1 Labelled Transition Systems

In this series of lectures we will be interested in the extending an existing semantic theory of standard process algebras to encompass at least some aspects of timed systems. Accordingly in this first section we will review the situation for standard "untimed process algebras".

In general a process algebra may be viewed as a specification or description language for communicating concurrent systems which emphasises their conceptual structure. Essentially a process algebra consists of a collection of combinators for constructing new descriptions from existing ones together with a set of laws or equations for manipulating these descriptions. A large number of process algebras have by now been proposed in the literature; we will try to avoid getting embroiled in the details of these individual languages, each with their own advantages and disadvantages, by simply choosing to work with our favourite, *CCS*, [Mil89]; however most of what we say is equally applicable to other process calculi such as *CSP* and *ACP*. We will also work as much as possible at the more abstract level of intensional operational semantics.

We describe concurrent systems in terms of their ability to perform *action* . For the most part the nature of these actions will be unspecified and we will simply assume a set of possible actions *Act*; typically these are some form of synchronisations with other concurrent systems. But we will have need for a special action symbol τ, which we assume is not in *Act*, to represent internal synchronisations or activity of a system. For convenience we use Act_τ to represent $Act \cup \{\tau\}$, ranged over by α, while a ranges over the external actions *Act*. The ability of a concurrent system to perform these uninterpreted actions can be conveniently represented in terms of *labelled tran ition y tem* :

Definition 1.1 A labelled transition system is a triple $< P, Act_\tau, \longrightarrow>$ where

P is a set of process states

Act_τ is, as already explained, a set of external actions Act and special action τ

$\longrightarrow$ is a subset of $P \times Act_\tau \times P$; we write $p \xrightarrow{\alpha} q$ instead of $(p, \alpha, q) \in \longrightarrow$

$\square$

Intuitively $p \xrightarrow{\alpha} q$ means that in the state p the process may perform the action α and thereby be transformed into the the state q.

An arbitrary process algebra may be given an intensional semantics by interpreting it as a labelled transition system. A standard method for doing so is by structural operational semantics, [Plo81]. The simplest way to explain this is to consider an example, the language *CCS*. In *CCS* communication or synchronisation is a binary operation, i.e. it involves only two participants, one sending the synchronisation signal and the other receiving it. Accordingly we assume that *Act* is equipped with a complementation operation $\overline{a}$ and informally we view $\overline{a}$ as the action of sending a synchronisation signal on a virtual communication channel a while the action a represents its receipt.

The terms of the language are defined by

$$\begin{aligned} p \ ::= \ & \Omega \mid nil \mid P \mid \alpha.p \\ & \mid p + p \mid p \mid p \\ & \mid p[S] \mid p\backslash A, \ A \subseteq Act \end{aligned}$$

We have the constant processes *nil* which can do no actions and Ω which can only diverge and we are allowed the use of process identifiers such as P which will have associated with them a definition of the form

$$P \Leftarrow t.$$

The combinators of the language are

action prefixing: $\alpha.p$ is a process which can perform the action α and then become the process p

choice: $p + q$ is the process which can act like p or q

parallel: $p \mid q$ is the process which consists of two subprocesses p and q running in parallel

re triction: $p\backslash A$ is a process which acts like p except that all actions in A and their complements are local to p

renaming: $p[S]$ is a process which acts like p except that the actions performed are relabelled using the relabelling function S; it is assumed that S is a function over *Act* which preserves complementation and which is almost everywhere the identity. We will also assume that $S(\tau) = \tau$.

When writing terms in this language we will observe the usual rules of precedence between the operators,

$$\backslash A \;=\; [S] \;>\; \text{prefixing} \;>\; \mid \;>\; +$$

and we will usually omit trailing occurrences of *nil*.

This informal description of the intended meaning of the combinators is made precise by providing the language with an operational semantics. This involves defining a relation $\xrightarrow{\alpha}$ between process terms for each action α. To do so we assume the existence of a *declaration*, i.e. a set of definitions of the form

$$P \Leftarrow p,$$

one for each identifier; the term associated with an identifier in a declaration D will be referred to as its *body*, and denoted by $D(P)$; for convenience we will assume that each occurrence of any identifier in the bodies of a declaration are

$(Op1)$	$\alpha.p \xrightarrow{\alpha} p$	
$(Op2)$	$p \xrightarrow{\alpha} p'$	implies $p + q \xrightarrow{\alpha} p'$
$(Op3)$	$p \xrightarrow{\alpha} p'$	implies $p \mid q \xrightarrow{\alpha} p' \mid q$
$(Op4)$	$p \xrightarrow{\alpha} p'$	implies $p\backslash A \xrightarrow{\alpha} p'\backslash A$ provided A admits α
$(Op4)$	$p \xrightarrow{\alpha} p'$	implies $p[S] \xrightarrow{S(\alpha)} p'[S]$
$(Op5)$	$D(P) \xrightarrow{\alpha} p'$	implies $P \xrightarrow{\alpha} p'$
$(Op6)$	$p \xrightarrow{a} p',\ q \xrightarrow{\overline{a}} q'$	implies $p \mid q \xrightarrow{\tau} p' \mid q'$
$(Op7)$	$\Omega \xrightarrow{\tau} \Omega$	

Figure 1: Operational semantics of *CCS*

the process states consists of all the terms in the language

the next-state relations $\xrightarrow{\alpha}$ are defined in Figure 1.

Alternatively we can view every process term p as a labelled transition system obtained by restricting the set of process states to those accessible from p; note that this labelled transition system is rooted. Not much distinction will be made between these two slightly different views of the intensional semantics of *CCS*. We will also generally consider the process states of a transition system as simply processes.

As an example of a process description consider the definition

$$\begin{aligned} VM &\Leftarrow coin.(VM_t + VM_c) \\ VM_t &\Leftarrow \overline{tea}.VM \\ VM_c &\Leftarrow \overline{coffee}.VM \end{aligned}$$

This is a simple vending machine which when given a coin will perform exactly one of the actions $\overline{tea}$ or $\overline{coffee}$. It has slightly different behaviour than the vending machine defined by

$$\begin{aligned} VM^i &\Leftarrow coin.(\tau.VM_t + \tau.VM_c) \\ VM_t &\Leftarrow \overline{tea}.VM \\ VM_c &\Leftarrow \overline{coffee}.VM \end{aligned}$$

When given a coin this machine will also provide either tea or coffee but, in contrast to VM there is no knowing which will be provided. The difference between these two machines can be seen by putting them in parallel with a user such as

$$U\ er_t \quad \Longleftarrow \quad \overline{coin}.tea.\overline{happy}$$

One can check that the process $(U\ er_t \mid VM)\backslash A$, where $A = \{coin, tea, coffee\}$ will always reach the happy state, i.e. will always perform the action $\overline{happy}$. On the other hand the process $(U\ er_t \mid VM^{i})\backslash A$ may or may not reach this state depending on what the vending machine chooses to provide.

The intensional semantics of a process algebra as expressed in a labelled transition system gives a relatively abstract view of how a process behaves in terms of the actions it performs. However two processes, or process descriptions, could have very different representations as labelled transition systems and still be considered to be *exten ionally* equivalent in the sense of providing more or less the same behaviour to any potential user. For example the process descriptions $(U\ er_t \mid VM)\backslash A$ and $\overline{happy}.nil$ yield different labelled transition systems but one can argue that they should be considered to be extensionally equivalent. Much research has been carried out into what exactly extensional equivalence should mean and a number of viable alternatives have emerged. In these lectures we will concentrate on one possibility, called *te ting equivalence* [Hen88], where informally two processes are deemed to be extensionally equivalent if there is no test which can possibly distinguish between them.

This can be formalised at the level of labelled transition systems. We can view a given labelled transition system as providing processes to be experimented upon while the experimenters are also furnished by a labelled transition that is, it is either infinite or has a maximal element $e_m \mid p_m$ from which no further derivation can

$$\begin{array}{lll}
(W1) & a.p \xrightarrow{\sigma} a.p & \\
 & nil \xrightarrow{\sigma} nil & \\
(W2) & p \xrightarrow{\sigma} p',\ q \xrightarrow{\sigma} q' & \text{implies } p + q \xrightarrow{\sigma} p' + q' \\
(W3) & p \xrightarrow{\sigma} p',\ q \xrightarrow{\sigma} q', & \\
 & p \mid q \not\xrightarrow{\tau} & \text{implies } p \mid q \xrightarrow{\sigma} p' \mid q' \\
(W4) & p \xrightarrow{\sigma} p' & \text{implies } p\backslash A \xrightarrow{\sigma} p'\backslash A \\
(W5) & p \xrightarrow{\sigma} p' & \text{implies } p[S] \xrightarrow{\sigma} p'[S] \\
(W6) & p \not\xrightarrow{\tau} & \text{implies } \lfloor p \rfloor (q) \xrightarrow{\sigma} q \\
(W7) & D(P) \xrightarrow{\sigma} p' & \text{implies } P \xrightarrow{\sigma} p
\end{array}$$

Figure 2: The passage of time in *TEPL*

It is worth examining in detail the behaviour of the timeout construct. In $\lfloor p \rfloor(q)$ if p can perform any action α then it will do so and the "exception" process q will be discarded. If on the otherhand no opportunity for synchronisation is forthcoming, and this includes the supposition that $p \not\xrightarrow{\tau}$, then when the clock tick arrives the new residual will be q, i.e. under those assumptions $\lfloor p \rfloor(q) \xrightarrow{\sigma} q$. Thus $(a.p \mid \lfloor \overline{a}.q \rfloor(r))\backslash\{a\}$ can only perform a τ move to $(p \mid q)\backslash\{a\}$ while $(b.p \mid \lfloor a.q \rfloor(r))\backslash\{a,b\}$, under the assumption that $a \neq \overline{b}$, can only wait by performing a σ move and become $(b.p \mid r)\backslash\{a,b\}$. Notice that the language does not have a simple delay construct. But it can be implemented as $\lfloor nil \rfloor(p)$; this process can do nothing until the first time cycle when it becomes p. Since this delay construct will be frequently used we introduce an abbreviation for it.

Definition 2.2 We use $\sigma.p$ to denote the term $\lfloor nil \rfloor(p)$. This notation should be intuitive because the only action it can perform is σ to become p. □

With these rules we now have an interpretation of the language *TPL* as a t-labelled transition system where the process states consists of the terms from *TPL* and the next-state relations $\xrightarrow{\mu}$ are defined by the rules just outlined. In order to illustrate these rules we now consider a simple example, again a vending machine.

$$\begin{array}{rcl}
VM_d & \Longleftarrow & coin.\sigma.VM_t \\
VM_t & \Longleftarrow & \lfloor \overline{tea}.VM_d \rfloor (VM_c) \\
VM_c & \Longleftarrow & \lfloor \overline{coffee}.VM_d \rfloor (VM_d)
\end{array}$$

The vending machine will accept a coin as before and then after a time cycle will be in the state VM_t. Here it will produce tea if requested or after another time cycle will produce coffee. The next time cycle will bring the machine back to the original state VM_dbackto the original behaviour is somewhat more complicated than the original vending machine it can still be used successfully by users who want a particular drink. For example $(U\ er_t \mid VM)\backslash A$, where as before $A = \{coin, tea, coffee\}$, will always reach the happy state and the same is true of $U\ er_c$ defined by

$$U\ er_c \quad \Leftarrow \quad \overline{coin}.coffee.\overline{happy}.nil$$

However users have to be a little careful of how long they wait before accepting their drink. For example the user $U\ er_t^2$ defined by

$$U\ er_t^2 \quad \Leftarrow \quad \overline{coin}.\sigma^2.tea.\overline{happy}.nil,$$

where $\sigma^2.p$ is an abbreviation for $\sigma.\sigma.p$, will not reach the happy state although the corresponding user who wants coffee will have no problem.

We now reconsider various properties that one might want to associate with the special action σ. Each of the following lemmas refer to the particular t-labelled transition system determined by *TPL*.

Lemma 2.3 *(Time-determini m) If $p \xrightarrow{\sigma} q$ and $p \xrightarrow{\sigma} r$ then q and r are yntactically identical.*

This is a natural property to associate with the passage of time although there are process algebras which do not have this property, [Gro90].

Lemma 2.4 *(Maximal progre) If $p \xrightarrow{\sigma}$ then $p \not\xrightarrow{\tau}$*

This is the formal counterpart to our fourth informal assumption about the nature of processes. Again there are timed process algebras which do not satisfy this property, [NS90, MT90]. One advantage of assuming maximal progress is that one can easily describe processes in which particular actions are forced to happen; if p can perform the action a then when placed in the context $(\overline{a}.nil \mid [\ \ ])\backslash A$ it will be forced to synchronise.

Lemma 2.5 *(Patience) If $p \not\xrightarrow{\tau}$*

}]⌊

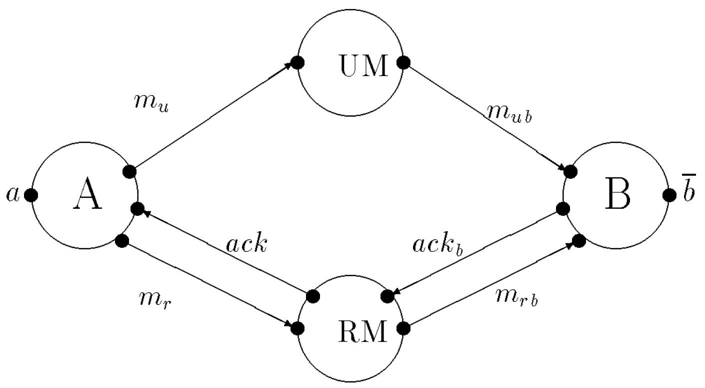


Figure 3: The security costs protocol

states in a t-labelled transition systems and the interaction relation $\mapsto$ is defined in exactly the same way; a parallel operator is defined between arbitrary t-labelled transition systems using the rules $(Op3)$ with α equal to τ, $(Op6)$ and $(W3)$ and $e \mid p \mapsto e' \mid p'$ if $e \mid p \xrightarrow{\tau} e' \mid p'$ or $e \mid p \xrightarrow{\sigma} e' \mid p'$. This leads as before to a semantic preorder defined by:

> in a t-labelled transition system $p \sqsubseteq_t q$ if p *guarantee* e implies q *guarantee* e for every experimenter e,

and the associated equivalence relation is denoted by $\simeq_t$.

Question: Design two tests which show that the processes *VM* and VM_d are incomparable with respect to $\sqsubseteq_t$. □

We aim to show that this new semantic equivalence between timed systems enjoys many of the desirable properties of the original testing equivalence $\simeq$. In particular it has a reasonable mathematical theory and an associated proof system based on equational rewriting.

We end this section with a slightly more extended example of a timed process description using *TPL*. We view it as a pro-typical example of the potential use of *TPL*; only one process involved uses a timing construct and the remainder are described using the

It accepts a message destined for it, called m_r and sends it on to B in the form of the action $\overline{m}_{rb}$. Alternatively it acts as conduit for acknowledgements from B to A: it accepts an acknowledgement from B in the form of the action ack_b and passes it on as $\overline{ack}$ to A. The unreliable medium is defined by

$$UM \Longleftarrow m_u.(\tau.UM + \tau.\overline{m}_{ub}.UM)$$

It accepts a message and then nondeterministically decides to either lose it and return to the original state UM or to send it on to B in the form of the action

3 Acceptances and Barbs

In this section we investigate the properties of processes which determine their ability to guarantee tests. This is essential if we are going to develop a theory of $\simeq_t$; it is easy to demonstrate that two processes are not extensionally equivalent as one only has to provide a test which distinguishes them but to show that they are equivalent we must establish that they guarantee exactly the same set of tests. Rather than consider the reaction of processes with respect to *all* tests we extract the relevant behaviour of their operational semantics which in effect determines which equivalence class of $\simeq_t$ a process belongs to.

Before we start we need some definitions which are given relative to an arbitrary t-labelled transition system although they apply equally well to labelled transition systems since the latter may be considered as t-labelled transition systems where the relation $\xrightarrow{\sigma}$ is empty. For any $s \in (Act_\sigma)^*$ let the relations $\overset{s}{\Longrightarrow}$ be defined in the obvious way:

1. $p \overset{\varepsilon}{\Longrightarrow} p$
2. $p \xrightarrow{\mu} p',\ p' \overset{s}{\Longrightarrow} q$ implies $p \overset{\mu.s}{\Longrightarrow} q$
3. $p \xrightarrow{\tau} p',\ p' \overset{s}{\Longrightarrow} q$ implies $p \overset{s}{\Longrightarrow} q$
4. $p \overset{s}{\Longrightarrow} p',\ p' \xrightarrow{\tau} q$ implies $p \overset{s}{\Longrightarrow} q$.

Let $S(p)$ denote the set $\{\, a \in Act \mid p \overset{a}{\Longrightarrow} \}$ and we say that p is *table* if $p \not\xrightarrow{\tau}$. Finally we say that p *diverge* if there exists an infinite sequence $p \xrightarrow{\tau} p$

if X is Ω then $q \Uparrow$

if X is a finite subset of Act then q is stable and $X = S(q)$.

We use $Acc(p)$ to denote the set of acceptances generated by p.

These definitions enable us to state a condition which is sufficient to ensure that processes in an arbitrary labelled transition system are related extensionally. Of course in a labelled transition system acceptances have no occurrence of the timed action σ.

Theorem 3.1 *In any labelled tran ition y tem $Acc(p) \ll_a Acc(q)$ implie $p \underset{\sim}{\sqsubset} q$.*

Proof: See [Hen88]. □

The converse depends essentially on the expressive power of the labelled transition system in question. For example if in the labelled transition system every process which can do an a action can also do a b action with the same effect then the two processes $a.nil + b.nil$ and $\tau.a.nil + b.nil$ will not be distinguishable; but they will in a labelled transition system which has a process which can not do a b action but can do an a action to a terminated state. We will not go detail about the exact expressive power necessary. Instead let us just say that a labelled transition system is *ufficiently expre - ive* if it contains a denotation for every finite term in *fCCS eq*, i.e. terms in *CCS* which only use the combinators *nil*, $+$ and prefixing by actions in Act_τ.

Theorem 3.2 *In a ufficiently expre ive finitely branching labelled tran ition y tem $p \underset{\sim}{\sqsubset} q$ implie $Acc(p) \ll_a Acc(q)$.*

Proof: See [Hen88]. □

Question: Is this true if the condition on finite branching is dropped ? □

This is the situation, which is well-known, for testing in labelled transition systems. Let us now consider t-labelled transition systems. In fact we will restrict our attention to particular kinds of t-labelled transition systems, essentially those having the properties of *TPL* discussed in the previous section.

Definition 3.3 A t-labelled transition system is called *regular* if it satisfies

1. (Time-determinism) If $p \xrightarrow{\sigma} q$ and $p \xrightarrow{\sigma} r$ then $q = r$
2. (Maximal progress) If $p \xrightarrow{\sigma}$ then $p \not\xrightarrow{\tau}$
3. (Patience) If $p \not\xrightarrow{\tau}$ then $p \xrightarrow{\sigma}$

□

One can easily show that the characterisation of testing in terms of acceptances for

Exam le 3.4 Let p, q denote the terms $a + \tau.b,\ \ \tau.a + \tau.(a + b)$ respectively. Then $Acc(p) \ll_a Acc(q)$ but $p \not\sqsubseteq_t q$; they can be differentiated by the test $\sigma.(\overline{a}.fail + \overline{b})$. This is guaranteed by p because when the clock tick happens all possible synchronisations will have occurred, in particular all τ actions, and so only b is possible. However in q when the clock tick occurs a is also possible and using it to synchronise with the tester leads to the terminal unsuccessful state $fail \mid nil$. $\square$

So in order to characterise testing in regular t-labelled transition systems we need to take into account more information about processes than that contained in their acceptances. But before tackling this problem another aspect of the example deserves comment. Both p and q are terms in *CCS* and are extensionally equivalent for untimed testing, i.e. $p \simeq q$ in the labelled transition system determined by *CCS*, but when considered as timed processes, i.e. as terms of *TPL*, they are not equivalent, $p \not\simeq_t q$. This is because although they are untimed processes which can not be distinguished using untimed tests there is a timed test which can tell them apart!. A more striking example of this phenomenon, taken from [Lan89], is given by the two processes

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

and

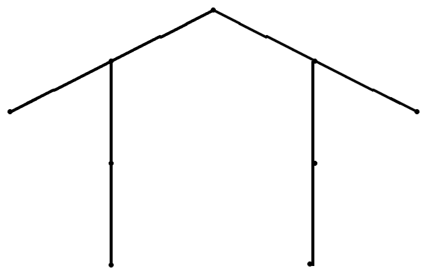
$$coin.(tea + hit.coffee) + coin.(coffee + hit.tea)$$

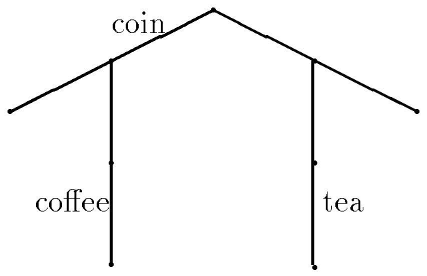
whose behaviour as labelled transition systems are given in Figure 4. They have exactly the same acceptances, namely

$$\begin{array}{rl}
\epsilon\{\{coin\}\} & \\
coin\{\{tea, hit\}\} & coin\{\{coffee, hit\}\} \\
coin.tea\{\emptyset\} & \\
coin.coffee\{\emptyset\} & \\
coin.hit.\{\{tea\}\} & coin.hit.\{\{coffee\}\} \\
coin.hit.tea.\{\emptyset\} & \\
coin.coffee.tea.\{\emptyset\} &
\end{array}$$

But they can be distinguished by the timed test $\overline{coin}.(\overline{tea}. + \sigma.\overline{hit}.\overline{tea})$. This test says that if you can not do a *tea* action immediately after doing a *coin* action then you will be able to do so after performing a *hit* action.

This phenomenon may strike the reader as odd but on reflection it is not unnatural. These processes are really timed systems which when viewed as labelled transitions systems have their timing features abstracted away to such an extent that they can no longer be distinguished. Moreover this abstraction from time is consistent in the sense that so long as we only test using similarly abstracted processes then this level of abstraction can be maintained and we obtain coherent theory of "time-free" process descriptions. [illegible]




coin
coffee
tea

A barb therefore has the form

$$s_1 A_1 s_2 A_2 ... s_k X$$

where each $s_i \in (Act \cup \sigma)^*$, each A_i is a finite subset of Act and the final X is either Ω or again a finite subset of Act. This barb is generated by the process p if there is a derivation of the form

$$p \overset{s_1}{\Longrightarrow} p_1 \overset{s_2}{\Longrightarrow} p_2 \ldots \overset{s_k}{\Longrightarrow} p_k$$

where each p_i is stable, for $1 \leq i < k$ $S(p_i) = A_i$ and if the final X is Ω then $p_k \Uparrow$ and otherwise p_k is also stable with $S(p_k)$ equal to X. We use $Barb(p)$ to denote the set of barbs generated by p. Barbs are ordered in much the same way as acceptances: $\ll_b$ is the least relation over barbs which satisfies

$\Omega \ll_b \underline{b}$ for any barb $\underline{b}$

$A \ll_b A'$ if $A \subseteq A'$

$a\underline{b} \ll_b a\underline{b'}$ if $\underline{b} \ll_b \underline{b'}$

$A\underline{b} \ll_b A'\underline{b'}$ if $\underline{b} \ll_b \underline{b'}$ and $A \subseteq A'$

and is lifted to sets of barbs in exactly the same way as acceptances.

Theorem 3.6 *In any regular t-labelled tran ition y tem if p and q are h-σ-con tant then $Barb(p) \ll_b Barb(q)$ implie $p \sqsubseteq_t q$.* $\square$.

Question: Which of the clauses in the definition of regular t-labelled transition system can be dropped while maintaining this result ? $\square$

Once again the converse depends on the expressive power of the t-labelled transition system in question and let us say that it is sufficiently expressive if it contains a denotation for every term in the language defined using *nil*, + and prefixing by every action in $Act_{\tau\sigma}$. We then obtain

Theorem 3.7 *In any finite branching regular t-labelled tran ition y tem if p and q are h-σ-con tant then $p \sqsubseteq_t q$ implie $Barb(p) \ll_b Barb(q)$.* $\square$

This means that for a large class of t-labelled transition systems barbs capture exactly the ability of "time-free" processes to guarantee tests. Is the same true for arbitrary processes ? Unfortunately the answer is no.

Exam le 3.8 Let p, q be the processes $\tau.(b + \sigma.a) + \tau.(a + c)$ and $a + b$. We will soon be able to check that $p \sqsubseteq_t q$ but their barbs are not properly related. The barb $\{a, b\}a$ is generated by q but p

Theorem 3.9 *In any ufficiently expre ive finite-branching regular t-labelled tran ition y tem* $p \underset{\sim}{\sqsubset}_t q$ *if and only if* $SBarb(p) \ll_b SBarb(q)$. □

Question: Define a closure operator, C, on sets of barbs with the property that $B \ll_b B'$ if and only if $\mathcal{C}(B') \subseteq \mathcal{C}(B)$. □

Question: A t-labelled transition system is *divergence free* if it contains no infinite τ derivations. Define a closure operator $\mathcal{C}'$ on barbs with the property $\mathcal{C}'(B)$ is finite if B is finite and which satisfies $Barb(p) \ll_b Barb(q)$ if and only if $\mathcal{C}'(Barb(p)) \subseteq \mathcal{C}'(Barb(q))$,

$$\begin{array}{lll}
1.\ \textit{Reflexivity} & \dfrac{}{t \leq t} & \\[2ex]
2.\ \textit{Transitivity} & \dfrac{t \leq t',\ t' \leq t''}{t \leq t''} & \\[2ex]
3.\ \textit{Substitution} & \dfrac{\underline{t} \leq \underline{t}'}{f(\underline{t}) \leq f(\underline{t}')} & \text{for every operator } f \\[2ex]
4.\ \textit{Instantiation} & \dfrac{t \leq t'}{t\rho \leq t'\rho} & \text{for every substitution } \rho \\[2ex]
5.\ \textit{Inequations} & \dfrac{}{t \leq t'} & \text{for every inequation} \\
 & & t \leq t' \text{ in } E \\[2ex]
6.\ \Omega - \textit{Rule} & \dfrac{}{\Omega \leq t} &
\end{array}$$

Figure 5: The Proof System

$p \mathrel{\underset{\sim}{\sqsubset}}^{+} q$ if for some action a not occurring in $p,\ q,\ \ a + p \mathrel{\underset{\sim}{\sqsubset}} a + q$.

One can show that $\mathrel{\underset{\sim}{\sqsubset}}$ is preserved by all the operators of *CCS* and moreover is the largest such relation contained in $\mathrel{\underset{\sim}{\sqsubset}}$. Accordingly we transfer our attention to giving an inequational characterisation for $\mathrel{\underset{\sim}{\sqsubset}}^{+}$ and the related congruence $\simeq^{+}$.

Question: Show that for any p, q, r the following is true:

$$\begin{array}{rcl}
((\tau.p + \tau.q) \mid r)\backslash A & \simeq^{+} & \tau.(p \mid r)\backslash A + \tau.(q \mid r)\backslash A \\
(a.p \mid \overline{a}.q \mid r)\backslash\{a\} & \simeq^{+} & \tau.(p \mid q \mid r)\backslash\{a\}
\end{array}$$

□

Many of the required equations are not of any particular interest and are relegated to the appendix; these simply state obvious properties of restriction, renaming, Ω and $+$, most of which are discussed at length in [Mil89]. Here we will concentrate on two aspects of the equational system, the interleaving law which relates parallel composition to nondeterminism and the laws governing τ. The first is given in Figure 6 and applies only to terms of the form $\sum_I \alpha_i.x_i \mid \sum_J \beta_j.y_j$. This parallel term can be rewritten to a nondeterministic term which essentially has a prefix for each of the possible actions which x and y can perform either individually or together. The τ laws, of which there are five, are given Figure 7.

Let E_1 denote the collection of all these inequations.

Theorem 4.1 *For all p, q in fCCS $p \mathrel{\underset{\sim}{\sqsubset}}^{+} q$ if and only if $p \leq_{E_1} q$.*

For $x = \lfloor \sum_I \alpha_i . x_i \rfloor (x_\sigma)$ and $y = \lfloor \sum_J \beta_j . y_j \rfloor (y_\sigma)$

$$x \mid y \quad = \quad \lfloor \textstyle\sum_I \alpha_i . (x_i \mid y) + \sum_J \beta_j . (x \mid y_j) + \sum$$

where A is defined by $A \Longleftarrow \lfloor a \rfloor(A)$ and this can not be proved by the equations even with the help of ω-induction!. We need to add one more rule, called the *Stability* rule, to the proof system:

$$\frac{}{\sum_{i \in I} \alpha_i.t_i \leq S}$$

where S is defined by $S \Longleftarrow \lfloor \sum_{i \in I} \alpha_i.t_i \rfloor(S)$.

A quite useful form of induction is called *Unique Fixpoint Induction.* The form is identical that of Recursion Induction but the inequality is replaced by equality.

> If $\{ P_i \Longleftarrow D_i \mid i \in I \}$ is a declaration and for each $i \in I$ there is a process q_i such that $D_i\{\underline{q}/\underline{P}\} = q_i$ then for each $i \in I$, $P_i = q_i$

Unfortunately it is unsound for the standard extensional equivalences including $\simeq$ and $\simeq_t$. However for a large class of definitions it is sound and it is the rule which tends to be most used in the literature. If we ensure it is used only for definitions where in the bodies all occurrences of process names are all guarded by external actions and do not occur within occurrences of $\mid$ then we will not run into difficulty.

Question: Show that if we allow process names to be guarded by σ then Unique Fixpoint Induction becomes unsound.
Give an example which shows that if we allow occurrences of process names within $\mid$, even if they are guarded by external actions, then the rule is also unsound. $\square$

5 An Example Proof

In this section we discuss how one might use the proof system to prove properties of of timed systems. We use as an example the security costs protocol of section two.

As in the untimed case the proof methodology consists in using the proof system as the basis for a transformation system for manipulating process descriptions. The most common use of the resulting transformation system is to transform a description of a high-level specification of a system into a more detailed description of a proposed implementation. Refering to the example in section two this means transforming $SPEC$ into SYS. Since the proof system is complete in some sense in theory one should be able to transform any two behaviourally related processes into each other using the inequations. But is practice it is virtually essential to augment the set of equations with more useful transformations. For example

$$\begin{aligned} x \mid y &= y \mid x \\ x \mid nil &= nil \mid x \\ (x \mid y) \mid z &= x \mid (y \mid z) \end{aligned}$$

are all sound and are not derivable in the complete proof system. Two other interesting transformations involving parallel are

$$\begin{aligned} ((\tau.x_1 + \tau.x_2) \mid y)\backslash A &= \tau.(x_1 \mid y)\backslash A + \tau.(x_2 \mid y)\backslash A \\ (a.x \mid \overline{a}.y \mid z)\backslash A &= \tau.(x \mid y \mid z)\backslash A \quad \text{if } a \in A \end{aligned}$$

However the most useful transformation rule is a generalisation of the interleaving law to the case where an arbitrary number of processes are running in parallel and a restriction is in force.

Let us start by manipulating SYS. By unwinding each of the recursive definitions, applying $EXP\sigma$ and rewinding we obtain

$$SYS = a.(A_1 \mid RM \mid UM \mid B)\backslash I$$

where A_1 is $\overline{m}_u.\lfloor ack.\, ack.A\rfloor(\overline{m}_r.\, ack.A)$. This procedure of unwinding definitions, applying some form of the expansion theorem and rewinding some of the resulting processes is a very frequently used proof tactic. In fact the unwinding and rewinding of definitions is so persuasive we will not mention it in future; instead we only indicate the variety of expansion theorem used. The next application is ($EXP\tau$) from which we obtain

$$SYS = a.(A_2 \mid RM \mid UM_1 \mid B)\backslash I$$

where A_2 is $\lfloor ack.\, ack.A\rfloor(\overline{m}_r.\, ack.A)$ and UM_1 is $\tau.\, UM + \tau.\overline{m}_{ub}.\, UM$. By applying the rule

$$((\tau.x_1 + \tau.x_2) \mid y)\backslash A = \tau.(x_1 \mid y)\backslash A \; + \; \tau.(x_2 \mid y)\backslash A$$

we obtain

$$SYS = a.(\tau.S_1 + \tau.S_2)$$

where S_1, S_2 denote $(A_2 \mid RM \mid UM \mid B)\backslash I$, $(A_2 \mid RM \mid \overline{m}_{ub}.\, UM \mid B)\backslash I$ respectively.

1. We show $S_1 = \sigma.\tau.\overline{b}.\, SYS$
   Applying (EXP) we obtain

   $$S_1 = \lfloor nil\rfloor((\overline{m}_r.\, ack.A \mid RM \mid UM \mid B)\backslash I).$$

   The proof proceeds by five more applications of various forms of the expansion theorem:

   $$\begin{array}{rcll} S_1 & = & \sigma.(\overline{m}_r.\, ack.A \mid RM \mid UM \mid B)\backslash I & (\, EXP\sigma) \\ & = & \sigma.\tau.(\, ack.A \mid \overline{m}_{rb}.\, RM \mid UM \mid B)\backslash I & (\, EXP\tau) \\ & = & \sigma.\tau.\tau.(\, ack.A \mid RM \mid UM \mid \overline{b}.\, \overline{ack_b}.B)\backslash I & (\, EXP\tau) \\ & = & \sigma.\tau.\tau.\overline{b}.(\, ack.A \mid RM \mid UM \mid \overline{ack_b}.B)\backslash I & (\, EXP\sigma) \\ & = & \sigma.\tau.\tau. & \end{array}$$

where S_3 represents $(\ ack.A \mid\ RM \mid\ UM \mid \overline{ack_b}.B)\backslash I$ and S_4 the process $(\ ack.A \mid RM \mid\ UM \mid \overline{b}.\ \overline{ack_b}.B)\backslash I$. One application of $(\ EXP\sigma)$ gives $S_4 = \overline{b}.S_3$ so we now have

$$S_2 = \tau.\tau.(\overline{b}.\tau.S_3 + \tau.\overline{b}.S_3).$$

On the other hand three applications of $(\ EXP\tau)$ gives

$$S_3 = \tau.\tau.\tau.SYS.$$

which leads to

$$S_2 = \tau.\tau.(\overline{b}.\tau.\tau.\tau.\tau.\ SYS + \tau.\overline{b}.\tau.\tau.\tau.\ SYS).$$

The same τ-reduction rule, $\alpha.x = \alpha.\tau.x$, reduces this to

$$S_2 = \tau.(\overline{b}.\ SYS + \tau.\overline{b}.\ SYS).$$

Another derived equation is $x + \tau.x = \tau.x$ which when applied gives the required

$$S_2 = \tau.\overline{b}.\ SYS.$$

Combining these two sub–proofs we now have

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

Applying the equation $(\sigma\tau 2)$ we obtain the required

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

This completes the proof that $SYS = SPEC$ and as we have seen it consists of a large number of applications of the expansion theorem with periodic interventions using τ-reduction rules. The proof is no different in style than corresponding proofs for time-free processes. In other words we can apply the techniques originally developed for standard process algebras to prove properties of at least some types of time dependent systems. Performing such proofs is undoubtedly tedious but they are eminently suited to mechanical assistance. Software systems have already been developed which help in the development of these proofs, [Lin91, MV89], and they can easily be extended to handle *TPL*. For example the above proof has been carried out by the system *PAM*, [Lin91].

6 Extensions

The language we have investigated is somewhat simple and is best viewed as the core of more extensive and more useful timed languages. This core can be extended in many ways and the choice is probably best made in the light of intended applications. Here we briefly sketch two possibilities; one concentrates on the passage of time and introduces more constructs for the manipulation of the implicit time variable underlying the operational semantics while the other adds *urgent* or *in i tent* actions.

Manipulating Time

To make descriptions in the language more compact one can easily extend the syntax with a variety of notational conventions; a large number may be found in [NS90]and here we will examine a small selection. For any $k \geq 0$ $\sigma^k.p$ can be viewed as a shorthand for $\sigma. \ldots \sigma.p$ which means delay for $k+1$ time-cycles before continuing like p. More generally we can define a *delay tart* operator, $\lfloor p \rfloor^k(q)$. Intuitively $\lfloor p \rfloor^k(q)$ behaves like p provided p can perform an action within k clock cycles and otherwise, after the k^{th} occurrence of the clock tick, it behaves like q. As an example of its use consider

$$\begin{array}{rcl} VM & \Longleftarrow & coin.VM' \\ VM' & \Longleftarrow & \lfloor \sigma^2.\overline{tea}.VM + \sigma^3.\overline{coffee}.VM \rfloor^{30}(VM) \end{array}$$

After receiving a coin tea is ready in two clock cycles while coffee takes three and after thirty seconds the machine reverts to its original state and the coin is lost.

There are two ways of viewing the extension of the language with the operator $\lfloor\ \rfloor^k(\ )$. In the first the syntax of the language is actually extended by adding an infinite set of new operators, $\lfloor\ \rfloor^k(\ )$, one for each $k \geq 0$. The operational semantics of the language must now be also extended to cover processes which use the new operators; this amounts to adding extra clauses to the structural operational rules in Figure 2. The appropriate rules, which reflect our intuition, are

$$\begin{array}{lll} p \xrightarrow{\alpha} p' & \text{implies} & \lfloor p \rfloor^k(q) \xrightarrow{\alpha} p' \\ p \not\xrightarrow{\tau} & \text{implies} & \lfloor p \rfloor^0(q) \xrightarrow{\sigma} q \\ p \xrightarrow{\sigma} p' & \text{implies} & \lfloor p \rfloor^{k+1}(q) \xrightarrow{\sigma} \lfloor p' \rfloor^k(q) \end{array}$$

One can show that with this extension one still obtains a regular t-labelled transition system and therefore the characterisation of testing in terms of barbs also applies to this language. However it is necessary to check that the behaviour preorder is preserved by the new operator, i.e. $p \sqsubset\!\!\!\sim_t p'$, $q \sqsubset\!\!\!\sim_t q'$ implies $\lfloor p \rfloor^k(q) \sqsubset\!\!\!\sim_t \lfloor p' \rfloor^k(q')$. Finally one must find equations which capture entirely the behaviour of the new operator, i.e. equations which when added to the set given in section four provide a complete axiomatisation for the extended language. In this case the required equations are

$$\lfloor t \backslash \rceil$$

The possible computations from S are of the form

$$S \xrightarrow{start} G \xrightarrow{\sigma} \ldots \xrightarrow{\sigma} \xrightarrow{stop} \xrightarrow{display!k} S$$

where k is the amount of lapsed time, i.e. the length of the sequence $\xrightarrow{\sigma} \ldots \xrightarrow{\sigma}$.

A slightly more complicated form of timer may be defined by

$$\begin{array}{rcl} W & \Leftarrow & settime?\mathtt{t}.S(\mathtt{t}) \\ S(\mathtt{t}) & \Leftarrow & start.G(\mathtt{t}) \\ G(\mathtt{t}) & \Leftarrow & \lfloor\, stop@\mathtt{u}.S(\mathtt{t}-\mathtt{u})\rfloor^{\mathtt{t}}(\overline{timeout}.W) \end{array}$$

Here W can receive a time, say 10, and then be started to become $G(10)$. In this state it awaits 10 clock cycles and then performs the timeout action. But while waiting it can also be stopped. For example after 6 clock cycles it is in the state $\lfloor\, stop@\mathtt{u}.S(\mathtt{10}-(\mathtt{u}+\mathtt{6}))\rfloor^{\mathtt{t}}(\overline{timeout}.W)$ where it can perform the action $stop$ to the state $S(4)$. In this state it can be restarted at will and the remaining 4 clock cycles will be counted down - unless it is stopped once more.

Recapitulating the language in question now looks like

$$\begin{array}{rcl} p & ::= & \Omega \mid nil \mid P \mid \alpha.p \mid \quad \mid p+p \mid p \mid p \\ & & \mid p[S] \mid p\backslash A, \end{array}$$

We describe a simple distributed implementation of the vending machine using a timer and a separate unit for brewing the drinks. The timer is defined as follows:

$$\begin{array}{rcl} T & \Leftarrow & \mathit{settime}?\mathtt{t}.W(\mathtt{t}) \\ W(\mathtt{t}) & \Leftarrow & \lfloor\, \mathit{reset}.T \rfloor^{\mathtt{t}-1}(T') \\ T' & \Leftarrow & \overline{\mathit{timeout}}.T + \mathit{reset}.T \end{array}$$

while the independent unit for brewing the tea and coffee is given by:

$$\begin{array}{rcl} C & \Leftarrow & \mathit{coin}.\mathit{settime}!4.B \\ B & \Leftarrow & \sigma^2.\overline{\mathit{tea}}.F + \sigma^3.\overline{\mathit{coffee}}.F + \mathit{timeout}.C \\ F & \Leftarrow & \overline{\mathit{reset}}.C \end{array}$$

Let the implementation be defined by

$$I \Leftarrow (C \mid T)\backslash A$$

where A is the set $\{$ *settime*, *timeout*, *reset*$\}$. The behaviour of this system is slightly different than the other vending machines we have seen. In particular it has some genuinely nondeterministic behaviour. After the fourth clock tick one may still obtain a drink but this is not guaranteed. The complete behaviour is defined by

$$\begin{array}{rcl} S & \Leftarrow & \mathit{coin}.S' \\ S' & \Leftarrow & \lfloor \sigma^2.\overline{\mathit{tea}}.S + \sigma^3.\overline{\mathit{coffee}}.S \rfloor^3(\overline{\mathit{tea}}.S + \overline{\mathit{coffee}}.S + \tau.S) \end{array}$$

We now outline a proof that I is an implementation of S, i.e. $I \simeq^+ S$, by showing how to transform one into the other using a proof system based on that in section four. This involves a use of Unique Fixpoint Induction; we will show that

$$\begin{array}{rcl} I & = & \mathit{coin}.I' \\ I' & = & \lfloor \sigma^2.\overline{\mathit{tea}}.I + \sigma^3.\overline{\mathit{coffee}}.I \rfloor^3(\overline{\mathit{tea}}.I + \overline{\mathit{coffee}}.I + \tau.I) \end{array}$$

for some term I', from which the result will then follow. It is quite straightforward to discover the required term I'. By two applications of an interleaving law we obtain

$$I = \mathit{coin}.\tau.(B \mid W(4))\backslash A.$$

So let I' denote the term $(B \mid W(4))\backslash A$. Using τ-absorption we obtain

$$I = \mathit{coin}.I'$$

and therefore it remains to show

$$I' = \lfloor \sigma^2.\overline{\mathit{tea}}.I + \sigma^3.\overline{\mathit{coffee}}.I \rfloor(\overline{\mathit{tea}}.I + \overline{\mathit{coffee}}.I + \tau.I)$$

By expanding out recursive definitions and doing some rearrangements I' may be rewritten to a form susceptible to an expansion theorem:

$$((\lfloor \mathit{nil} \rfloor(\sigma.\overline{\mathit{tea}}.F + \sigma^2.\overline{\mathit{coffee}}.F + \mathit{timeout}.C)) \mid (\lfloor\, \mathit{reset}.T \rfloor^3(T')))\backslash A.$$

On applying the theorem we obtain

$$I' = \sigma . I_1$$

where

$$I_1 \text{ is } \ ((\sigma . \overline{tea}.F + \sigma^2 . \overline{coffee}.F + \ timeout.C) \mid (\lfloor \ reset.T \rfloor^2 (T')))\backslash A.$$

Repeating this procedure we obtain

Insistent Actions

Throughout these notes we have assumed that processes in our language are *patient* in that they satisfy the condition

$$\text{if } p \not\xrightarrow{\tau} \text{ then } p \xrightarrow{\sigma}$$

which means that processes will wait indefinitely until they can perform a synchronisation. This gives a particular flavour to where $\ll$ is the chaining operator from [Hoa85]. In general $X \ll Y$ is defined to be the process $(X[S_l] \mid Y[S_r]) \backslash \{ mid \}$ where mid

A The standard laws

The first set of equations deal with nondeterminism:

$$\begin{array}{rcll} x+x &=& x & (+1) \\ x+y &=& y+x & (+2) \\ x+(y+z) &=& (x+y)+z & (+3) \\ x+nil &=& x & (+4) \end{array}$$

The next set deal with restriction and renaming:

$$\begin{array}{rcll@{\qquad}rcll} nil\backslash A &=& nil & (res1) & nil[S] &=& nil & (ren1) \\ a.x\backslash A &=& nil & (res2) & (\alpha.x)[S] &=& S(\alpha).x[S] & (ren2) \\ && \text{if } a \text{ or } \overline{a} \in A &&&&& \\ \alpha.x\backslash A &=& \alpha.(x\backslash A) & (res3) & (x+y)[S] &=& x[S]+y[S] & (ren3) \\ && \text{if } \alpha \text{ and } \overline{\alpha} \notin A &&&&& \\ (x+y)\setminus a &=& x\setminus a + y\setminus a & (res4) &&&& \end{array}$$

The final set essentially says that all the operators apart from prefixing by an external action are strict.

$$\begin{array}{rcll@{\qquad}rcll} \tau.\Omega &=& \Omega & (\Omega 1) & \Omega\setminus a &=& \Omega & (\Omega 4) \\ x+\Omega &=& \Omega & (\Omega 2) & \Omega[S] &=& \Omega & (\Omega 5) \\ x \mid \Omega &=& \Omega & (\Omega 3) &&&& \end{array}$$

When the language is extended with the time-out construct $\lfloor\ \rfloor(\ )$ we also need the obvious laws:

$$\begin{array}{rcll} \lfloor\Omega\rfloor(x) &=& \Omega & (\Omega 5) \\ (\lfloor x\rfloor(y))\backslash A &=& \lfloor x\backslash A\rfloor(y\backslash A) & (res5) \\ (\lfloor x\rfloor(y))[S] &=& \lfloor x[S]\rfloor(y[S]) & (ren4) \end{array}$$

References

[BB92] J. C. M. Baeten and J. A. Bergstra. Discrete time process algebra. Technical Report P9208, University of Amsterdam, 1992.

[BG88] G. Berry and G. Gonthier. The synchronous programming language ESTEREL: design, semantics, implementation. Report 842, INRIA, Centre Sophia-Antipolis, Valbonne Cedex, 1988. To appear in *Science of Computer Programming*.

[DH84] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[Gro90] J.F. Groote. Specification and verification of real time systems in ACP. Report CS-R9015, CWI, Amsterdam, 1990. An extended abstract appeared in L. Logrippo, R.L. Probert and H. Ural, editors, *Proceeding* 10th *International Sympo ium on Protocol Specification, [illegible] tola and*

[MV89] S. Mauw and G.J. Veltink. An introduction to PSF_d. In J. Díaz and F. Orejas, editors, *TAPSOFT89, vol 2*, volume 352 of *Lecture Note in Computer Science*, pages 272–285, 1989.

[NS90] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. Technical Report RT-C26, Laboratoire de Génie Informatique de Grenoble, 1990. to appear in Information and Computation.

[Plo81] G.D. Plotkin. A structural approach to operational