An Asynchronous Process Algebra with Multiple Clocks

Henrik Reif Andersen and Michael Mendler

Technical University of Denmark, Department of Computer Science,
Building 344, DK-2800 Lyngby, Denmark.
E-mail: {hra,mvm}@id.dth.dk.

Abstract. In this paper we introduce a novel approach to the specification of real-time behaviour with process algebras. In contrast to the usual pattern, involving a fixed, measurable, and global notion of time, we suggest to represent real-time constraints indirectly through uninterpreted *clocks* enforcing broadcast synchronization between processes. Our approach advocates the use of *asynchronous* process algebras, which admit the faithful representation of nondeterministic and distributed computations.

Technically, we present a non-trivial extension of the *Calculus of Communicating Systems* CCS [Mil89a] by *multiple clocks* with associated *timeout* and *clock ignore* operators. We illustrate the flexibility of the proposed process algebra, called *PMC*, by presenting examples of rather different nature. The timeout operators generalize the timeout of ATP [NS90] to multiple clocks. The main technical contribution is a complete axiomatization of strong bisimulation equivalence for a class of finite-state processes and a complete axiomatization of observation congruence for finite processes.

1 Introduction

According to consolidating tradition in timed process algebras a real-time system is perceived to operate under the regime of a global time parameter constraining the occurrence of actions [NS91b]. Time has algebraic structure, typically a totally ordered commutative monoid, to express quantitative timing constraints. The semantics of a timed process then is given as a transition system enriched by quantitative timing information such as the absolute duration of actions or their time of occurrence.

This paper puts forward yet another process algebra; why bother? Most of the salient approaches, such as [MT90, Wan90, Lia91, NS91a, Klu91, SDJ+91], primarily aim at describing completely the global real-time behaviour of timed systems in a fairly realistic fashion. The means for abstracting from real time is restricted to the choice of the time domain; for instance, instead of working with real numbers one may decide to go for rational or discrete time. We believe that these approaches are often overly realistic with disadvantages for both the specification and the modelling of real-time systems. Firstly, for specifying a timed process, complete quantitative information about the intended timing behaviour of the implementation is required. This includes not only the specification-relevant, i.e. safety-critical timing, but also specification-irrelevant timing parameters which, so we believe, constitute the majority in practice. Being forced to include a lot of irrelevant timing information, which

can be left well up to the implementation, is unfortunate as this is unnecessarily cutting down the design space, perhaps even preventing the designer from finding a reasonable implementation at all. Secondly, many real-time process algebras require to give exact numbers for the duration of actions, such as "3.141 time units to enter a valid login response". Examples are Timed-ACP as described in [Klu91], Timed-CSP [SDJ+91], ATPD [NS91a], or [Wan90]. But exact delays are in general very difficult to implement due to uncontrollable fabrication parameters, operating conditions such as circuit temperature or external events. At best we can hope to implement delay intervals. A process algebra using delay intervals rather than exact time was proposed by Liang [Lia91]. Such an algebra, however, suffers even more from being cluttered up with irrelevant timing information. Another process algebra with interval durations is CIPA [AM93a]. A disadvantage of time intervals are the severe problems they cause for simulation, in particular where time is dense: It is not feasible faithfully to simulate time intervals for the purpose of timing validation.

In this paper we propose a rather abstract approach to the specification and modelling of real-time systems that captures the nature of timing constraints through the use of *multiple clocks*. Clocks enforce global synchronization of actions without compromising the abstractness of time by referring to a concrete time domain which is globally fixed once and for all.

The concept of time underlying the use of clocks is abstract, qualitative, and local. Firstly, it is *abstract* since it does not prejudice any particular way of realizing a clock. We are free to interpret a clock as the ticking of a global real-time watch measuring absolute process time, as the system clock of a synchronous processor, or as the completion signal of a distributed termination protocol. Clocks are a general and flexible means for bundling asynchronous behaviour into intervals. Secondly, the concept of time underlying the use of clocks is *qualitative* since it is not the absolute occurrence time or duration of actions that is constrained but their relative ordering and sequencing wrt. clocks. Our approach postpones the analysis of timing inequations until a concrete realization is fixed assigning specific delays to actions and clock distances. For timing-critical aspects, of course, a particular delay constraint would be imposed on a particular clock already at specification time. Finally, clocks admit a *local* notion of time since in different subprocesses independent clocks can be used, which may or may not be realized referring to the same time base.

The contribution of this paper is to introduce the syntax and semantics of the process algebra *PMC*, which is a non-trivial extension of CCS [Mil89a] by multiple clocks. The semantics of *PMC* is based on transition systems with separate action and clock transitions. Actions are *insistent*, so that local constraints on the progress of clocks can be expressed. The important features of PMC demonstrated in this paper are its flexibility in expressing timing constraints, and the fact that it admits a complete axiomatization of strong bisimulation equivalence for a class of finite-state processes and of observation congruence for finite processes.

2 Syntax and Semantics of PMC

In PMC concurrent systems are described by their ability to perform *actions* and synchronize with *clocks*. As in CCS we assume a set of *action names* $\mathcal{A}$ and their complemented versions $\bar{\mathcal{A}}$. Let ¯ be a bijection between $\mathcal{A}$ and $\bar{\mathcal{A}}$ whose inverse is also denoted by ¯. We assume an additional *silent* action τ, which is not in $\mathcal{A} \cup \bar{\mathcal{A}}$ and take the set of *actions* to be $Act =_{\text{def}} \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$. Communication takes place as a synchronization between complementary actions a and $\bar{a}$; the actions $a \in \mathcal{A}$ are considered to be *input actions* and the actions $\bar{a}$ to be *output actions*. In the sequel we let $\alpha, \beta, \ldots$ range over Act.

In addition to the ordinary actions of CCS, PMC assumes a finite set of clocks Σ the elements of which are ranged over by $\rho, \sigma, \sigma', \sigma_1$ etc. Whereas the actions are used for two-by-two synchronization between parallel processes, the clocks enforce broadcast synchronization in which all processes of a parallel composition must take part. In fact, clocks mimic the properties of time in that the effect of a clock tick reaches through almost all syntactic operators.

Let x range over a set of *process variables*. Process terms t are generated from the following grammar:

$$t ::= \mathbf{0} \mid \alpha.t \mid t_0 + t_1 \mid t \setminus a \mid \lfloor t_0 \rfloor \sigma(t_1) \mid t \uparrow \sigma \mid x \mid rec\ x.t$$

Roughly, the meaning of the process operators, in terms of their ability to perform actions or to take part in clock ticks, is as follows. *Nil*: $\mathbf{0}$ is the process which can do nothing, neither an action nor does it admit a clock tick. *Insistent prefix*: $\alpha.t$ is the process which performs α and then behaves as t; it prevents all clocks from ticking, which motivates calling it 'insistent' prefix. The term 'insistent' is taken from Hennessy [Hen93]. Prefixes that stop time from progressing also have been called *urgent* [BL91] or *immediate* [NS91a]. *Sum*: $t_0 + t_1$ is the process which must behave as any of t_0 or t_1, the choice being made with the first action. *Composition*: $t_0 \mid t_1$ represents t_0 and t_1 performing concurrently with possible communication. *Restriction*: $t \setminus a$ behaves like t but with actions $a, \bar{a}$ not allowed to occur. Each one of the processes $t_0 + t_1$, $t_0 \mid t_1$, and $t \setminus a$ takes part in a clock tick σ by having all of its components t_0, t_1, t take part in it. *Timeout*: $\lfloor t_0 \rfloor \sigma(t_1)$ is the process which behaves like t_0 if an initial action of t_0 is performed or a clock tick different from σ occurs; if however σ occurs first it is transformed into t_1, whence the name 'timeout'. *Ignore*: The process $t \uparrow \sigma$ behaves just like t but it will always take part in a σ clock tick without changing its state. *Recursion*: $rec\ x.t$ is a distinguished solution of the process equation $x = t$.

The notion of a *closed* term and the set of *free* variables of a term are defined as usual. A variable x is *weakly guarded* in a term t if each occurrence of x is within a subexpression t' of t which is in the scope of a $\alpha.t'$ or of a $\lfloor \cdot \rfloor \sigma(t')$. If we require $\alpha \neq \tau$ in this definition, then x is *strongly guarded* in t. A term t is weakly/strongly guarded if every variable occurring in t is weakly/strongly guarded. We will use the symbol $\equiv$ to denote syntactic equality.

The semantics of PMC is given by a *labelled transition system* $\mathcal{T} = (\mathcal{P}, \mathcal{L}, \rightarrow)$, where $\mathcal{P}$ is the set of closed process terms, $\mathcal{L} =_{\text{def}} Act \cup \Sigma$ is the set of labels, and $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is the transition relation. In distinguishing between pure action

$$\alpha.p \xrightarrow{\alpha} p \qquad \frac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'} \qquad \frac{q \xrightarrow{\alpha} q'}{p+q \xrightarrow{\alpha} q'}$$

$$\frac{p \xrightarrow{a} p' \quad q \xrightarrow{\bar{a}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'} \qquad \frac{p \xrightarrow{\alpha} p'}{p \mid q \xrightarrow{\alpha} p' \mid q} \qquad \frac{q \xrightarrow{\alpha} q'}{p \mid q \xrightarrow{\alpha} p \mid q'}$$

$$\frac{p \xrightarrow{\alpha} p'}{p \setminus a \xrightarrow{\alpha} p' \setminus a} \; (\alpha \neq a, \alpha \neq \bar{a}) \qquad \frac{t[rec \; x.t/x] \xrightarrow{\alpha} p}{rec \; x.t \xrightarrow{\alpha} p}$$

$$\frac{p \xrightarrow{\alpha} p'}{\lfloor p \rfloor \sigma(q) \xrightarrow{\alpha} p'} \qquad \frac{p \xrightarrow{\alpha} p'}{p \uparrow \sigma \xrightarrow{\alpha} p' \uparrow \sigma}$$

Fig. 1. Action rules.

and pure clock/time transitions the semantics follows the popular pattern for timed process algebras [NS91b]. Of course, there are other ways of incorporating time. We mention ICPA [AM93a] where the transitions carry both action and time information. The transition relation → of PMC is defined in Plotkin style as the least relation closed under the set of *action rules* and *clock progress rules* given in Figs. 1 and 2.

$$\frac{p \xrightarrow{\sigma} p' \quad q \xrightarrow{\sigma} q'}{p+q \xrightarrow{\sigma} p'+q'} \quad \frac{p \xrightarrow{\sigma} p' \quad q \xrightarrow{\sigma} q'}{p \mid q \xrightarrow{\sigma} p' \mid q'} \quad \frac{p \xrightarrow{\sigma} p'}{p \setminus a \xrightarrow{\sigma} p' \setminus a} \quad \frac{t[rec \; x.t/x] \xrightarrow{\sigma} p}{rec \; x.t \xrightarrow{\sigma} p}$$

$$\lfloor p \rfloor \sigma(q) \xrightarrow{\sigma} q \qquad \frac{p \xrightarrow{\sigma'} p'}{\lfloor p \rfloor \sigma(q) \xrightarrow{\sigma'} p'} \; (\sigma \neq \sigma')$$

$$p \uparrow \sigma \xrightarrow{\sigma} p \uparrow \sigma \qquad \frac{p \xrightarrow{\sigma'} p'}{p \uparrow \sigma \xrightarrow{\sigma'} p' \uparrow \sigma} \; (\sigma \neq \sigma')$$

Fig. 2. Clock progress rules.

The action rules (Fig. 1) for the processes $\alpha.p, p+q, p \mid q, p \setminus a$, and *rec x.t* follow the usual rules for CCS. Also, the clock progress rules (Fig. 2) for these processes require little comment. The idea is that a clock tick is a global time synchronization which synchronously affects all subterms of a process. The action and clock progress rules for $p \uparrow \sigma$ and $\lfloor p \rfloor \sigma(q)$ reflect the informal explanation given above.

It will be convenient to extend the timeout operator to (possibly empty) sequences $\underline{\sigma} = \sigma_1 \cdots \sigma_n$ of clocks by the following inductive definition:

$$\lfloor t \rfloor = t$$
$$\lfloor t \rfloor \sigma_1(u_1) \cdots \sigma_n(u_n) = \lfloor \lfloor t \rfloor \sigma_1(u_1) \cdots \sigma_{n-1}(u_{n-1}) \rfloor \sigma_n(u_n).$$

We shall sometimes use the vector notation $\lfloor t \rfloor \underline{\sigma}(\underline{u})$ to abbreviate $\lfloor t \rfloor \sigma_1(u_1) \cdots \sigma_n(u_n)$.

The following is a list of some important derived constructions which are all special cases of timeout:

$$
\begin{array}{lll}
0_{\underline{\sigma}} =_{\text{def}} rec\ x. \lfloor 0 \rfloor \sigma_1(x) \cdots \sigma_n(x), & \text{(relaxed nil)} \\
\alpha :_{\underline{\sigma}} t =_{\text{def}} rec\ x. \lfloor \alpha.t \rfloor \sigma_1(x) \cdots \sigma_n(x) & \text{(relaxed prefix)} \\
\sigma.t =_{\text{def}} \lfloor 0 \rfloor \sigma(t) & \text{(wait)} \\
\sigma :_{\underline{\sigma}} t =_{\text{def}} rec\ x. \lfloor 0 \rfloor \sigma_1(x) \cdots \sigma_n(x)\sigma(t) & \text{(relaxed wait)}
\end{array}
$$

For the clock progress rules of Fig. 2 we notice the absence of a rule for nil and prefix. They both stop *all* clocks; 0 has the rather dramatic effect of stopping all clocks indefinitely, it is a time-lock. In contrast, the process $0_{\underline{\sigma}}$ in the above table is a relaxed version of nil which does not perform any action but allows the clocks in $\underline{\sigma}$ to proceed. The maximally relaxed nil process, which enables all clocks, is abbreviated by 1, i.e. $1 =_{\text{def}} 0_{\sigma_1 \cdots \sigma_N}$, where $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$. If Σ is empty, then 1 and 0 coincide. The process $\alpha :_{\underline{\sigma}} t$ above is a *relaxed* version of a prefix: it lets clocks $\underline{\sigma}$ tick away freely without changing its state, until an α action occurs which transforms it into t. The derived process $\sigma.t$ may be the most common construct in applications: it waits for the clock σ before proceeding with t, and in doing so it is stopping all other clocks. Finally, $\sigma :_{\underline{\sigma}} t$ is a *relaxed* wait: it waits for clock σ but allows all clocks in $\underline{\sigma}$ to tick.

It should be mentioned that by a generalization of the above constructions for 're-laxing' processes the ignore $p \uparrow \sigma$ actually can be eliminated from *closed* processes by induction on the structure of p. This does not mean that $\uparrow$ is redundant syntactically, since it is not a derived operator and the elimination does not work on open terms.

3   Example: Signal Analyzer

The initial ideas and motivation leading to PMC developed from an attempt formally to specify the Brüel & Kjær 2145 Vehicle Signal Analyzer.[1]

The Brüel & Kjær 2145 Vehicle Signal Analyzer is an instrument for measuring noise from machines with rotating objects such as cars and turbines. The instrument receives input from microphones and tachometers, computes running spectra of ampli-tudes of frequencies, and presents the results on a screen. Various measuring scenarios and presentation modes are available. The analyzer is operated in real-time and the results are visualized on the screen as they are computed.

In this example, to illustrate an application of PMC, we will simplify the real design and describe only that part of the instrument that collects sound samples from one microphone, speed pulses from one tachometer, and computes a speed-related spectrum. Figure 3 gives an overview of the simplified system. It features seven communication channels represented by the solid lines, and three clocks represented by dashed lines. The signal analyzer reads in sound samples along channel s, tacho pulses along p, and outputs on channel $srsp$ the speed-related spectrum computed

---

[1] This case study is done in co-operation with the manufacturing company and is part of the Co-design Project at the Department of Computer Science, DTH.
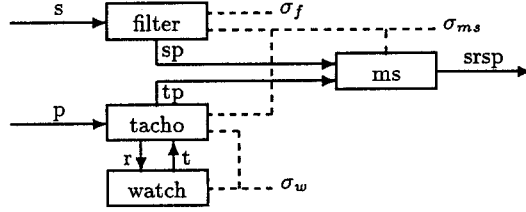
Fig. 3. The Signal Analyzer.

from the samples and the tacho information. The clocks reflect central timing aspects of the system behaviour. The sampling rate is modelled by the clock σ_f which ticks with a fixed frequency determined by the input bandwidth. Another clock σ_w is used for measuring the time between tacho pulses. It also ticks with a fixed frequency determining the precision by which time is measured in the instrument. A third clock σ_{ms} is used for synchronizing the exchange of information between the three processes 'filter', 'tacho' and 'ms'.

Abstracting away from the actual values sent along the channels and concentrating on the communication patterns and the real-time aspects, the system consists of the following four processes:

• The process 'filter' reads after each tick of σ_f a sample on s, computes a spectrum (modelled by a τ) and then either delivers the spectrum on sp in response to a tick of σ_{ms} or waits for the next sample to arrive.

• The process 'watch' keeps track of the time based on ticks of the clock σ_w. It can be reset using channel r and the current time can be read on channel t.

• The process 'tacho' records the number of tacho pulses arriving on p and, whenever the clock σ_{ms} ticks, delivers the pulse count on tp together with the time distance since the last delivery (as measured by 'watch'). For safe real-time operation 'tacho' must occasionally prevent σ_w from ticking.

• The measuring process 'ms' collects, with every tick of the clock σ_{ms}, a spectrum on sp, a tacho count and time distance on tp. From these it computes a speed-related spectrum finally delivered on $srsp$. Depending on the spectrum received and the value of the tacho count, this can be more or less involved. We model this by the 'ms' process making an internal choice between delivering the result immediately or performing some lengthy internal computation (modelled by a sequence of τ's below) before doing so.

The following is a PMC description of the system:

$$\text{filter} = \lfloor \sigma_f \cdot s \cdot \tau \cdot \text{filter} \rfloor \sigma_{ms}(\overline{sp} \cdot \text{filter})$$

$$\text{watch} = \lfloor r \cdot \text{watch} + \bar{t} \cdot \text{watch} \rfloor \sigma_w(\text{watch})$$

$$\text{tacho} = \lfloor p :_{\sigma_w} \text{tacho} \rfloor \sigma_{ms}(t :_{\sigma_w} \bar{r} \cdot \overline{tp} :_{\sigma_w} \text{tacho})$$

$$\text{ms} = \sigma_{ms} \cdot sp \cdot tp \cdot (\tau \cdot \overline{srsp} \cdot \text{ms} + \tau \cdot \cdots \tau \cdot \overline{srsp} \cdot \text{ms})$$

The filter, tacho, and watch processes constitute the input system

$$\text{INP} = (\text{filter} \uparrow \sigma_w) \mid (\text{tacho} \uparrow \sigma_f \mid \text{watch} \uparrow \sigma_f \uparrow \sigma_{ms}) \setminus t \setminus r$$

and the complete system is

$$\text{SYS} \; = \; (\text{INP} \mid \text{ms} \uparrow \sigma_w \uparrow \sigma_f) \setminus sp \setminus tp.$$

With the formal description of the system at hand we can make precise the rôle of the three clocks $\sigma_w, \sigma_{ms}, \sigma_f$.

For correct real-time operation it is important to make sure that with every input action t 'tacho' obtains the exact number of σ_w ticks arrived since the last time it read the watch. This implies that *no* tick of σ_w must fall between reading the current time of the watch with action t and resetting it with action $\bar{r}$. This real-time requirement is conveniently dealt with in PMC by using an insistent prefix after $\bar{r}$ in the tacho process, which prevents σ_w from ticking between reading and resetting of the watch. Using ordinary actions instead of a clock to distribute time signals, this mutual exclusion between the tacho and the watch process would have to be encoded using a protocol.

The clock σ_{ms} ensures that whenever 'filter', 'tacho' and 'ms' are ready, consistent pairs of spectra and tacho values are sent. This relies on the broadcast feature of clocks forcing all parties to synchronize. Using normal actions an effect like this requires a rather complex protocol, and it is quite hard to ensure that the values from 'filter' and 'tacho' never arrive out of synchrony.

Finally, the clock σ_f also plays some rôle. It might be argued that as long as the samples from the environment arrive at the right speed on the channel s, the clock σ_f is unnecessary. However, if the samples are available from the environment as the values of a state variable that can always be read, it is important that this happens only at certain well-defined points. This is enforced by the clock. Moreover, one could imagine adding another filter process sampling a parallel input channel in synchrony with the first one. The synchronization of both filters, which comes for free with the clock σ_f, otherwise would have to be encoded via a protocol.

4 Example: Synchronous Hardware

In hardware design one is dealing frequently with architectures consisting of a number of interconnected synchronous systems that are all driven by independent, i.e. local clocks. Such systems are called *multi-clock synchronous systems* or *globally-asynchronous, locally-synchronous machines* [Cha87]. The synchronous subsystems exchange data via communication buffers which decouple the computations and compensate for different relative clock speeds. The simplest case of a communication buffer is the *input synchronizer* as shown in Fig. 4.

The input synchronizer S is prepared to accept input data on x from the environment at any time, and offers it to the synchronous system at its output y only on the next tick of the local clock σ_2. Of any sequence of input data arriving before the clock tick only the most recent input is transmitted. Since the output value the synchronizer offers on y may change with every clock tick the output behaviour will not be preserved by clock transitions. It can be shown [AM93b] that this cannot be expressed merely with wait, nil, or prefixes, be they relaxed or not. However, with the
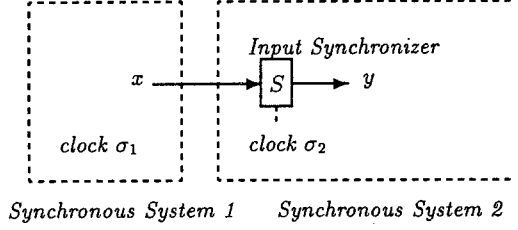
Fig. 4. An input synchronizer.

general timeout operator the synchronizer can be defined:

$$S(v,w) =_{\mathrm{def}} \lfloor x?d \,.\, S(d,w) \;+\; y!w \,.\, S(v,w) \rfloor \sigma_2(S(v,v)),$$

where we encode value-passing as in CCS by the abbreviations

$$x?d \,.\, p \equiv_{\mathrm{def}} \Sigma_{d\in\mathbf{D}} x_d \,.\, p \qquad y!w \,.\, p \equiv_{\mathrm{def}} \bar{y}_w \,.\, p,$$

assuming that $\mathbf{D}$ is a finite domain of values and x,y families of distinct actions indexed by the elements $d, w \in \mathbf{D}$. The synchronizer process $S(v,w)$ has two parameters: the first one, v, represents the state of the input line which can be changed to a new value d by an $x?d$ action at any time. The second parameter w is the state of the output line. It is passed on to the synchronous system with $y!w$ at any time, and is updated with the value of the first parameter whenever clock σ_2 ticks.

The input synchronizer can be used to connect up two single-clock synchronous systems, say SC_1 and SC_2, running with independent clocks σ_1, σ_2:

$$MC(v) =_{\mathrm{def}} SC_1 \uparrow \sigma_2 \;\mid\; (S(v,v) \mid SC_2) \uparrow \sigma_1.$$

How single-clock synchronous systems can be modelled in PMC is explained in [AM93b]. The idea is to use the insistent prefixes to synchronize the function blocks globally and force the timing constraint upon the clock signal. This leads to a compositional calculus of synchronous systems, in which all the subcomponents of SC_i themselves are special cases of synchronous systems.

5 Axiomatization

Having set up the syntax and semantics of PMC, we are now going to present a formal calculus for reasoning about equivalence of PMC processes. We wish to axiomatize two notions of equivalence, viz. strong bisimilarity and observation congruence. These notions carry over naturally from CCS [Mil89a] by treating clock and ordinary action transitions in the same way:

Definition 1. A relation $S \subseteq \mathcal{P} \times \mathcal{P}$ is a *strong bisimulation* if it is symmetric, and for all $(p,q) \in S$ and $l \in \mathcal{L} = Act \cup \Sigma$, whenever $p \xrightarrow{l} p'$ then for some q', $q \xrightarrow{l} q'$ and $(p',q') \in S$. Two processes p and q are *strongly bisimilar* if $p \sim q$, where $\sim$ is the union of all strong bisimulations.

Example 1. The following processes are equivalent formulations of filter and watch from Sec. 3:

$$\text{filter}' \equiv rec \; x. \; \sigma_f :_{\sigma_{ms}} s \; . \; \tau \; . \; x \; + \; \sigma_{ms} :_{\sigma_f} \overline{sp} \; . \; x$$
$$\text{watch}' \equiv (rec \; x. \; r \; . \; x + \overline{t} \; . \; x) \uparrow \sigma_w,$$

i.e. we have filter $\sim$ filter$'$ and watch $\sim$ watch$'$. Notice that the latter equivalence holds only because we are abstracting away from values. If values were considered our watch surely would not ignore time.

To define observation congruence we need a few auxiliary concepts: If $s \in \mathcal{L}^* = (Act \cup \Sigma)^*$ then $\hat{s} \in (\mathcal{L} \backslash \tau)^*$ is the sequence obtained from s by deleting all occurrences of τ. For $s = s_1 \cdots s_n \in \mathcal{L}^*$ $(n \geq 0)$, we write $p \xrightarrow{s} q$ if $p \xrightarrow{s_1} \cdots \xrightarrow{s_n} q$, and $p \xRightarrow{s} q$ if $p \xrightarrow{\tau}* \xrightarrow{s_1} \xrightarrow{\tau}* \cdots \xrightarrow{s_n} \xrightarrow{\tau}*q$. Note in particular, $p \Rightarrow p$.

Definition 2. A relation $S \subseteq \mathcal{P} \times \mathcal{P}$ is a *weak bisimulation* if it is symmetric, and for all $(p,q) \in S$ and $l \in \mathcal{L}$, whenever $p \xrightarrow{l} p'$ then for some q', $q \xRightarrow{\hat{l}} q'$ and $(p',q') \in S$. Two processes p and q are *weakly bisimilar* if $p \approx q$, where $\approx$ is the union of all weak bisimulations. *Observation congruence*, written $\approx^c$, is the largest congruence contained within $\approx$.

We extend both equivalences to open terms in the usual way by stipulating tEu if for every substitution θ of closed terms for the free variables, $t[\theta]Eu[\theta]$, where E is one of $\sim$, $\approx^c$. For processes p, q without clock transitions the definitions coincide with the corresponding notions for CCS, whence no extra equivalences are introduced for the CCS sublanguage of PMC. In other words, PMC is a conservative extension of CCS.

One can show that the offset between $\approx$ and $\approx^c$ is almost of the same nature as for CCS.

Lemma 3. $p \approx^c q$ *iff one of the following three equivalent properties holds:*

- *For all r, $p + r \approx q + r$.*
- *If $p \xRightarrow{\sigma} p'$ then for some q', $q \xRightarrow{\sigma} q'$ and whenever $p' \xrightarrow{\alpha} p''$ then $q' \xRightarrow{\alpha} q''$ for some q'' such that $p'' \approx q''$; the same holds symmetrically with p and q interchanged.*
- *$(p,q) \in S$ where S is any symmetric binary relation on $\mathcal{P}$ with the property that for all $(r,s) \in S$, $\alpha \in Act$, $\sigma \in \Sigma$,*
  (i) if $r \xrightarrow{\alpha} r'$ then for some s', $s \xRightarrow{\alpha} s'$ and $r' \approx s'$,
  (ii) if $r \xrightarrow{\sigma} r'$ then for some s', $s \xrightarrow{\sigma} s'$ and $(r',s') \in S$.

According to the first characterization the reason why $\approx$ fails to be a congruence can be localized in the sum operator. In this respect the situation is precisely as in CCS. The second characterization brings up the difference: while in CCS congruent processes need a strong match for the first τ actions in PMC they need to match any initial clock sequence followed by a τ. The third characterization coincides with one given by Moller and Tofts for observation congruence in TCCS [MT92]. The equivalence between the last two characterizations is due to the following property:

$$
\begin{array}{ll}
\textbf{S1} & t + u = u + t \\
\textbf{S2} & t + (u + v) = (t + u) + v \\
\textbf{S3} & t + t = t \\
\textbf{S4} & \alpha.t + 0 = \alpha.t \\
\textbf{S5} & t + 1 = t
\end{array}
$$

$$
\begin{array}{lll}
\textbf{B1} & \lfloor \lfloor t \rfloor \sigma(u) \rfloor \sigma(v) = \lfloor t \rfloor \sigma(v) & \\
\textbf{B2} & \lfloor \lfloor t \rfloor \sigma(u) \rfloor \sigma'(v) = \lfloor \lfloor t \rfloor \sigma'(v) \rfloor \sigma(u) & \sigma \neq \sigma' \\
\textbf{B3} & \lfloor t \rfloor \sigma(u) + \lfloor v \rfloor \sigma(w) = \lfloor t + v \rfloor \sigma(u + w) & \\
\textbf{B4} & \lfloor t \rfloor \sigma(u) + \lfloor 0 \rfloor \underline{\sigma}(\underline{v}) = t + \lfloor 0 \rfloor \underline{\sigma}(\underline{v}) & \sigma \notin \underline{\sigma}
\end{array}
$$

$$
\textbf{CR} \quad \text{if } \alpha.u = \alpha.v \text{ then } \alpha.\lfloor t \rfloor \sigma(u) = \alpha.\lfloor t \rfloor \sigma(v)
$$

$$
\begin{array}{lll}
\textbf{C1} & 0 \setminus a = 0 & \\
\textbf{C2} & (\alpha.t) \setminus a = \begin{cases} 0 & \alpha = a \text{ or } \alpha = \bar{a} \\ \alpha.(t \setminus a) & \text{otherwise} \end{cases} & \\
\textbf{C3} & (t + u) \setminus a = t \setminus a + u \setminus a & \\
\textbf{C4} & \lfloor t \rfloor \sigma(u) \setminus a = \lfloor t \setminus a \rfloor \sigma(u \setminus a) &
\end{array}
$$

$$
\begin{array}{ll}
\textbf{I1} & 0 \uparrow \sigma = \lfloor 0 \rfloor \sigma(0 \uparrow \sigma) \\
\textbf{I2} & (\alpha.t) \uparrow \sigma = \lfloor \alpha.(t \uparrow \sigma) \rfloor \sigma((\alpha.t) \uparrow \sigma) \\
\textbf{I3} & (t + u) \uparrow \sigma = t \uparrow \sigma + u \uparrow \sigma \\
\textbf{I4} & \lfloor t \rfloor \sigma'(u) \uparrow \sigma = \lfloor t \uparrow \sigma \rfloor \sigma'(u \uparrow \sigma) \sigma(\lfloor t \rfloor \sigma'(u) \uparrow \sigma)
\end{array}
$$

Fig. 5. Equational laws for sum, timeout, restriction, and ignore.

$$
\textbf{E} \qquad t \mid u = \lfloor r \rfloor \sigma_1(t'_1 \mid u'_1) \cdots \sigma_k(t'_k \mid u'_k),
$$

where t, u are terms

$$
t \equiv \lfloor \sum_I \alpha_i.t_i \rfloor \rho_1(t'_1) \cdots \rho_l(t'_l) \qquad u \equiv \lfloor \sum_J \beta_j.u_j \rfloor \sigma_1(u'_1) \cdots \sigma_m(u'_m),
$$

such that $\rho_i = \sigma_i$ for $1 \leq i \leq k$, and $\rho_i \neq \sigma_j$ for $k < i$ or $k < j$, and r is the term

$$
r \equiv \sum_I \alpha_i.(t_i \mid u) + \sum_J \beta_j.(t \mid u_j) + \sum_{\alpha_i = \bar{\beta}_j} \tau.(t_i \mid u_j).
$$

Fig. 6. The expansion law.

$$
\begin{array}{llll}
\textbf{R0} & rec\ x.t & = & rec\ y.t[y/x] \qquad y \text{ not free in } rec\ x.t \\
\textbf{R1} & rec\ x.t & = & t[rec\ x.t/x] \\
\textbf{R2} & \text{if } u = t[u/x] \text{ then } u = rec\ x.t \qquad x \text{ guarded* in } t \\
\textbf{R3} & rec\ x.(\lfloor x \rfloor \underline{\sigma}(\underline{u}) + t) & = & rec\ x.(\lfloor 0 \rfloor \underline{\sigma}(\underline{u}) + t)
\end{array}
$$

* Weakly guarded if = is interpreted as $\sim$, strongly guarded if it stands for $\approx^c$. In the latter case t must be regular too.

Fig. 7. Laws for recursion.

T1	$\alpha.\tau.t = \alpha.t$
T2	$\tau.t + t = \tau.t$
T3	$\alpha.(t + \lfloor \tau.u \rfloor \underline{\sigma}(\underline{v})) + \alpha.u = \alpha.(t + \lfloor \tau.u \rfloor \underline{\sigma}(\underline{v}))$
T4	$\alpha.\lfloor \tau.\lfloor r \rfloor \sigma(\lfloor \tau.s + t \rfloor \underline{\rho}(\underline{u})) + v \rfloor \underline{\kappa}(\underline{w}) = \alpha.\lfloor \tau.\lfloor r \rfloor \sigma(\lfloor \tau.s + t \rfloor \underline{\rho}(\underline{u})) + v \rfloor \sigma(s)\underline{\kappa}(\underline{w})$

Fig. 8. Tau laws.

Proposition 4 Clock Determinism. *If $p \overset{\sigma}{\rightarrow} q$ and $p \overset{\sigma}{\rightarrow} r$, then $q \equiv r$.*

By definition, $\approx^c$ is a congruence wrt. all operators, including the recursion operator *rec x*. It is not difficult to verify that $\sim$ also is a congruence wrt. all operators. Congruicity is important, since it shows that in an axiomatization of both equivalences, $\sim$, $\approx^c$, Leibniz' rule of 'substituting equals for equals' is sound, that is, if we have proven $t = u$ then t and u can be interchanged in any context. We shall use the symbol $\vdash$ to denote that an equality $\vdash t = u$ is derivable using equational reasoning and the special laws that we shall consider in the sequel.

Theorem 5 Soundness. *The laws of Fig. 5–7 are sound for $\sim$ and $\approx^c$. The τ laws of Fig. 8 are sound for $\approx^c$.*

Example 2. Let us prove, by equational reasoning from our axioms, the equivalence in example 1 for the watch process:

$$
\begin{aligned}
\text{watch}' &\equiv A \uparrow \sigma_w \\
&= (r \,.\, A + \bar{t} \,.\, A) \uparrow \sigma_w && \textbf{R1} \\
&= (r \,.\, A) \uparrow \sigma_w + (\bar{t} \,.\, A) \uparrow \sigma_w && \textbf{I3} \\
&= \lfloor r \,.\, \text{watch}' \rfloor \sigma_w((r \,.\, A) \uparrow \sigma_w) + \lfloor \bar{t} \,.\, \text{watch}' \rfloor \sigma_w((\bar{t} \,.\, A) \uparrow \sigma_w) && \textbf{I2} \\
&= \lfloor r \,.\, \text{watch}' + \bar{t} \,.\, \text{watch}' \rfloor \sigma((r \,.\, A) \uparrow \sigma_w + (\bar{t} \,.\, A) \uparrow \sigma_w) && \textbf{B3} \\
&= \lfloor r \,.\, \text{watch}' + \bar{t} \,.\, \text{watch}' \rfloor \sigma(\text{watch}') && \textbf{I3, R1}.
\end{aligned}
$$

By rule **R2** we conclude $\vdash \text{watch}' = \text{watch}$.

Definition 6. A term t is said to be *regular* if it is built from nil, prefix, sum, timeout, variables and the recursion operator. A process term t is *rs-free* (rs abbreviates 'recursion through static operators') if every subterm *rec x.u* of t is regular. A process p is *finite* if it does not contain any recursion or ignore operators.

Theorem 7 Completeness. *If p and q are rs-free processes with $p \sim q$ then $\vdash p = q$ using equational reasoning from the laws of Figs. 5–7 (without **CR**). If p and q are finite processes and $p \approx^c q$, then $\vdash p = q$ using equational reasoning from the laws of Figs. 5–8.*

The proof of completeness for $\sim$ can be found in [AM93b] and for $\approx^c$ it will appear elsewhere. The proofs are basically an adaptation of Milner's technique [Mil84, Mil89a] but rather more involved due to a more complicated normal form representation and the larger gap between $\approx$ and $\approx^c$.

In view of the completeness theorem one might hope that a lot of the mathematical theory of CCS can be carried over easily to PMC. But the situation is not quite so simple. There are some subtle technical complications making PMC a non-trivial extension of CCS.

Firstly, the standard approach extending completeness from finite to finite-state processes [Mil89b] builds on the fact that in CCS unguarded processes can always be transformed into guarded ones. Unfortunately, this property fails to hold for PMC, with the consequence if $\approx^c$ can be completely axiomatized for finite-state processes then a new proof strategy must be found. For instance, take the unguarded process $p \equiv rec\ x.\ \lfloor \tau\ .\ \lfloor \tau\ .\ x \rfloor \sigma(\mathbf{1}) \rfloor \sigma(\mathbf{0})$. In every state of p reachable through, possibly zero, τ actions there is a weak σ transition both to $\mathbf{0}$ and to $\mathbf{1}$. This property must be enjoyed by any process q weakly bisimilar to p. However, to fulfill this property q must either be infinite state or have a τ loop. To see why, consider a state q_0 such that $q \Rightarrow q_0$ and $q_0 \overset{\sigma}{\rightarrow} \Rightarrow \mathbf{0}$ which must exist by assumption. But there must also be a state q_1 such that $q_1 \overset{\sigma}{\rightarrow} \Rightarrow \mathbf{1}$ and q_1 is reachable from q_0 through a sequence of τ's. This sequence cannot be empty for otherwise $q_1 \equiv q_0$ and we would have $q_0 \overset{\sigma}{\rightarrow} \Rightarrow \mathbf{1}$ contradicting clock-determinism (Prop. 4). Hence we must have $q_0 \overset{\tau}{\Rightarrow} q_1$. Now the same argument applies to q_1, so we could go on constructing a sequence $q_0 \overset{\tau}{\Rightarrow} q_1 \overset{\tau}{\Rightarrow} q_2 \overset{\tau}{\Rightarrow} \cdots$. But this means that q either has a τ loop or is infinite state.

Secondly, the sum $+$ which is a dynamic operator in CCS, has *static* behaviour wrt. time steps, i.e. it does not disappear after any number of clock transitions. So, to obtain the transition system of $p + q$ from the transition systems of p and q we take the *disjoint union* with respect to ordinary action transitions, but the *product* with respect to clock transitions. This means that in the equational characterization of sums $p+q$ more equations must be added than are needed for the pure CCS fragment. In connection with recursion the situation becomes even more involved: Because of the static nature of $+$ wrt. to clock transitions there are regular processes with infinite syntactic unfolding. For instance the process $P \equiv rec\ x.\ \lfloor \mathbf{0} \rfloor \sigma(x + (p \uparrow \sigma))$ admits the infinite transition sequence

$$P \overset{\sigma}{\rightarrow} P + p \uparrow \sigma \overset{\sigma}{\rightarrow} \cdots \overset{\sigma}{\rightarrow} P + p \uparrow \sigma + \cdots + p \uparrow \sigma \overset{\sigma}{\rightarrow} \cdots$$

producing bigger and bigger terms. In the pure CCS fragment, on the other hand, a regular process always has a finite syntactic unfolding. Nevertheless regular PMC processes have a finite number of states modulo $\sim$ (cf. [AM93b]).

6   Related Work

We begin with a remark on terminology. The most distinguished feature of PMC is the notion of 'clock'. For most purposes this term would denote a means for measuring time in order to time-stamp observations. In the process language CIPA [AM93a] or in the timed automata of Alur and Dill [AD91] clocks are used in this sense. In PMC, however, the intended interpretation of 'clock' is more like that of a hardware clock, viz. a global signal used to synchronize asynchronous computations in a lock-step fashion.

We believe that in the context of asynchronous process calculi the concept of multiple synchronization clocks — in our sense — is novel. Yet, it is not entirely new as it has been used already in synchronous real-time description languages. LUSTRE [HPOG89] is a language for synchronous data-flow with multiple clocks, where all clocks are derived from a master clock through boolean expressions. LUSTRE was developed originally for real-time programming but is used also for describing digital circuits. Another quite successful real-time language with a multi-form notion of time is ESTEREL [BC84]. It must be noted however, that in both these languages clocks are not built-in; they are ordinary signals or variables, not an independent semantical concept as in PMC. A synchronous language where clocks do possess genuine semantical meaning with an associated 'clock calculus' is SIGNAL [BBG93].

The obvious — albeit not stringent — path towards a technical comparison with other published work is to view PMC with a single clock as a discrete time process algebra, where a time delay of size n corresponds to n successive clock ticks. For instance, if we fix a particular clock σ we can define timing operators $(n).t$ and $\delta.t$ as follows

$$
\begin{aligned}
(0).t &\equiv t \\
(n+1).t &\equiv \sigma \cdot (n).t \\
\delta.t &\equiv rec\ x.\ \lfloor t \rfloor \sigma(x)
\end{aligned}
$$

with n ranging over natural numbers. For every n, the process $(n).t$ waits n clock transitions of σ before it evolves into t, and until then it remains quiet. The process $\delta.t$ is a delayed version of t which allows σ to proceed until such time as the environment is ready to communicate with it. These constructs are taken as primitives in the timed process calculus TCCS of Moller and Tofts [MT90]. In [MT92] a complete axiomatization for TCCS of observation congruence on finite, sequential processes is presented. Both PMC and TCCS use insistent action prefixes, but where PMC has a timeout operator to produce relaxed actions, relaxed behaviour is introduced in TCCS by a different nonstandard primitive, the *weak* sum $p \oplus q$. It behaves exactly as $p + q$ for ordinary actions, but in contrast to + forces both components to take part in a time transition only if both can do a time transition together. If one of p and q does not admit a time transition it is considered *stopped*, in which case the other process can engage in a time step all by itself while the stopped process is simply dropped from the computation. As hinted at in [MT90] the $\oplus$ plays an important role in obtaining an expansion law for TCCS, which in turn is crucial for proving completeness. It is interesting to note that the equational axiomatization of PMC seems to be considerably simpler than that of TCCS where the expansion law for parallel composition has to consider various special cases (to do with $\oplus$) while in PMC one single equation scheme suffices.

It is possible to view discrete time TCCS as a subcalculus of PMC modulo the following syntactic encoding of $\oplus$:

$$
(s \oplus t)^* \equiv_{\text{def}} \begin{cases} s^* + \lfloor t^* \rfloor \sigma(1) & \text{if } s^* \xrightarrow{\sigma} \text{ and } t^* \overset{\sigma}{\not\rightarrow} \\ \lfloor s^* \rfloor \sigma(1) + t^* & \text{if } s^* \overset{\sigma}{\not\rightarrow} \text{ and } t^* \xrightarrow{\sigma} \\ s^* + t^* & \text{otherwise.} \end{cases}
$$

The TCCS constructs $(n).t$ and $\delta.t$ are replaced by the definitions given above. All other constructs are represented in PMC by their respective equivalents. It can be shown that the condition $\overset{\sigma}{\not\rightarrow}$ in the encoding of $\oplus$ can be decided on the syntactic structure, so that $(\cdot)^*$ is in fact a well-defined syntactic operation [AM93b] on closed terms. We conjecture that for closed TCCS processes p with all variables guarded the operational behaviour of the encoding p^* in PMC is precisely the one obtained for p in TCCS.

A rather different class of timed process algebras is that with *relaxed* actions and *maximal progress* [Wan90, HR91]. These principles reflect a rigid two-phase view of real-time execution: In the first phase a component is allowed to perform an arbitrary but finite number of internal communications at zero time cost. When all internal chatter has ceased, i.e. the component has stabilized, time is allowed to proceed in the second phase. The duration of the second phase is the amount of time elapsing until the component again becomes internally unstable. In [NS91b] this two-phase model is generalized to an arbitrary set of *urgent* actions for which maximal progress is enforced. These urgent actions play the role of internal communication in that these actions must be performed before time is allowed to proceed.

This two-phase model cements a *globally-synchronous, locally-asynchronous* type of behaviour. The major mode is synchronous operation since the phases of asynchronous cooperation are bundled together when all subcomponents of a process synchronize to let time advance. In PMC we adopt a more flexible scheme which allows us to localize the notion of time and progress. We can thus obtain simpler and more abstract specifications for distributed systems covering not only globally synchronous, locally asynchronous systems but also the class of *globally-asynchronous, locally-synchronous* behaviour (cf. Sec. 4). It might be possible to extend the maximal progress approach in this direction, an idea suggested in [Hen93], but not without major modification such as 'localizing' maximal progress in some appropriate way.

Some remarks on the timeout operators are in order. Our timeouts are an extension to multiple clocks of Nicollin and Sifakis' timeout introduced originally with the process algebra ATP. In [NS90] they present a complete axiomatization for ATP of strong bisimulation equivalence. ATP is rather like a single clock version of PMC but there are some notable differences. ATP is restricted to rs-free guarded processes without time-locks but has a generalized restriction and parallel composition. We mention that although the syntax of PMC is more involved due to multiple clock constructs the normal form representation seems to be more uniform than in ATP. In PMC only one normal form scheme needs to be handled while in ATP three different cases are treated separately.

There are other "time-insistent" variants of timeouts used in the literature which differ from our's basically in that for $\lfloor p \rfloor \sigma(q)$ to perform a σ time step the process p must not prevent time from progressing, which is not the case here. Examples are the constructs $p \overset{d}{\triangleright} q$ of Nicollin and Sifakis [NS91a] and $\lfloor p \rfloor (q)$ of Hennessy and Regan [HR91]. The relaxed time behaviour of $\lfloor p \rfloor \sigma(q)$ wrt. p is important for PMC as it allows us to derive from it a number of useful "time-relaxed" constructs such as relaxed prefixes. With a time-insistent timeout these relaxed prefixes would have to be added

to the language as primitives, resulting in additional axioms and more complicated normal forms. For instance, in [NS91a] the normal form has to treat separately three different cases. Another central design decision simplifying the normal form of PMC processes is that the clock transitions of $\lfloor p \rfloor \sigma(q)$ for clocks $\sigma' \neq \sigma$ are treated in the same way as ordinary actions of p, i.e. they remove the timeout.

7 Conclusion

We have presented an extension of CCS by multiple clocks, timeout, and ignore operators, and we have given a complete axiomatization of strong bisimulation equivalence for a class of finite-state processes and of observation congruence for finite processes. The process algebra PMC was developed to capture the quantitative nature of real-time constraints and it is aimed at applications in which real-time requirements are few but essential. We believe that PMC offers a promising compromise between expressiveness and realizability or executability. PMC is currently being used as a specification language in an industrial case study at the Department of Computer Science, DTH. A prototype implementation of a value-passing version of PMC is under development, using the ML-Kit [BRTT93].

Acknowledgement

References

[AD91] R. Alur and D. Dill. The theory of timed automata. In de Bakker et al. [dBHdRR91], pages 45–73.

[AM93a] L. Aceto and D. Murphy. On the ill-timed but well-caused. In E. Best, editor, *Proc. Concur'93*, pages 97–111. Springer LNCS 715, 1993.

[AM93b] H. R. Andersen and M. Mendler. A process algebra with multiple clocks. Technical Report ID-TR:1993-122, Department of Computer Science, Technical University of Denmark, August 1993.

[BBG93] A. Benveniste, M. Le Borgne, and P. Le Guernic. Hybrid systems: The SIGNAL approach. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, pages 230–254. Springer LNCS 736, 1993.

[BC84] G. Berry and L. Cosserat. The ESTEREL synchronous programming language and its mathematical semantics. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, pages 389–448. Springer LNCS 197, 1984.

[BK90]   J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR '90*, volume
            458 of *LNCS*. Springer-Verlag, 1990.

[BL91]   T. Bolognesi and F. Lucidi. Timed process algebras with urgent interactions
            and a unique powerful binary operator. In de Bakker et al. [dBHdRR91], pages
            124–148.

[BRTT93]   L. Birkedal, N. Rothwell, M. Tofte, and D. N. Turner. The ML Kit, Version 1.
            Technical Report, DIKU, March 1993.

[Cha87]   Daniel M. Chapiro. Reliable high-speed arbitration and synchronization. *IEEE
            Transaction on Computers*, C-36(10):1251–1255, October 1987.

[dBHdRR91] J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors.
            Real-Time: Theory in Practice, volume 600 of *LNCS*. Springer-Verlag, 1991.

[Hen93]   M. Hennessy. Timed process algebras: A tutorial. Technical Report 93:02,
            Department of Computer Science, University of Sussex, January 1993.

[HPOG89]   N. Halbwachs, D. Pilaud, F. Ouabdesselam, and A.-C. Glory. Specifying, pro-
            gramming and verifying real-time systems using a synchronous declarative lan-
            guage. In *Workshop on automatic verification methods for finite state systems*,
            Grenoble, France, June 12–14 1989. Springer LNCS 407.

[HR91]   M. Hennessy and T. Regan. A process algebra for timed systems. Computer
            Science Technical Report 91:05, Department of Computer Science, University
            of Sussex, April 1991. To appear in Information and Computation.

[Klu91]   A. S. Klusener. Abstraction in real time process algebra. In de Bakker et al.
            [dBHdRR91], pages 325–352.

[Lia91]   Chen Liang. An interleaving model for real-time systems. Technical Report
            ECS-LFCS-91-184, Laboratory for Foundations of Computer Science, Univer-
            sity of Edinburgh, November 1991.

[Mil84]   Robin Milner. A complete inference system for a class of regular behaviours.
            J. of Computer and System Sciences, 28(3):439–466, June 1984.

[Mil89a]   Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil89b]   Robin Milner. A complete axiomatisation for observational congruence of
            finite-state behaviours. *Information and Computation*, 81:227–247, 1989.

[MT90]   Faron Moller and Chris Tofts. A temporal calculus of communicating systems.
            In Baeten and Klop [BK90], pages 401–415.

[MT92]   F. Moller and Ch. Tofts. Behavioural abstraction in TCCS. In W. Kuich,
            editor, *Proc. ICALP'92*, pages 559–570. Springer LNCS 623, 1992.

[NS90]   X. Nicollin and J. Sifakis. The algebra of timed processes ATP: theory and ap-
            plication. Technical Report RT-C26, LGI-IMAG, Grenoble, France, December
            1990.

[NS91a]   X. Nicollin and J. Sifakis. From ATP to timed graphs and hybrid systems. In
            de Bakker et al. [dBHdRR91], pages 549–572.

[NS91b]   X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras.
            In de Bakker et al. [dBHdRR91], pages 526–548.

[SDJ+91]   S. Schneider, J. Davies, D.M. Jackson, G.M. Reed, J.N. Reed, and A.W.
            Roscoe. Timed CSP: Theory and practice. In de Bakker et al. [dBHdRR91],
            pages 526–548.

[Wan90]   Yi Wang. Real-time behaviour of asynchronous agents. In Baeten and Klop
            [BK90].