# A Process Algebra for Timed Systems*

Matthew Hennessy, Tim Regan
University of Sussex

**Abstract**: A standard process algebra is extended by a new action $\sigma$ which is meant to denote *idling*

very descriptive languages with which one may describe the minutiae of detailed timing considerations in complex systems. Such languages are certainly required but there are certain applications, those in which time plays a restricted role, for which these languages may be inappropriate because the descriptions may be unnecessarily complex. Our proposal is quite modest: we wish to make a relatively minor extension to a standard process algebra with a mathematically simple notion of time which, although not universally applicable, will be sufficiently useful in particular application areas such as protocol verification. Protocols are a typical example of systems where timing considerations affect the behaviour of only a small part of the overall system. Our language is designed so that specification of the time-independent part of the system may be carried out as usual while the time-dependent part may be treated with our time-based extension. We hope that by introducing a simple notion of time many of the characteristics of standard process algebras which have made them so successful will still be retained in the enlarged setting. In particular we wish to extend the semantic theory of processes based on testing, [He88], to a setting where time plays a significant role. From a methodological point of view it seems appropriate to start with a language in which the concept of time is rather simple.

The idea is to introduce into a standard process algebra, *CCS*, a $\sigma$ action. The execution of this $\sigma$ action by a process indicates that it is idling or doing nothing until the next clock cycle. This action will share many of the properties of the standard actions of *CCS* but because it represents the passage of time it will be distinguished from the standard actions by certain of its properties. For example in our process algebra this action will be *deterministic* in the sense that a process can only reach at most one new state by performing $\sigma$ . This is a reflection of the assumption that the passage of time is deterministic. There is also an intuitive assumption underlying the usual (asynchronous) theories of process algebras, such as *CCS* as expounded in [Mil89] and *CSP* as expounded in [Hoa85], that all processes may idle indefinitely and the semantic theory is formulated in terms of the actions which a process may perform, if it so wishes. Indeed, this view of processes is investigated in detail in [Mil8 ]. We continue to use this assumption; using the syntax of *CCS,* the process $a.p$ can idle, i.e. it can perform a $\sigma$ action. This means that we assume all processes are *patient* in that they will wait indefinitely until communications in which they can participate become possible. Moreover this means that the implicit assumption underlying *CCS* that all communication actions are instantaneous is retained in our language since we have a distinguished action $\sigma$ denoting the passage of time and, as we will see, all other actions are performed in between occurrences of this time action. However, we add one further assumption, namely that communications must occur if they are possible: a process cannot delay if it can perform a communication. This we call the *maximal progress* assumption, [HdR89], which is a common feature of many proposed timed process algebras. So, again using the syntax of *CCS*, although $a.p + b.q$ can idle, $(a.p + b.q)|\overline{a}.q$ cannot idle; the communication via the $a$ iddhjΩ1 .68010T.4TdΩ[ajΩ1 .68010T0TdΩ[(wil6jΩ-4 2.481[8d]TJΩ75.847(TdΩ(arejΩ10.08010

1. *discrete time:* in our language time proceeds in discrete steps represented by occurrences of the action $\sigma$,

2. *time determinism:* we assume that the passage of time is deterministic,

. *actions are instantaneous:* time is not associated directly with

passage of time. An intuition for this is that if a process is offering an action $a$ which is requested by another process by the offer of an action $\overline{a}$, we do not want unspecified delay to occur; the communication, the $\tau$ move, must fire immediately.

- $\sigma$.. The passage of time is modelled in our system by an occurance of a $\sigma$ action. As discussed in the introduction this represents a relatively abstract notion of time but it can be intuitively thought of as the click of a clock which measures the passage of time for the system. We chose $\sigma$ as the symbol to represent the passage of time because of its similarity to Phillips 'broadcast stability operator' of [Ph87].

- $+$. Deterministic and nondeterministic choice between two processes is modelled in *CCS* by the operator $+$. For actions $a$ in *Act* and the action $\tau$ the operator $+$ behaves in the same way as it does in the *CCS* setting. The difference comes with the action $\sigma$ . If two processes are just idling before the environment requests one of them the choice between them will not be made by the passage of time alone. That is to say that $+$ is not decided by the action $\sigma$ . This is necessary to ensure that the passage of time is deterministic.

- $\lfloor\ \rfloor(\ )$. This operator comes from the process algebra *ATP* put forward in [NRSV92]. It is similar to the context $\_ + \sigma.\tau.\_$

$$ACT_1 : \frac{}{\alpha.p \xrightarrow{\alpha} p}$$

$$SUM_1 : \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad SUM_2 : \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'}$$

$$THEN_1 : \frac{p \xrightarrow{\alpha} p'}{\lfloor p \rfloor(q) \xrightarrow{\alpha} p'}$$

$$COM_1 : \frac{p \xrightarrow{\alpha} p'}{p|q \xrightarrow{\alpha} p'|q} \qquad COM_2 : \frac{q \xrightarrow{\alpha} q'}{p|q \xrightarrow{\alpha} p|q'}$$

$$COM_3 : \frac{p \xrightarrow{a} p',\ q \xrightarrow{\overline{a}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

$$REL_1 : \frac{p \xrightarrow{\alpha} p'}{p[S] \xrightarrow{S(\alpha)} p'[S]} \qquad RES_1 : \frac{p \xrightarrow{\alpha} p',\ \alpha \notin \{b, \overline{b}\}}{p \setminus b \xrightarrow{\alpha} p' \setminus b}$$

$$REC_1 : \frac{t[recx.t/x] \xrightarrow{\alpha} p'}{recx.t \xrightarrow{\alpha} p'}$$

Figure 1: Standard Operational Semantics

to delay until the next time cycle. Similarly, $nil$ may delay indefinitely as it can never perform a communication. Note, however, that $\tau.p$ cannot delay; it

$$ACT_2 : \frac{}{a.p \xrightarrow{\sigma} a.p} \qquad NIL : \frac{}{nil \xrightarrow{\sigma} nil}$$

$$WAIT : \frac{}{\sigma.p \xrightarrow{\sigma} p}$$

$$SUM_3 : \frac{p \xrightarrow{\sigma} p',\ q \xrightarrow{\sigma} q'}{p+q \xrightarrow{\sigma} p'+q'}$$

$$THEN_2 : \frac{p \not\xrightarrow{\tau}}{\lfloor p \rfloor(q) \xrightarrow{\sigma} q}$$

$$COM_4 : \frac{p \xrightarrow{\sigma} p',\ q \xrightarrow{\sigma} q',\ p|q \not\xrightarrow{\tau}}{p|q \xrightarrow{\sigma} p'|q'}$$

$$REL_2 : \frac{p \xrightarrow{\sigma} p'}{p[S] \xrightarrow{\sigma} p'[S]} \quad RES_2 : \frac{p \xrightarrow{\sigma} p'}{p \setminus a \xrightarrow{\sigma} p' \setminus a}$$

$$REC_2 : \frac{t[recx.t/x] \xrightarrow{\sigma} p'}{recx.t \xrightarrow{\sigma} p'}$$

Figure 2: Operational Semantics for $\sigma$

□

**Example 2.2** The process $Egg_1$ is defined so that if left too long before eating the egg MAY be unhealthy (and may not).

$$Egg_1 \Leftarrow \overline{eat}.healthy.nil + \sigma.\sigma.\overline{eat}.unhealthy.nil.$$

The process $Egg_2$ is defined so that if left too long before eating the egg WILL be unhealthy.

$$Egg_2 \Leftarrow \lfloor \overline{eat}.healthy.nil \rfloor (\lfloor \overline{eat}.healthy.nil \rfloor (\overline{eat}.unhealthy.nil))$$

□

**Example 2.3** A "leaking counter" defined informally as

$$\begin{aligned} C_0 &\Longleftarrow press.up.C_1 \\ C_{n+1} &\Longleftarrow press.up.C_{n+2} + \sigma.down.C_n. \end{aligned}$$

It can perform on *up* action each time it is *pressed* but if no *press* is forthcoming before the next clock cycle it can perform a *down* action. So, for example, $(C_0|\overline{press}^k) \setminus press$ acts like the process $up^k.(\sigma.down)^k$. □

Some of the informal assumptions underlying the design of the language which we discussed in the introduction can now be seen to be reflected in the operational semantics. This is the import of the following proposition:

**Proposition 2.4**

1. *(Time-determinism) if* $p \xrightarrow{\sigma} q$ *and* $p \xrightarrow{\sigma} q'$ *then* $q$ *and* $q'$ *are syntactically the same*
2. *(Maximal progress) if* $p \xrightarrow{\tau} q$ *then* $p \xrightarrow{\sigma} r$ *for no process* $r$

**Proof:** By induction on the length of the proof of $p \xrightarrow{\sigma} q$, $p \xrightarrow{\tau} q$ respectively. □

The informal assumption of *patience* is not as straightforward to capture. Intuitively this should state that if a process $p$ can not perform a $\tau$ action then it must be able to delay, i.e. perform a $\sigma$ action. But because of the presence of recursive definitions the situation is more complicated. For example the processes $recx.x$ and $\Omega$ can perform no action whatsoever. Intuitively these terms represent under-defined or "badly defined processes" and therefore they require special attention. Terms which intuitively represent well-defined processes are captured in the following definition:

**Definition 2.5** (Strong Convergence)
Let $\downarrow$ be the least (postfix) predicate over *TPL* which satisfies

$$\begin{array}{rl} i) & nil\downarrow,\ \alpha.p\downarrow,\ \sigma.p\downarrow \\ ii) & p\downarrow \text{ implies } (\lfloor p\rfloor(q))\downarrow, (p|q)\downarrow,\ p\setminus a\downarrow,\ p[S]\downarrow, \\ iii) & p\downarrow,\ q\downarrow, \text{ implies } (p+q)\downarrow, \\ iv) & t[recx.t/x]\downarrow \text{ implies } recx.t\downarrow . \end{array}$$

□

We write $p\uparrow$ to denote the negation of $\downarrow$ and one can check that, for example, $\Omega\uparrow$ and $recx.x\uparrow$. With this new notation we can now see how *patience* is reflected in our operational semantics.

**Proposition 2.6** *(Patience) If* $p\downarrow$ *and* $p \xrightarrow{\tau} q$ *for no process* $q$ *then there exists a process* $r$ *such that* $p \xrightarrow{\sigma} r$

**Proof:** By induction on the proof that $p\downarrow$. □

We now turn our attention to the definition of a behavioural preorder between processes. We follow the approach of [He88] which is based on testing and for convenience we only consider the "must" case. However, because *TPL* is an extension of *CCS*, the definitions we employ will be based on those from [dNH84] where the predicate $\downarrow$ plays a necessary role. A test $e$ is a process from *TPL* which may additionally use the special action $\omega$ for reporting success. A test $e$ is applied to a process $p$ by "running" the process

$e|p$, i.e. allowing it to evolve via $\tau$ actions or $\sigma$ actions. Specifically a *computation* from $e|p$ is a maximal sequence (which may be finite or infinite) of the form

$$e|p = e_0|p_0 \mapsto e_1|p_1 \mapsto \ldots \mapsto e_i|p_i \mapsto \ldots \quad (\text{where } \mapsto = \xrightarrow{\tau} \cup \xrightarrow{\sigma})$$

To say when such an application is a success we need the notion of strong convergence defined above.

We say $p\ must\ e$ if in every computation from $e|p$,

$$e|p = e_0|p_0 \mapsto \ldots \mapsto e_k|p_k \mapsto \ldots$$

there exists some $n \geq 0$ that $e_n \xrightarrow{\omega}$, i.e. $e_n$ can report success, and for every $k, 0 \leq k < n$ $e_k|p_k \downarrow$. Finally, we say that

$$p \sqsubseteq q$$

if for every test $e$, $p\ must\ e$ implies $q\ must\ e$. We use $\simeq$ to denote the kernel of this preorder.

The definition of $\sqsubseteq$ is close to that employed in [dNH84] and, therefore, if we restrict both the processes and the tests to *CCS* the resulting theory is exactly that developed in [dNH84]. However here we also allow occurrences of $\sigma$ in the tests and these new tests, even when applied to *CCS* terms, i.e. terms not involving the timing constructs $\sigma$ and $\lfloor\ \rfloor(\ )$, have more distinguishing power than standard *CCS* tests. An interesting difference in the power $\sigma$ vests in testing languages can be found in [La89]:

**Example 2.7** This example concerns two vending machines (shown diagrammatically in Figure  where $\sigma$ actions are ignored) with slightly different internal behaviour

$$coin.(tea + hit.tea) + coin.(coffee + hit.coffee)$$

and

$$coin.(tea + hit.coffee) + coin.($$

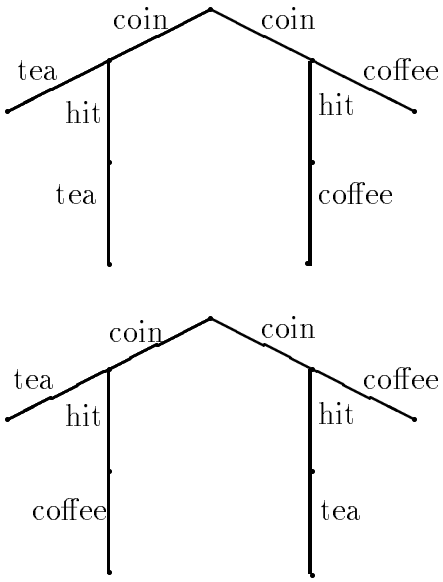



Figure : Langerak's Vending Machines

# 3 Alternative Characterisation

In this section we look at an alternative characterisation of

with $1 \leq k$ where $s_i \in (Act \cup \sigma)^*$ and each $A_i$ is a finite subset of $Act$. These barbs may be compared using the following order:

**Definition 3.2** $\ll$ is the least preorder over barbs which satisfies

1. $\Omega \ll \underline{b}$

The $r$ superscript in this definition stands for *regular* since, although this ordering serves as an alternative characterisation for *CCS* (as the next theorem states), it is inadequate for *TPL*. We will need to restrict our attention to a specialisation of barbs to obtain an alternative characterisation of *TPL*. The reason for this is that any stable *CCS* process may perform a $\sigma$ action to itself whereas this is not true in *TPL* (e.g. $\lfloor a.nil \rfloor (b.nil)$). It is also worth pointing out that this definition differs from that in [HR90]. There the definition of $\ll^r$ is defined in terms of the preorder $<$ and convergence of processes over barbs, a concept we have not defined. We prefer here to define $\ll^r$ in terms of the sets of barbs alone. This substantially reduces the amount of work involved in checking the equivalence of processes in next section's soundness proof.

**Theorem 3.4** *(Alternative Characterisation for CCS) For p,q in CCS, $p \sqsubseteq q$ if and only if $p \ll^r q$.* □

It is worth pointing out that this theorem is not true if we restrict *Barb(p)* to simple barbs, i.e. those where each sequence $s_i$ is of length at most one. For example, let $p$ denote the process $d.(a.nil + c.nil | \overline{c}.nil) \setminus c$. Then both $p$ and $d.nil$ have exactly the same simple barbs, namely prefixes of $\varepsilon$

**Proof:** Let us assume $p \ll q$ and $p$ *must* $e$. We prove $q$ *must* $e$ by examining an arbitrary computation from $e|q$:

$$C \equiv e|q = e_0|q_0 \mapsto e_1|q_1 \mapsto \ldots e_i|q_i \mapsto \ldots \tag{1}$$

Each move $\mapsto$ may either be $\stackrel{\sigma}{\longrightarrow}$ or $\stackrel{\tau}{\longrightarrow}$ and let us concentrate on the former. Then (1) may be rewritten in the form

$$C \equiv e|q = e_0|q_0 \stackrel{\tau}{\longrightarrow}^* f_1|r_1 \stackrel{\sigma}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* f_2|r_2 \ldots e_i|q_i \stackrel{\sigma}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* \ldots \tag{2}$$

where each $f_i|r_i$ corresponds to some $e_j|q_j$, $j \geq 1$. Note that this sequence of $f_i|r_i$'s may be finite even if the origional sequence is infinite. Now this computation may be "unzipped" to reveal the contributions from the test and process:

$$q_0 \stackrel{s_1}{\Longrightarrow} r_1 \stackrel{\sigma}{\longrightarrow} \stackrel{s_2}{\Longrightarrow} r_2 \stackrel{\sigma}{\longrightarrow} \ldots$$

$$e_0 \stackrel{\overline{s_1}}{\Longrightarrow} f_1 \stackrel{\sigma}{\longrightarrow} \stackrel{\overline{s_2}}{\Longrightarrow} f_2 \stackrel{\sigma}{\longrightarrow} \ldots \tag{ }$$

For the moment let us assume that each $e_i$ in the computation is strongly convergent. This allows us to concentrate on the derivation from $q_0$

Since $p$

**Lemma 3.7** *For every standard barb $\underline{b}$ and for every finite $L \subseteq Act$ such that $Sort(\underline{b}) \subseteq L$ if $Sort(p) \subseteq L$ then*

$$p \not\mathit{must}\ e(\underline{b}, L) \Leftrightarrow \exists \underline{a} \in SBarb(p)\ \ \underline{a} \ll \underline{b}.$$

**Proposition 3.8** *For p,q in TPL $p \sqsubseteq q$ implies $p \ll q$.*

**Proof:** We prove the contrapositive, namely $\neg(p \ll q)$ implies $\neg(p \sqsubseteq q)$. If $p \ll q$ is not true then for some standard barb $\underline{b} \in SBarb(q)$ $\underline{b}' \ll \underline{b}$ for no $\underline{b}'$ in $SBarb(p)$. In this case we employ the Lemma .7 where L is chosen so as to contain both of the finite sets $Sort(p)$ and $Sort(q)$. □

Combining these two propositions we immediately have the Alternative Characterisation Theorem for *TPL*. As a direct corollary to this we can restrict the experimenters considered with no change to the discriminatory power.

**Definition 3.9** An *F-test* $f$ is a *TPL* experiment of the following recursively defined form.

- $f = \tau.\omega$.
- $f = \lfloor \sum_A a.\omega \rfloor (nil)$.
- $f = \tau.\omega + a.f'$, where $f'$ is an F-test.
- $f = \lfloor \sum_A a.\omega \rfloor (f')$, where $f'$ is an F-test.

□

**Lemma 3.10** *For any TPL processes p,q $p \sqsubseteq q$ if, and only if, for all F-tests f p must f implies q must f*

**Proof:** We have proved above that $p \sqsubseteq q \Leftrightarrow p \ll q$. The proof of Lemma .7 then gives that we need only consider F-tests. □

With this alternative characterisation it is now relatively straightforward to compare processes with respect to $\sqsubseteq$. As an example we return to Example 2.7. We can distinguish the two vending machines with the barb $coin\{tea, hit\}\sigma hit\{tea\}$ which is a standard barb of the second process unmatched by one from the first. The alternative characterisation also enables us to prove the characterisation of $\sqsubseteq^c$ promised in the previous section.

**Theorem 3.11**

$$p \sqsubseteq^c q \Leftrightarrow p \sqsubseteq^+ q.$$

**Proof:** It is sufficient to show that $\sqsubseteq^+$ is respected by all the operators. We leave the individual proofs, which are quite tedious to the reader but we should point out the proof for the restriction operator requires some care. □

# 4 Proof Systems

In this section we develop a sound and complete proof system for the language *TPL*. The importance of a proof system for a language is great. Some process algebras are defined equationally since their designers feel this to be the most intuitive starting point. Indeed Schmidt says in [Sch86]:

> "The [axiomatic] format is best used to provide preliminary specifications for a language or to give documentation about properties that are of interest to the users of the language."

Hence it is often to the equations that one turns to reveal the differences between languages and the equivalences defined upon them. The importance of the soundness of a proof system is obvious, it merely requires that the equations and proof rules are true of the language under examination. Completeness is often harder to prove since it requires that all truths in the language should be provable in the proof system.

The proof system we consider is based on the inequations given in Figures 4 and 5. Many of these are standard equations for *CCS* but the operators $\lfloor\ \rfloor(\ )$ and $\sigma$ introduce new and sometimes complex axioms, particularly in relation to the internal operator $\tau$. From the equation $\sigma 1$ it is apparent that $\sigma$ is expressible in terms of $\lfloor\ \rfloor(\ )$ but we have deliberately used $\sigma$ in the presentation because it is easily understood intuitively. Also note that $e1$ is not derivable from $e2$.

The proof system is defined in Figure 6. It is essentially inequational reasoning with extra rules for recursive terms, $REC$ and $\omega - Induction$. In the latter $App(t)$ denotes the set of finite approximations to $t$, $\{t^n : n \geq 0\}$, defined by:

1. $t^0 = \Omega$
2. (a) $x^{n+1} = x$
   (b) $f(\underline{t})^{n+1} = f(\underline{t}^{n+1})$
   (c) $(recx.t)^{n+1} = t^{n+1}[(recx.t)^n/x]$.

These have been discussed at length in [He88]. The only extra rule is the $Stability-Rule$, a simple form of which equates the process $a.nil$ with $recx.(a.nil + \sigma.x)$ or even $nil$ with $recx.\sigma.x$.

Let $\vdash_E t \leq u$ denote that $t \leq u$ is derivable in this proof system, $t \leq_E u$ that $t \leq u$ is derivable with purely inequational reasoning and finally $t \leq_{Er} u$ that $t \leq u$ is derivable using in addition the unfolding rule $REC$.

We first discuss the soundness of the axioms. To characterise their importance we

$$
\begin{array}{rcll}
x + x & = & x & +1 \\
x + y & = & y + x & +2 \\
x + (y + z) & = & (x + y) + z & + \\
x + nil & = & x & +4
\end{array}
\qquad
\begin{array}{rcll}
\tau.x + x & = & \tau.x & \tau 1 \\
\tau.x + y & = & \tau.(\tau.x + y) & \tau 2 \\
a.x + a.y & = & a.(\tau.x + \tau.y) & \tau \\
\tau.x + y & \leq & \tau.x & \tau 4 \\
\tau.x + \tau.y & \leq & \tau.(x + y) & \tau 5
\end{array}
$$

$$
\begin{array}{rcll}
nil \setminus a & = & nil & res1 \\
\alpha.x \setminus a & = & nil & res2 \\
 & & \alpha \in \{a, \overline{a}\} & \\
\alpha.x \setminus a & = & \alpha.(x \setminus a) & res \\
 & & \alpha \notin \{a, \overline{a}\} & \\
(x + y) \setminus a & = & x \setminus a + y \setminus a & res4 \\
\lfloor x \rfloor (y) \setminus a & = & \lfloor x \setminus a \rfloor (y \setminus a) & res5
\end{array}
\qquad
\begin{array}{rcll}
nil[S] & = & nil & rel1 \\
(\alpha.x)[S] & = & S(\alpha).x[S] & rel2 \\
(x + y)[S] & = & x[S] + y[S] & rel \\
\lfloor x \rfloor (y)[S] & = & \lfloor x[S] \rfloor (y[S]) & rel4
\end{array}
$$

$$
\begin{array}{rcll}
\tau.\Omega & = & \Omega & \Omega 1 \\
x + \Omega & = & \Omega & \Omega 2 \\
x | \Omega & = & \Omega & \Omega
\end{array}
\qquad
\begin{array}{rcll}
\Omega \setminus a & = & \Omega & \Omega 4 \\
\Omega[S] & = & \Omega & \Omega 5 \\
\lfloor \Omega \rfloor (x) & = & \Omega & \Omega 6
\end{array}
$$

Figure 4: The Inequation System $E$

We write $p \sim q$ to say that $p$ and $q$ pT-16000(Inequation$\Omega$11.76020nInequa0v)20nI98()]TJ$\Omega$/R840.24con

$$\sigma.x = \lfloor nil \rfloor(x) \qquad \sigma 1$$

$$a.x = \lfloor a.x \rfloor(a.x) \qquad \sigma 2$$

$$\lfloor \lfloor x \rfloor(y) \rfloor(z) = \lfloor x \rfloor(z) \qquad \sigma$$

$$\lfloor x \rfloor(y) + \lfloor u \rfloor(v) = \lfloor x + u \rfloor(y + v) \qquad \sigma 4$$

$$\lfloor \tau.x \rfloor(y) = \tau.x \qquad \sigma\tau 1$$

$$\tau.\lfloor x \rfloor(y) = \tau.\lfloor x \rfloor(\tau.y) \qquad \sigma\tau 2$$

$$x + \tau.\lfloor y \rfloor(z) \leq \tau.\lfloor x + y \rfloor(z) \qquad \sigma\tau$$

$$\begin{array}{rcll} x = \sum_I \mu_i.x_i & & y = \sum_J \gamma_j.y_j & \\ x|y & = & \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) + \sum_{\mu_i=\overline{\gamma_j}} \tau.(x_i|y_j) & e1 \\ & & & \\ x = \lfloor \sum_I \mu_i.x_i \rfloor(x_\sigma) & & y = \lfloor \sum_J \gamma_j.y_j \rfloor(y_\sigma) & \\ x|y & = & \lfloor \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) + \sum_{\mu_i=\overline{\gamma_j}} \tau.(x_i|y_j) \rfloor(x_\sigma|y_\sigma) & e2 \end{array}$$

Figure 5: Extra Inequations for System $E$

**Lemma 4.3** *For any TPL processes $p$ and $q$*

$$p \sim q \;\Rightarrow\; p \approx q \;\Rightarrow\; p \eqsim q.$$

**Proof:** The first implication is straightforward and to show the second it is sufficient to prove prove that for any F-test $f$ $p$ *must* $f$ and $p \approx q$ implies $q$ *must* $f$ by induction on the size of $f$. This we leave to the reader. □

We can now consider the soundness of most of our equations with respect to these congruences. The laws $+1, +2, +\ ,$ and $+4$ are called the monoid laws (although commutativity idempotence is not required of monoids). In [Mil90] they are shown to be sound with respect to $\sim$. The equations $res1, res2, res\ , res4, rel1, rel2, rel\ ,$ and $e1$ are also discussed there. Note that this does not necessarily imply that these laws are sound over *TPL*. For example in [Mil90] $x + y \sim y + x$ follows since $x + y \xrightarrow{\alpha} z$ can only be the result of $x \xrightarrow{\alpha} z$ or $y \xrightarrow{\alpha} z$ but not both. We would also have to consider the case where $x \xrightarrow{\sigma} x'$ and $y \xrightarrow{\sigma} y'$ implying $x + y \xrightarrow{\sigma} x' + y'$. However these extra cases are straightforward to check.

We can justify many more of our equations in terms of derivation and observation congruence. The equations $\sigma 1, \ldots \sigma 4, \sigma\tau 1$ and $e2$ are all true of $\sim$. As examples we consider the two equations $\sigma 4$ and $e2$.

- If $x = \lfloor \sum_I \mu_i.x_i \rfloor (x_\sigma)$ and $y = \lfloor \sum_J \gamma_j.y_j \rfloor (y_\sigma)$ then $x|y = \lfloor \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i|y_j) \rfloor (x_\sigma|y_\sigma)$. We examine wait transitions first. For the left hand side these can only be the result of $COM_4$ which can only be applied when $x \stackrel{\sigma}{\longrightarrow} x'$, $y \stackrel{\sigma}{\longrightarrow} y'$ and $x|y \stackrel{\tau}{\not\longrightarrow}$. $THEN_2$ translates this first two conditions into $\tau \notin (\{\mu_i : i \in I\} \cup \{\gamma_j : j \in J\})$ giving $x \stackrel{\sigma}{\longrightarrow} x_\sigma$ and $y \stackrel{\sigma}{\longrightarrow} y_\sigma$. The third condition implies (by $COM_3$) that $\{\mu_i : i \in I\} \cap \{\overline{\gamma_j} : j \in J\} = \emptyset$. These conditions also imply $\lfloor \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i|y_j) \rfloor (x_\sigma|y_\sigma)$ is $\lfloor \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) \rfloor (x_\sigma|y_\sigma)$ and by $THEN_2$ we have $\lfloor \sum_I \mu_i.(x_i|y) + \sum_J \gamma_j.(x|y_j) \rfloor (x_\sigma|y_\sigma) \stackrel{\sigma}{\longrightarrow} x_\sigma|y_\sigma$. The other moves are derived for the left hand side by $COM_1$, $COM_2$, and $COM_3$ and for the right hand side by $THEN_1$. We examine only one case in detail. [illegible] $\stackrel{a}{\longrightarrow} x_a|y$ by $COM_1$. Then by repeated use of $SUM_1$ we have $\sum_I \mu_i$ [illegible] $|y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i|y_j) \stackrel{a}{\longrightarrow} x_a|y$. $THEN_1$ then gives desired move from the [illegible]

The axioms $\tau 1$, $\tau\sigma 2$, $\Omega 1$, $\Omega 4$, and $\Omega 5$ are all true [illegible] ational congruence. The only non-trivial axiom to check is $\tau\sigma 2$ which relates t[illegible]ruence to stability: for any two processes $p$, $q$ $\tau.\lfloor p \rfloor (q)$ and $\tau.\lfloor p \rfloor (\tau.q)$ are obvio[illegible]rvationally equivalent because $\lfloor p \rfloor (q)$ and $\lfloor p \rfloor (\tau.q)$ are observational.Ω52. 199Ω[illegible]TfΩ5.280080TdΩ(to)TjΩ15.6yΩ22.80T

- $x + \tau.\lfloor y \rfloor(z) \leq \tau.\lfloor x + y \rfloor(z)$. We examine $\underline{b} \in SBarb(\tau.\lfloor x + y \rfloor(z))$. If $\underline{b} = \Omega$ then either $x \Uparrow$ or $y \Uparrow$ giving $(x + \tau.\lfloor y \rfloor(z)) \Uparrow$ and $\Omega \in SBarb(x + \tau.\lfloor y \rfloor(z))$. If $\underline{b} = a\underline{b}'$ then either $x \stackrel{a}{\longrightarrow} x'$ with $\underline{b}' \in SBarb(x')$ or $y \stackrel{a}{\longrightarrow} y'$ with $\underline{b}' \in SBarb(y')$. In either case $\underline{b} \in SBarb(x + \tau.\lfloor y \rfloor(z))$. Otherwise $\underline{b} = A\sigma\underline{b}'$. There are three cases to consider.

  - $x \stackrel{\tau}{\Longrightarrow} x'$ with $A = S(x')$ and $\underline{b} \in SBarb(x')$. Then $x + \tau.\lfloor y \rfloor(z) \stackrel{\tau}{\Longrightarrow} x'$ and $\underline{b} \in SBarb(x + \tau.\lfloor y \rfloor(z))$.
  - $y \stackrel{\tau}{\Longrightarrow} y'$ with $A = S(y')$ and $\underline{b} \in SBarb(y')$. Then $x + \tau.\lfloor y \rfloor(z) \stackrel{\tau}{\Longrightarrow} y'$ and $\underline{b} \in SBarb(x + \tau.\lfloor y \rfloor(z))$.
  - $A = S(x) \cup S(y)$ and $\underline{b}' \in SBarb(z)$. Then $S(y)\sigma\underline{b}' \in SBarb(x + \tau.\lfloor y \rfloor(z))$ with $S(y)\sigma\underline{b}' \ll A\sigma\underline{b}'$.

**Definition 4.6** The *depth* of a finite process $d$ written $|d|$ is defined structurally as follows.

- $|\Omega| = |nil| = 0$.
- $|a.d| = 1 + |d|$.
- $|\tau.d| = |d|$.
- $|\sigma.d| = |d|$.
- $|d + e| = max\{|d|, |e|\}$.
- $|\lfloor d \rfloor(e)| = max\{|d|, |e|\}$.
- $|d|e| = |d| + |e|$.
- $|d \setminus a| = |d|$.
- $|d[S]| = |d|$.

□

The depth of a term is supposed to represent the maximum length of a trace from that term, ignoring $\tau$ and $\sigma$ actions. To avoid complication $|d \setminus a|$ is defined as $|d|$ when obviously it could be much less. The reason for ignoring $\sigma$ comes from the line $|\lfloor d \rfloor(e)| = max\{|d|, |e|\}$. If we replace this with the perhaps more intuitive $|\lfloor d \rfloor(e)| = max\{|d|, 1 + |e|\}$ (and adjust $|\sigma.d|$ accordingly) it is difficult to see how to construct a normal form from the choice between the two normal forms $\sum_A a.n_a$ and $\lfloor \sum_B b.m_b \rfloor(m_\sigma)$ without possibly *increasing* the overall depth. Normalisation will be performed using the following measure.

**Definition 4.7** The measure $\prec$ is the preorder defined by

1. $|d| < |f|$ or
2. $|d| = |f|$ and $M_\sigma(d) < M_\sigma(f)$ where $M_\sigma(p)$ denotes the number of occurences of the constuct $\lfloor\ \rfloor(\ )$ in $p$.

We write $d \preceq f$ when either $d \prec f$ or $|d| = |f|$ and $M_\sigma(d) = M_\sigma(f)$, that is when neither the depth or $M_\sigma$ are greater in $d$ than $f$. □

The following fact is used repeatedly when normalising a finite term and so is dealt with separately.

**Lemma 4.8** *For finite sets of normal forms* $\{p_a : a \in A\}$ *and* $\{q_b : b \in B\}$ $(A, B \subseteq Act)$ *there exists normal forms* $r_c$ *such that* $\sum_A a.p_a + \sum_B b.q_b =_E \sum_{A \cup B} c.r_c$ *and* $\sum_{A \cup B} c.r_c \preceq \sum_A a.p_a + \sum_B b.q_b$.

**Proof:** We first show by a case analysis on $n$ that if $n$ is a normal form then there exists a normal form $n'$ such that

1. $n' =_E \tau.n$.
2. $n' \preceq n$.

This in turn is used to show that if $n_1$ and $n_2$ are normal forms then there exists a normal form $n_3$ such that

1. $n_3 =_E \tau.n_1 + \tau.n_2$.
2. $n_3 \preceq \tau.n_1 + \tau.n_2$.

This is proved by induction on the depth of $n_1 + n_2$. The proof of the result is now

- $nf(p) =_E \sum_A a.p_a + \sum_I \tau.p_i$, $nf(q) =_E \sum_B b.q_b + \sum_J \tau.q_j$ .

  $$p + q \quad =_E \quad nf(p) + nf(q)$$

- $d \equiv \lfloor \sum_A a.d_a \rfloor (d_\sigma), \quad q \equiv \lfloor \sum_B b.q_b \rfloor (q_\sigma)$

  Firstly we prove that $n_\sigma \leq_E q_\sigma$. $d_\sigma \ll^c q_\sigma$ follows directly from the $\sigma$-property, Part  of Lemma 4.14 and so by induction we have $d_\sigma \leq_E q_\sigma$.

  Now we prove $\sum_A a.d_a \leq_E \sum_B b.q_b$. Any barb $b\underline{v} \in SBarb(q)$ must be matched by one in $SBarb(d)$ so $B \subseteq A$. Any barb $B\sigma\underline{v} \in SBarb(q)$ must be matched by one in $SBarb(d)$ so $A \subseteq B$ i.e. $A = B$. Also for all $a \in A \quad d_a \ll q_a$. For consider $\underline{v} \in SBarb(q_a)$. Then $a\underline{v} \in SBarb(q)$ and so by $d \ll^c q$ we have $a\underline{u} \in SBarb(d)$ with $a\underline{u} \ll a\underline{v}$. Hence $\underline{u} \ll \underline{v}$ with $\underline{u} \in SBarb(d_a)$. By the $\tau$-preservation propery, Part 1 of Lemma 4.14 $d_a \ll q_a \quad \Rightarrow \quad \tau.d_a \ll^c \tau.q_a$ and so by induction for all $a \in A$ $\tau.d_a \leq_E \tau.q_a$. Hence for all $a \in A$ $a.\tau.d_a \leq_E a.\tau.q_a$ and so $\sum_A a.\tau.d_a \leq_E \sum_A a.\tau.q_a$ which, by $\tau$  gives $\sum_A a.d_a$

– $I \neq \emptyset$.

This last case is the most complicated and we will go through it in some detail. Here $d$ is an unstable normal form and therefore by Lemma 4.11 we may assume $d$ is a strong normal form.

$$d \equiv \sum_{A} a.d_a + \sum_{I \neq \emptyset} \tau.d_i$$

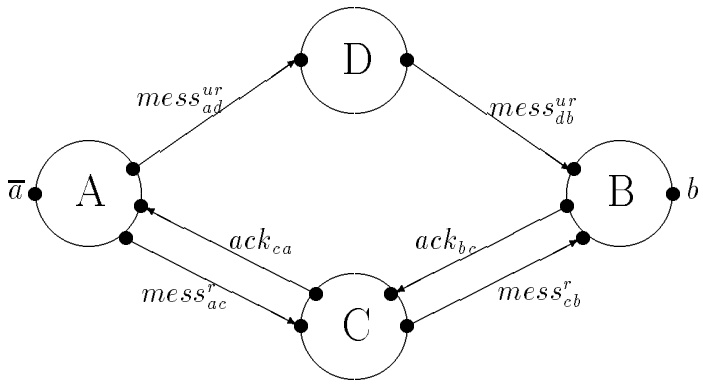
$$\text{where} \quad d_i = \begin{cases} \sum_{B_i} b.d_b^i & \text{or} \\ \lfloor \sum_{B_i} b.d_b^i \rfloor (d_\sigma) \end{cases}$$

with for each $B_i$ we have $B_i \subseteq A$ and $\tau.d_a + \tau.d_a^i =_E \tau.d_a^i$ for any $a \in B_i \cap A$. Further by the stability property, Part 2 of Lemma 4.14, we may assume that

$$q \equiv \sum_{C} c.q_c + \sum_{J \neq \emptyset} \tau.\lfloor \sum_{E_j} e.q_e^j \rfloor (q_\sigma^j).$$

First let us concentrate on the terms $c.q_c$. It is easy to establish that $C \subseteq A$ and for each $c$ in $C$ $d_c \ll q_c$ and therefore from $\tau$-preservation $\tau.d_c \ll^c \tau.q_c$. By induction we have $\tau.d_c \leq_E \tau.q_c$ and therefore $c.d_c \leq_E c.q_c$. This means that for each such $c$ $d \leq_E d + c.q_c$ and, because of $\tau 4$, to complete the theorem it is sufficient to prove $d \leq_E d + \tau.\lfloor \sum$

* $|I'| > 1$.
  We suppose without loss of generality that $I' =$




Accept:

$$A \Leftarrow \overline{a}.mess^{ur}_{ad}.(\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)$$

Reliable Medium:

$$C \Leftarrow \overline{mess^{r}_{ac}}.mess^{r}_{cb}.C + \overline{ack_{bc}}.ack_{ca}.C$$

Unreliable Medium:

$$D \Leftarrow \overline{mess^{ur}_{ad}}.(\tau.D + \tau.mess^{ur}_{db}.D)$$

Transmission:

$$B \Leftarrow \overline{mess^{ur}_{db}}.ack_{bc}.b.ack_{bc}.B + \overline{mess^{r}_{cb}}.b.ack_{bc}.B$$

$$System \Leftarrow (A|B|C|D) \setminus S \quad where \;\; S = Sort(A) \cup Sort(B) \cup Sort(C) \cup Sort(D) \setminus \{a, b\}$$

The message is received by the protocol on port $\overline{a}$. This is done at the module $A$, "Accept". $A$ then sends the message to the unreliable medium, $D$, along port $mess^{ur}_{ad}$. $D$ now either passes the message on to the final module of the protocol ($B$, "Transmission") along port $mess^{ur}_{db}$, or it losses the message. Upon the possible receipt of the message from $D$, $B$ will send an acknowledgement to $A$ via the reliable medium along ports $ack_{bc}$ and $ack_{ca}$. If $A$ does not receive this acknowledgement, then $D$ has lost the message and after one time unit $A$ will retransmit it to the reliable medium along port $mess^{r}_{ac}$. This is then passed onto $B$ by the reliable medium, $C$, along port $mess^{r}_{cb}$. When the environment has accepted transmission of the message from $B$ an acknowledgement is sent to $A$ so that it can reset and be ready to receive another message. This final point avoids $A$ receiving a second message before the delivery of the first. So in the summand $\overline{mess^{ur}_{db}}.ack_{bc}.b.ack_{bc}.B$ of $B$ the first $ack_{bc}$ represents the acknowledgement "message received over unreliable medium, do not resend" whilst the second $ack_{bc}$ represents "message delivered to environment, reset to accept a new message".

We can now prove equationally that

$$System = \overline{a}.(\tau.\sigma.b.System + \tau.b.System)$$

$System = (A|B|C|D) \setminus S$ *by definition*

$= \overline{a}.(mess^{ur}_{ad}.(\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)|B|C|D) \setminus S$ *by d*1

$= \overline{a}.\tau.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)|B|C|(\pi$
$\overline{a}.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)|B|C|(\pi$
$\overline{a}.(X$

$$w = \sum_A a.w_a,\ x = \sum_B b.x_b,\ y = \sum_C c.y_c,\ z = \sum_D d.z_d$$

$$\begin{array}{rcll}
(w|x|y|z) \setminus E & = & ext + int & \\
\text{where} & & & \\
ext & = & \sum_{A \setminus E} a.((w_a|x|y|z) \setminus E) + \sum_{B \setminus E} b.((w|x_b|y|z) \setminus E) + & \\
 & & \sum_{C \setminus E} c.((w|x|y_c|z) \setminus E) + \sum_{D \setminus E} d.((w|x|y|z_d) \setminus E) & \\
int & = & \sum_{A \cap \overline{B}} \tau.((w_a|x_b|y|z) \setminus E) + \sum_{A \cap \overline{C}} \tau.((w_a|x|y_c|z) \setminus E) + & \\
 & & \sum_{A \cap \overline{D}} \tau.((w_a|x|y|z_d) \setminus E) + \sum_{B \cap \overline{C}} \tau.((w|x_b|y_c|z) \setminus E) + & \\
 & & \sum_{B \cap \overline{D}} \tau.((w|x_b|y|z_d) \setminus E) + \sum_{C \cap \overline{D}} \tau.((w|x|y_c|z_d) \setminus E) & d1
\end{array}$$

$$w = \sum_A a.w_a + \sigma.w$$

$$
\begin{aligned}
&= \tau.\sigma.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + mess^r_{ac}.\overline{ack_{ca}}.A)|B|C|D) \setminus S \qquad \textit{by } d2 \\
&= \tau.\sigma.\tau.(\overline{ack_{ca}}.A|B|mess^r_{cb}.C|D) \setminus S
\end{aligned}
$$

# 6 Related Work

There is now an extensive literature on timed process algebras which can be classified from many different viewpoints. For a general discussion on the varieties of timed process algebras the reader is refered to [Je91] but from the purely syntactic level they can be viewed as extensions of the three main process algebras, *ACP, CSP* and *CCS*, each of which represent three somewhat different approaches. For example [BB89] presents a real-time extension of *ACP*, [Re88] contains an extension of *CSP* called *Timed CSP* while *CCS* is the starting point for [MT90a] where the process algebra *TCCS* is defined. Moreover the starting point determines to some extent the type of work reported in these papers. In [Re88] a denotational model for *Timed CSP* is presented, reflecting the fact that much of the work on *CSP* is based on a denotational approach to semantics. Similarly the concern of the *ACP* school of semantics with algebraic theories influences the approach taken in [BB89] while the operational viewpoint, which underlies much of the research on *CCS* is reflected in [MT90a]. However in subsequent work by researchers from these schools this distinction is much less clear. For example in [Gr89] an operational semantics is given to a real-time extension of *ACP* while in [Sch91] *Timed CSP* is considered from the operational point of view of testing.

It is perhaps more fruitful to classify the different approaches by their view of time and the way it is represented semantically. Here the *ACP* and *CSP* approaches, as expounded in [BB89, Re88] respectively, have much in common. They both take time to be real-valued and, at least semantically, associate time directly with actions, as indeed is the case with [QAF89]; Thus actions occur at some specific point in time. This approach is very different from ours as can be seen if we try to compare *TPL* with *Real-time ACP* and Timed CSP using the informal terminology of the introduction.

$\lfloor \_ \rfloor(\_)$ operator comes. Although they pay much attention to showing that their language is of use in describing realistic phenomena they also develop an equational theory for strong bisimulation. Neither of [NRSV92], [MT90] assume *maximal progress* but in its place they have *insistent* actions, i.e. actions which will not delay until the next time cycle. Needless to say the presence of insistent actions means that in general processes are not *patient*, in the informal terminology of the introduction. It seems that in timed process algebras in general either *maximal progress* is assumed or insistent actions are allowed; this is reasonable as both provide a mechanism for forcing actions to happen.

The language presented in [Yi90, Yi91] is the closest in spirit to our language; in fact it can in some sense be viewed as a real-time version of *TPL* as it assumes that actions are instantaneous in addition to *time determinism, maximal progress* and *patience*. However as with [Gr89, MT90a, NRSV92] its semantic theory is based on bisimulation theory. It is also somewhat more expressive than *TPL* in that, roughly speaking, it has prefix constructs of the form

$$a$$

[BW90a] Baeten, J. and Weijland, W., "Process Algebra", Cambridge University Press, 1990.

[BW90b] Baeten, J. and Weijland, W., "Applications of Process Algebra", Cambridge University Press, 1990.

[BC84] Berry, G. and Cosserat, L., "The ESTEREL Synchronous Programming Language and its Mathematical Semantics", Technical Report 842, INRIA, Sophia-Antipolis, 1988.

[BW89] Burns, A. and Wellings A., "Real-Time Systems and their Programming Languages", Adison-Wesley, 1989.

[CH88] Cleaveland, R. and Hennessy, M., "Priorities in Process Algebras", Information and Control, vol 87, nos 1/2, July/August, pp 58–77, 1990.

[CPW86] Cohen, B., Pitt, D.H. and Woodcock, J.C.P., "The Importance of Time in The Specification of OST Protocols", Technical Report, NPL, London, 1986.

[DS89] Davies, J. and Schneider, S., "An Introduction to Timed CSP", Technical Report, PRG, Oxford, 1989.

[dNH84] De Nicola, R. and Hennessy, M., "Testing Equivalence for Processes" *Theoretical Computer Science* vol. 4, pp 8 -1 , North-Holland, 1984.

[GB87] Gert, R. and Boucher, A., "A timed failures model for extended communication processes", *Springer-Verlag Lecture Notes in Computer Science* vol.267, pp 95-114, 1986.

[vG88] van Glabbeek, R., "The Linear Time-Branching Time Spectrum", Proc. CONCUR90, Lecture Notes in Computer Science, vol 458, pp 278–297, Springer-Verlag, 1990.

[Gr89] Groote, J.F., "Specification and Verification of Real Time Systems in ACP" Technical Report CS-R9015, CWI, Amsterdam, 1989. An extended abstract appeared in L. Logrippo, R.L. Probert and H. Ural, editors, *Proceedings* $10^{th}$ *International Symposium on Protocol Specification, Testing and Verification*, Ottawa, pages 261–274, 1990.

[He88] Hennessy, M., "Algebraic Theory of Processes", MIT Press, Cambridge, 1988.

[He8 ] Hennessy, M., "Synchronous and Asynchronous Experiments on Processes", *Information and Control*, Vol 59, No 1- , pp 6-8 , 198 .

[He81] Hennessy, M., "A Term Model for Synchronous Processes", *Information and Control*, Vol 51, No 1, pp 58-75, 1981.

[HR90] Hennessy, M. and Regan, T., "A Temporal Process Algebra" University of Sussex Computer Science Technical Report 2:90, 1990.

[Hoa85] Hoare, C.A.R., "Communicating Sequential Processes", Prentice-Hall, 1985.

[Pn85] Pneuli, A., "Linear and Branching Structures in the Semantics and Logics of Reactive Systems", *Springer-Verlag Lecture Notes in Computer Science* vol.194, pp 15- 2, 1985.

[QAF89] Quemada, J., Azcorra, A. and Frutos, D., "TIC: A Timed Calculus for LOTOS", Technical Report, Madrid, 1989.

[Re88] Reed, G.M., "A Hierarchy of Domains for Real Time Distributed Computing" Technical Report, Oxford, 1988.

[Rea91] Regan, T., "Process Algebras for Real-Time Systems", PhD Thesis University of Sussex, 1991.

[RR86] Reed, G.M. and Roscoe, A., "A Timed model for communicating sequential processes", *Springer-Verlag Lecture Notes in Computer Science*, vol.226, pp 14- 2 , 1986.

[RS88] Rudkin, S. and Smith, C.R., "A Temporal Enhancement for LOTOS" British Telecom R and T ,1988.

[Sch86] Schmidt, D.A., "Denotational Semantics", Allyn and Bacon, 1986.

[Sch90] Schneider, S.A., "Correctness and Communication of Real-Time Systems", PhD Thesis, PRG, University of Oxford, 1990.

[Sch91] Schneider, S.A., "An Operational Semantics for Timed CSP", To appear in Information and Computation, 1992.

[Ste88] Steggles, P., "A Suggestion for a New Temporal LOTOS Semantics", Technical Report, GEC, 1988.

[Wi85] Winskel, G., "A Complete Proof System for SCCS With Modal Assertions" Technical Report, Cambridge, 1985.

[Yi90] Yi, W., "Real-Time Behavior of Asynchronous Agents", *Springer-Verlag Lecture Notes in Computer Science*, vol.458, pp 502-520, 1990.

[Yi91] Yi, W., " A Calculus of Real Time Systems", Ph.D Thesis, Chalmers University, 1991.

[Ze89] Zedan, H. (ed), "Real -Time Systems Theory and Applications", North-Holland, 1989.

[Jo89] Joseph, M., "Time and Real-time in Programs", *Springer-Verlag Lecture Notes in Computer Science* vol.405, FST & TCS 9, Bangalore, 1989.

[Ch91] Chen, L., "Decidability and Completeness in Real-Time Processes", Technical Report, LFCS, Edinburgh, 1991.

[Je91b] Jeffrey, A., "A Linear Time Process Algebra", CAV 91, 1991.

[MT91] Moller, F. and Tofts, C., "Relating Processes With Respect to Speed", *Springer-Verlag Lecture Notes in Computer Science* vol.527, CONCUR 91, pp 424-4 8, 1991.

[Kl91] Klusener, A.S., "Completeness in Real Time Process Algebra", *Springer-Verlag Lecture Notes in Computer Science* vol.527, CONCUR 91, pp 96-110, 1991.