4.

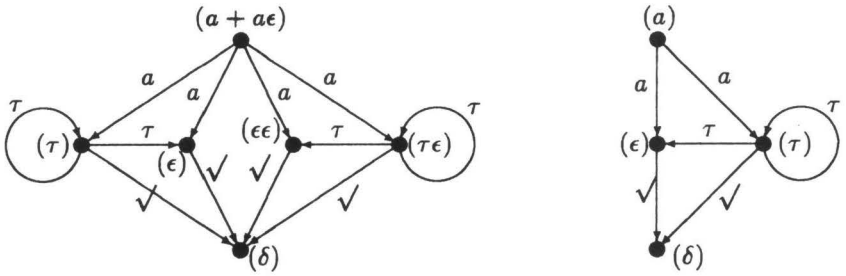

Figure 2.

We say that two terms p and q are bisimilar, notation $p \leftrightarrows q$, if there exists a bisimulation relation containing the pair (p, q).

The transition system specification in table 4 is in *tyft/tyxt*-format [17]. This immediately implies that $\leftrightarrows$ is a congruence relation.

Soundness and completeness of the axioms will be shown in a number of steps. First it is observed that every recursion free closed process expression is equal to a term consisting of the constants of $\mathrm{ACP}^t_{\tau\epsilon}$ and sequential and alternative composition. These expressions are called *basic terms*. Then, using the notions of *operational completeness* and *operational soundness* these expressions are related to transitions. Completeness of the proof system for recursion free closed terms then follows immediately.

Lemma 5.2. *(Soundness of $\mathrm{ACP}^t_{\tau\epsilon}$) Let p and q be closed process $\mathrm{ACP}^t_{\tau\epsilon}$-expressions.*

$$\mathrm{ACP}^t_{\tau\epsilon} + \mathrm{RSP} + \mathrm{REC} \vdash p = q \Rightarrow p \leftrightarrows q.$$

Proof. Long but straightforward. □

Definition 5.3. The set of *basic terms* over $\mathrm{ACP}^t_{\tau\epsilon}$ is the smallest set of $\mathrm{ACP}^t_{\tau\epsilon}$-expressions satisfying:

- δ and ϵ are basic terms,
- if p is a basic term, then ap $(a \in Act_{t\tau})$ is a basic term,
- if p, p' are basic terms, then $p + p'$ is a basic term.

Lemma 5.4. *Let p be a basic term over $\mathrm{ACP}^t_{\tau\epsilon}$. If $p \xrightarrow{a} p'$ $(a \in Act_{t\tau})$, then $p' = \epsilon \cdot q$ or $p' = \tau \cdot q$ for some basic term q. Moreover, p' contains at most as many function names as p.*
Proof. Use induction on the length of the derivation of $p \xrightarrow{a} p'$. □

Lemma 5.5. *Let p, p' be basic terms over $\mathrm{ACP}^t_{\tau\epsilon}$. Then there is a basic term q such that:*

1. $\mathrm{ACP}^t_{\tau\epsilon} \vdash p \square p' = q$ $(\square = +, \cdot, \between, |)$,
2. $\mathrm{ACP}^t_{\tau\epsilon} \vdash \square(p) = q$ $(\square = \tau_I, \partial_H)$.

Proof. Use induction on the structure of basic terms. $+$ is trivial. τ_I and ∂_H are straightforward. For $\cdot$ use induction on its left hand side. $\between$ and $|$ are proved simultaneously with induction on the number of symbols in p and q. For $\between$ we only explore the left hand side. The basic cases are:
$\delta \between p = \epsilon \between p = \delta$, $\delta \mid \delta = \delta \mid \epsilon = \epsilon \mid \delta = \delta$, $\epsilon \mid \epsilon = \epsilon$.
The inductive cases are given by:
$(p + p') \between p'' = p \between p'' + p' \between p''$. With induction we know that there are basic terms q, q' such that $p \between p'' = q$ and $p' \between p'' = q'$ are provable from $\mathrm{ACP}^t_{\tau\epsilon}$. Now $p \between p'' + p' \between p'' = q + q'$ which is a basic term.
$tp \between p' = \delta$ which is a basic term.
$ap \between p' = a(p \between p' + p' \between p + p \mid p') = a(q + q' + q'')$ with $(a \in Act_\tau)$ q, q', q'' basic terms and therefore $ap \between p'$ is provable equal to a basic term.
$(p + p') \mid p'' = p \mid p'' + p' \mid p'' = q + q'$ with q, q' basic terms, is a basic term. Using the commutativity of the communication merge it also follows that $p \mid (p' + p'')$ is provably equal to a basic term. In the following lines, we will not repeat this commutativity argument again, but only treat one case.
Now we assume that neither the left nor the right hand side of $|$ contain as outermost symbol $+$ or δ. Then we have:

$ap \mid bp' = (a \mid b)(p \between p' + p' \between p + p \mid p') = \gamma(a,b)(q + q' + q'')$ $(a, b \in Act)$ for basic terms q, q', q'' if γ is defined.
$ap \mid bp' = \delta$ if $\gamma(a,b)$ is not defined.
$\epsilon \mid ax = \delta$, $\epsilon \mid tx = \delta$, $\epsilon \mid \tau x = \delta$.
$ap \mid \tau p' = ap \mid p'$.
$tp \mid \tau p' = tp \mid p'$, $\tau p \mid \tau p' = \tau(p \between p' + p' \between p + p \mid p')$ which can again be proved equal to a basic term.
$tp \mid ap' = \delta$ $(a \in Act_\delta)$.
$tp \mid tp' = t(p \between p' + p' \between p + p \mid p')$ which is with induction provable equal to basic terms. □

Theorem 5.6. *Let p be a recursion free closed $\mathrm{ACP}^t_{\tau\epsilon}$-expression. Then there is a basic term q such that:*

$$\mathrm{ACP}^t_{\tau\epsilon} \vdash p = q.$$

Proof. First eliminate all occurrences of $\|$ from p using EM1. Apply induction on the newly obtained term p'. If $p' = \epsilon, \delta, a(\in Act), t, \tau$ then the basic terms are respectively: $\epsilon, \delta, a\epsilon, t\epsilon$ and $\tau\epsilon$. For binary operators $\square$ $(\square = +, \cdot, \between, \mid)$ $p = p' \square p''$ it follows with induction that p' and p'' are provably equal to basic terms q', q''. Then lemma 5.5 yields $q' \square q'' = q$ with q a basic term. For the unary operators a similar argument can be applied. □

The following notation is an abbreviation that turns out to be very useful. See for some examples for instance [15].

Notation 5.7. (*Summand inclusion*). Let p, p' be $\mathrm{ACP}^t_{\tau\epsilon}$-terms. We write $p \subseteq p'$ for $p + p' = p'$.

The following lemmas relate summand inclusion to the operational rules in table 4. They state that if a process p can perform an a-step ($p \xrightarrow{a} p'$) then it is provable that ap' is a summand of p. A weak variant of the converse also holds.

Lemma 5.8. *(Operational soundness) Let p, p' be $\mathrm{ACP}^t_{\tau\epsilon}$-terms and let $a \in Act_{t\tau}$:*

$$\mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} \vdash a.p' \subseteq p \;\Rightarrow\; \exists p''\ p \xrightarrow{a} p'' \text{ and } p'' \leftrightarrows p',$$

$$\mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} \vdash \epsilon \subseteq p \;\Rightarrow\; \exists p'\ p \xrightarrow{\surd} p'.$$

Proof. Direct using the soundness lemma 5.2 □

Lemma 5.9. *(Operational completeness) Let p, p' be $\mathrm{ACP}^t_{\tau\epsilon}$-terms and let $a \in Act_{t\tau}$:*

$$p \xrightarrow{a} p' \;\Rightarrow\; \mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} \vdash ap' \subseteq p,$$

$$p \xrightarrow{\surd} p' \;\Rightarrow\; \mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} \vdash \epsilon \subseteq p.$$

Proof. Straightforward induction on the proof of $p \xrightarrow{a} p'$ and $p \xrightarrow{\surd} p'$. □

Notation 5.10. Let p and q be $\mathrm{ACP}^t_{\tau\epsilon}$-terms. $p \rightleftarrows q$ stands for: if $p \xrightarrow{a} p' \;\Rightarrow\; \exists q'\ q \xrightarrow{a} q'$ and $p' \leftrightarrows q'$. $p \leftleftarrows q$ is used to abbreviate the converse. Note that the notation introduced here resembles respectively clause 1 and 2 in the definition of bisimulation.

Lemma 5.11. *Let p and q be basic terms over $\mathrm{ACP}^t_{\tau\epsilon}$. Then:*

1. *If $p \rightleftarrows q$ then $\mathrm{ACP}^t_{\tau\epsilon} \vdash p \subseteq q$,*
2. *If $p \leftleftarrows q$ then $\mathrm{ACP}^t_{\tau\epsilon} \vdash p \supseteq q$,*

3. If $p \leftrightarrows q$ *then* $\mathrm{ACP}^t_{\tau\epsilon} \vdash p = q$.

Proof. We use induction on the structure of both p and q, simultaneously. The proof employs the operational sound and completeness lemmas.

Basis. First 1 is proved. Suppose that $p = \epsilon$. $\epsilon \rightleftarrows q \Rightarrow q \xrightarrow{\surd} q' \Rightarrow \mathrm{ACP}^t_{\tau\epsilon} \vdash \epsilon \subseteq q$. Suppose $p = \delta$. This case is trivial using axiom A6. 2 follows in the same way. 3 follows directly using 1 and 2.
Induction. First consider 1. Suppose $p = p_1 + p_2 \rightleftarrows q$. This implies that $p_1 \rightleftarrows q$ and $p_2 \rightleftarrows q$. Using 1 inductively yields: $\mathrm{ACP}^t_{\tau\epsilon} \vdash p_1 \subseteq q$ and $\mathrm{ACP}^t_{\tau\epsilon} \vdash p_2 \subseteq q$. Now using axiom A1 leads to $\mathrm{ACP}^t_{\tau\epsilon} \vdash p_1 + p_2 \subseteq q$. Suppose that $p = ap_1 \rightleftarrows q$. Note that p_1 has a simpler structure than p. There is a p_2 such that $p \xrightarrow{a} p_2 \leftrightarrows p_1$. By bisimulation it follows that there is a q_1 such that $q \xrightarrow{a} q_1$, $p_1 \leftrightarrows p_2 \leftrightarrows q_1$. By lemma 5.4 q_1 has the form $\tau \cdot r$ or $\epsilon \cdot r$ with r a basic term. Define a basic term q_2 to be $\tau \cdot r$ or r, respectively. Note that $p_1 \leftrightarrows q_2$ and q_2 does not contain more function symbols than q. With the induction hypothesis conclude that $\mathrm{ACP}^t_{\tau\epsilon} \vdash p_1 = q_2$. By operational completeness we can show that $\mathrm{ACP}^t_{\tau\epsilon} \vdash aq_1 \subseteq q$. Therefore, $\mathrm{ACP}^t_{\tau\epsilon} \vdash aq_2 \subseteq q$ and thus $\mathrm{ACP}^t_{\tau\epsilon} \vdash ap_1 \subseteq q$. 2 follows in the same way. In case 3 we can conclude from 1, 2 and $p \leftrightarrows q$ that $\mathrm{ACP}^t_{\tau\epsilon} \vdash p \subseteq q$ and $\mathrm{ACP}^t_{\tau\epsilon} \vdash q \subseteq p$. Hence, $\mathrm{ACP}^t_{\tau\epsilon} \vdash p = q$. □

Theorem 5.12. *(Completeness) Let* p *and* q *be recursion free closed process* $\mathrm{ACP}^t_{\tau\epsilon}$*-expressions. It holds that:*

$$p \leftrightarrows q \quad \Leftrightarrow \quad \mathrm{ACP}^t_{\tau\epsilon} \vdash p = q$$

Proof. $\Leftarrow$ follows immediately from 5.2 $\Rightarrow$ can be proved as follows. Suppose $p \leftrightarrows q$. Then there are basic terms p' and q' that are provably equivalent to p and q. With soundness it follows that $p' \leftrightarrows q'$. An application of the previous lemma yields $\mathrm{ACP}^t_{\tau\epsilon} \vdash p' = q'$. Connecting all parts completes the proof. □

The completeness proof only applies to recursion free, closed terms. The proof can be extended to closed terms, using the power of RSP, but we think that this deviates too much from the main purpose of this paper.

6 Relating $\mathrm{ACP}_{\tau\epsilon}$ to $\mathrm{ACP}^t_{\tau\epsilon}$

Real time specifications contain more information than specifications in $\mathrm{ACP}_{\tau\epsilon}$. One is not always interested in the time information and therefore, one sometimes likes to work in the setting of $\mathrm{ACP}_{\tau\epsilon}$. However, it is interesting to know what an $\mathrm{ACP}_{\tau\epsilon}$ process means in terms of the timing of $\mathrm{ACP}^t_{\tau\epsilon}$, for instance when specifications where time does not play a role are combined with specifications where time is important. Here we give a translation of $\mathrm{ACP}_{\tau\epsilon}$ processes into $\mathrm{ACP}^t_{\tau\epsilon}$, thereby fixing an intuition about time in $\mathrm{ACP}_{\tau\epsilon}$.

The sequential composition in $\mathrm{ACP}_{\tau\epsilon}$ describes that its left argument must happen before its right argument with an arbitrary delay in between. We use Δ to model this delay. Moreover, a general delay is put in front and after every process to indicate that it may start after an undefined amount of time and that time can proceed after the process has been terminated. E.g. a is translated to $\Delta a \Delta$ and $a \cdot b$ is translated to $\Delta a \Delta b \Delta$. Note that somehow this intuition conflicts with the intuition adopted when testing processes [10] where the delay between actions that are not blocked, is bounded.

For more complex processes the translation is slightly more complicated. The translation of $a + b$ is $\Delta(a\Delta + b\Delta)$ as the choice between a and b is externally determinable. In order to achieve this, the translation is divided into two parts, T_1 and T_2. T_1 is used to put a Δ in front of the actions while T_2

translates a term in such a way that first an action $a(\neq t)$ must happen. There is one exception to this rule. The T_2-translation of τ is $\tau\Delta$ and can therefore start with a t action.

Definition 6.1. The *real time translations* T_1 and T_2 from closed $\mathrm{ACP}_{\tau\epsilon}$ expressions to closed $\mathrm{ACP}^t_{\tau\epsilon}$ expressions are defined by:

$$T_1(p) = \Delta \cdot T_2(p) \text{ with } p \text{ and } \mathrm{ACP}_{\tau\epsilon}\text{-term},$$

$$\begin{aligned}
&T_2(a) = a\Delta \ (a \in Act_\tau),\\
&T_2(\delta) = \delta,\\
&T_2(\epsilon) = \epsilon,\\
&T_2(x + y) = T_2(x) + T_2(y),\\
&T_2(x \cdot y) = T_2(x) \cdot T_2(y),\\
&T_2(x \parallel y) = T_2(x \between y + y \between x + x \mid y),\\
&T_2(x \between y) = T_2(x) \between T_1(y),\\
&T_2(x \mid y) = T_2(x) \mid T_2(y),\\
&T_2(\partial_H(x)) = \partial_H(T_2(x)),\\
&T_2(\tau_I(x)) = \tau_I(T_2(x)),\\
&T_2(< x|E >) =< x|T_2(E) >,\\
&T_2(x) = x.
\end{aligned}$$

For a guarded recursive specification E, $T_2(E)$ is defined as:

$$T_2(E) = \{x = T_2(p_x) | x = p_x \in E\}$$

The following lemmas are used to prove that the T_1 and T_2-translations of identities derivable in $\mathrm{ACP}_{\tau\epsilon}$ are also valid and derivable in $\mathrm{ACP}^t_{\tau\epsilon}$.

Definition 6.2. Let σ be a substitution, mapping variables to $\mathrm{ACP}^t_{\tau\epsilon}$-terms. We define the substitution σ_{T_2} by (x is a variable):

$$\sigma_{T_2}(x) = T_2(\sigma(x)).$$

Lemma 6.3. *Let p be an $\mathrm{ACP}^t_{\tau\epsilon}$ expression and let σ be a substitution:*

$$T_2(\sigma(p)) = \sigma_{T_2}(T_2(p)).$$

Proof. Use induction on the structure of p. □

Lemma 6.4. *We have the following fact:*

$$\mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash T_1(x) \parallel T_1(y) = \Delta T_2(x \parallel y)$$

Proof. It is shown that both sides satisfy the equation $X = tX + T_2(x \parallel y)$.
$T_1(x) \parallel T_1(y) = T_1(x) \between T_1(y) + T_1(y) \between T_1(x) + T_1(x) \mid T_1(y) =$
$\Delta T_2(x) \between T_1(y) + \Delta T_2(y) \between T_1(x) + \Delta T_2(x) \mid \Delta T_2(y) =$
$T_2(x) \between T_1(y) + T_2(y) \between T_1(x) + t(T_1(x) \parallel T_1(y)) + T_2(x) \mid T_2(y) =$
$T_2(x \parallel y) + t(T_1(x) \parallel T_1(y))$

$\Delta T_2(x \parallel y) = t\Delta T_2(x \parallel y) + T_2(x \parallel y)$ □

Lemma 6.5. *Let p, p' be $\mathrm{ACP}_{\tau\epsilon}$ expressions. Then:*

$$\mathrm{ACP}_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash p = p' \Rightarrow \mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash T_2(p) = T_2(p')$$

Proof. This lemma will be proved on the depth of the proof of $p = p'$. As a basic case $p = p'$ can be an instantiation of REC or an axiom $\mathrm{ACP}_{\tau\epsilon}$ or it can be an equational inference rule without premisses. Checking the axioms A1,..,A9 is completely trivial and therefore left out. Checking the reflexivity rule ($p = p$) of equational reasoning is also straightforward.
T1: $T_2(a\tau) = T_2(a)T_2(\tau) = a\Delta\tau\Delta = a\tau\Delta = a\Delta = T_2(a)$
T2: $T_2(\tau + \epsilon) = T_2(\tau) + T_2(\epsilon) = \tau\Delta + \epsilon = (\tau + \epsilon)(\Delta + \epsilon) + \epsilon = \tau(\Delta + \epsilon) + \Delta + \epsilon + \epsilon = \tau\Delta = T_2(\tau)$
T3: $T_2(a(\tau x + y)) = a\Delta(\tau\Delta T_2(x) + T_2(y)) = a(t\Delta(\tau\Delta T_2(x) + T_2(y)) + \tau\Delta T_2(x) + T_2(y)) =$
$= T_2(a(\tau x + y)) + a\Delta T_2(x) = T_2(a(\tau x + y)) + ax)$
EM1: trivial
EM2: $T_2(\epsilon \between x) = T_2(\epsilon) \between T_1(x) = \epsilon \between T_1(x) = \delta = T_2(\delta)$
EM3: $T_2(ax \between y) = a\Delta T_2(x) \between T_1(y) = a(\Delta T_2(x) \parallel T_1(y)) = a(T_1(x) \parallel T_1(y)) = a\Delta T_2(x \parallel y) = T_2(a(x \parallel y))$
EM4: $T_2((x + y) \between z) = (T_2(x) + T_2(y)) \between T_1(z) = T_2(x) \between T_1(z) + T_2(y) \between T_1(z) = T_2(x \between z + y \between z)$
EM5: trivial
EM6: $T_2(\epsilon \mid \epsilon) = \epsilon \mid \epsilon = \epsilon = T_2(\epsilon)$
EM7: $T_2(\epsilon \mid ax) = \epsilon \mid T_2(ax) = \delta = T_2(\delta)$
EM8: $T_2(ax \mid by) = a\Delta T_2(x) \mid b\Delta T_2(y) = (a \mid b)(\Delta T_2(x) \parallel \Delta T_2(y)) = (a \mid b)\Delta T_2(x \parallel y) = T_2(\gamma(a, b)(x \parallel y))$ if $\gamma(a, b)$ defined. Otherwise, $\delta = T_2(\delta)$ results.
EM9: trivial
EM10: $T_2(\tau x \between y) = \tau\Delta T_2(x) \between T_1(y) = \tau(\Delta T_2(x) \parallel T_1(y)) = \tau\Delta T_2(x \parallel y) = T_2(\tau(x \parallel y))$
EM11: $T_2(\epsilon \mid \tau x) = \epsilon \mid \tau\Delta T_2(x) = \epsilon \mid (t\Delta T_2(x) + T_2(x)) = \epsilon \mid T_2(x) = T_2(\epsilon \mid x)$
EM12: $T_2(\tau x \mid ay) = \tau\Delta T_2(x) \mid a\Delta T_2(y) = \Delta T_2(x) \mid a\Delta T_2(y) = T_2(x) \mid T_2(ay) + tT_2(x) \mid a\Delta T_2(y) = T_2(x) \mid T_2(ay) = T_2(x \mid ay)$
EM13: $T_2(\tau x \mid \tau y) = \tau\Delta T_2(x) \mid \tau\Delta T_2(y) = \tau(\Delta T_2(x) \mid \Delta T_2(y)) = \tau\Delta T_2(x \parallel y) = T_2(\tau(x \parallel y))$
Checking D1-D4,DT,DE,TE,TI1-TI5 goes in the same way.
REC: $T_2(< X|E >) =< X|T_2(E) >=< T_2(p_X)|T_2(E) >= T_2(< p_X|E >)$

Now we check the inference rules. As they all go in the same way we only consider RSP and the congruence rule for the parallel operator.

Suppose RSP is the last inference rule used to conclude that $\sigma(x) =< x|E >$. We must show that we can find a proof for $T_2(\sigma(x)) = T_2(< x|E >)$, which is by definition equal to $\sigma_{T_2}(x) =< x|T_2(E) >$. By induction there is a proof for $T_2(\sigma(E))$ which is, using lemma 6.3, the same as $\sigma_{T_2}(T_2(E))$. By applying RSP it follows that $\sigma_{T_2}(x) =< x|T_2(E) >$, which is exactly what we had to prove.

Suppose the last inference rule used is

$$\frac{p_1 = q_1 \quad p_2 = q_2}{p_1 \parallel p_2 = q_1 \parallel q_2}.$$

By induction we have a proof for $T_2(p_1) = T_2(q_1)$ and $T_2(p_2) = T_2(q_2)$. From this we can derive $T_2(p_1) \between \Delta T_2(p_2) + T_2(p_2) \between \Delta T_2(p_1) + T_2(p_1) \mid T_2(p_2) = T_2(q_1) \between \Delta T_2(q_2) + T_2(q_2) \between \Delta T_2(q_1) + T_2(q_1) \mid T_2(q_2)$. This is exactly equal to $T_2(p_1 \parallel p_2) = T_2(q_1 \parallel q_2)$, which we had to prove. □

Theorem 6.6. *Let p, p' be $\mathrm{ACP}_{\tau\epsilon}$-expressions. Then:*

$$\mathrm{ACP}_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash p = p' \Rightarrow \mathrm{ACP}^t_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash T_1(p) = T_1(p')$$

Proof. From $\mathrm{ACP}_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash p = p'$ it follows by lemma 6.5 that $\mathrm{ACP}_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash T_2(p) = T_2(p')$. By placing a general delay for $T_2(p)$ and $T_2(p')$ it follows that $\mathrm{ACP}_{\tau\epsilon} + \mathrm{REC} + \mathrm{RSP} \vdash T_1(p) = T_1(p')$. □

We are now able to explain why theorem 6.6 requires an unusual approach to the communication merge. The reason can be found in the T_2-translation of the term $(\tau a \mid \tau b)$ which is $(\tau \Delta a \Delta \mid \tau \Delta b \Delta)$. If we adopt the traditional law for τ and $\mid$, i.e. $\tau x \mid y = x \mid y$ [6], which holds when dropping τ in rule 7.2 in table 4, we would be able to prove $(\tau a \mid \tau b) = a \mid b$. But $T_2(a \mid b) \neq T_2(\tau a \mid \tau b)$, as the right hand side can do a t step while the left hand side cannot. So, without the axioms for the communication merge, theorem 6.6 does not hold.

7 Concluding remarks

This article presented a process algebra that contains the basic features to reason about distributed, real time systems. Comparing this work with other timed process algebras and studying possible extensions should reveal the weaker and stronger points of different approaches. It is our hope that this will lead to a common and generally applicable algebra.

Although we cannot possibly be complete, we mention some work in which timed process algebras are introduced. In 1983 MILNER presented SCCS [21] in which all actions take unit time. Although time is important in this work, we think that a unit time duration of actions is not appropriate for real time algebras. Recently, MOLLER & TOFTS have proposed timed CCS [24] which differs mainly from our work because their choice constructs (there are two different ones) are both time deterministic. We find these deterministic choices also in [18, 25]. But these articles allow time to continue when no actions can happen, e.g. $\partial_{\{a\}}(a)$ can perform time steps. In [18] the process $t \cdot a$ can perform an a action after one or more time steps. Such an unrestrained semantics may lead to difficulties when specifying systems where actions must be performed within strict time intervals.

There are many open questions in the area of real time process algebras. One of the most often raised questions is how dense time can be incorporated in process algebra. Work of BAETEN & BERGSTRA [1] shows that this is far from trivial. Especially, - as was also noted in [24] - it is hard to find a complete axiomatization. Other questions concern notions such as fairness and liveness in a setting where time is explicit. Also performance issues can be studied, but this depends very much on the possibility to add probabilities to the formalism [14].

Acknowledgements. I thank Alban Ponse, Frits Vaandrager and Fer-Jan de Vries for their helpful comments.

References

[1] J.C.M. BAETEN & J.A. BERGSTRA (1989): *Real time process algebra.* Report P8916, University of Amsterdam, Programming Research Group.

[2] J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1986): *Syntax and defining equations for an interrupt mechanism in process algebra.* Fund. Inf. IX(2), pp. 127-168.

[3] J.C.M. BAETEN & R.J. VAN GLABBEEK (1989): *Abstraction and empty process in process algebra.* Fundamenta Informaticae XII, pp. 221-242.

[4] J.A. BERGSTRA & J.W. KLOP (1985): *Algebra of communicating processes with abstraction.* Theoretical Computer Science 37(1), pp. 77-121.

[5] J.A. BERGSTRA & J.W. KLOP (1986): *A complete inference system for regular processes with silent moves.* In: Proceedings Logic Colloquium 1986 (F.R. Drake & J.K. Truss, eds.), North Holland, Hull, pp. 21-81, also appeared as Report CS-R8420, Centrum voor Wiskunde en Informatica, Amsterdam, 1984.

[6] J.A. BERGSTRA & J.W. KLOP (1986): *Process Algebra: specification and verification in bisimulation semantics.* In: Mathematics and Computer Science II, CWI Monograph 4 (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), North-Holland, Amsterdam, pp. 61-94.

[7] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG (1986): *Failure semantics with fair abstraction.* Report CS-R8609, Centrum voor Wiskunde en Informatica, Amsterdam.

[8] J.A. BERGSTRA, J.W. KLOP & E.-R. OLDEROG (1987): *Failures without chaos: a new process semantics for fair abstraction.* In: Formal Description of Programming Concepts - III, Proceedings of the third IFIP WG 2.2 working conference, Ebberup 1986 (M. Wirsing, ed.), North-Holland, Amsterdam, pp. 77-103.

[9] F.P.M. BIEMANS & P. BLONK (1986): *On the formal specification and verification of CIM architectures using LOTOS.* Computers in Industry 7, pp. 491-504.

[10] E. BRINKSMA (1987): *A theory for the derivation of tests.* In: Proceedings of the Eighth International Conference on Protocol Specification, Testing and Verification (S. Aggarwal ed.), North-Holland, Amsterdam.

[11] R. GERTH & A. BOUCHER (1987): *A timed failures model for extended communicating processes.* In: Proceedings 14^{th} ICALP, Karlsruhe (T. Ottmann ed.), LNCS 267, Springer Verlag, pp. 95-114.

[12] R.J. VAN GLABBEEK (1987): *Bounded nondeterminism and the approximation induction principle in process algebra.* In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidaal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347.

[13] R.J. VAN GLABBEEK (1990): *The linear time - branching time spectrum.* In: *Comparative Concurrency Semantics and Refinement of Actions*, pp. 13-48, Ph.D. Thesis, Free University, Amsterdam.

[14] R.J. VAN GLABBEEK, S.A. SMOLKA, B. STEFFEN & C.M.N. TOFTS (1990): *Reactive, generative, and stratified models of probabilistic processes.* To appear in: Proceedings 5^{th} Annual Symposium on Logic in Computer Science (LICS 90), Philadelphia, USA, IEEE Computer Society Press, Washington.

[15] R.J. VAN GLABBEEK & F.W. VAANDRAGER (1988): *Modular specifications in process algebra - with curious queues.* Report CS-R8821, Centrum voor Wiskunde en Informatica, Amsterdam. An extended abstract appeared in: Algebraic Methods: Theory, Tools and Applications (M. Wirsing & J.A. Bergstra, eds.), LNCS 394, Springer-Verlag, pp. 465-506

[16] R.J. VAN GLABBEEK & W.P. WEIJLAND (1989): *Branching time and abstraction in bisimulation semantics (extended abstract).* In: Information Processing 89, G.X. Ritter, ed. Elsevier Science Publishers B.V. (North Holland) pp. 613-618.

[17] J.F. GROOTE & F.W. VAANDRAGER (1988): *Structured operational semantics and bisimulation as a congruence.* Report CS-R8845, Centrum voor Wiskunde en Informatica, Amsterdam. An extended abstract appeared in: Proceedings ICALP 89, Stresa (G. Ausiello, M. Dezani-Ciancaglini & S. Ronchi Della Rocca, eds.), LNCS 372, Springer-Verlag, pp. 423-438.

[18] M. HENNESSY & T. REGAN (1990): *A Temporal process algebra.* Report 2/90, University of Sussex, Brighton.

[19] C.A.R. HOARE (1985): *Communicating sequential processes.* Prentice-Hall International.

[20] R. KOYMANS, R.K. SHYAMASUNDAR, W.P. DE ROEVER, R. GERTH & S. ARUN-KUMAR (1988): *Compositional semantics for real-time distributed computing.* I&C 79, pp. 210-256.

[21] R. MILNER (1983): *Calculi for synchrony and asynchrony.* Theoretical Computer Science 25., pp. 267-310.

[22] R. MILNER (1989): *Communication and concurrency.* Prentice-Hall International.

[23] F. MOLLER (1988): *Axioms for concurrency.* Ph.D. thesis, Report CST-59-89. Department of Computer Science, University of Edinburgh.

[24] F. MOLLER & C. TOFTS (1990): *A temporal calculus of communicating systems.* Technical Report, Department of Computer Science, University of Edinburgh.

[25] X. NICOLLIN, J.-L. RICHIER, J. SIFAKIS & J. VOIRON (1990): *ATP: An algebra for timed processes.* Report RT-C16. IMAG, Laboratoire de Génie informatique, Grenoble. to appear in: Proceedings IFIP Working Conference on Programming Concepts and Methods, Sea of Gallilee, Israel (M. Broy & C.B. Jones, eds.), North Holland, 1990.

[26] M. PHALIPPOU (1988): *A proposal for modelling real time in the MR.* SPECS internal deliverable P.CNET.WP5.5.

[27] G.M. REED & A.W. ROSCOE (1986): *A timed model for communicating sequential processes.* In: Proceedings 13^{th} ICALP, Rennes (L. Kott ed.), LNCS 226, Springer Verlag, pp. 314-323.

[28] J.L. RICHIER, J. SIFAKIS & J. VOIRON (1987): *Une algèbre des processus temporisés.* Actes du deuxième colloque C^3, Angouléme, 20-22 mai 1987, A. Arnold ed.

[29] W. P. WEIJLAND (1989): *Synchrony and asynchrony in process algebra.* Ph.D. thesis, University of Amsterdam, Amsterdam.