INFORMATION AND COMPUTATION 117, 221–239 (1995)

A Process Algebra for Timed Systems*

MATTHEW HENNESSY AND TIM REGAN

University of Sussex, Falmer, Brighton BN1 9QH, United Kingdom

A standard process algebra is extended by a new action σ which is meant to denote *idling* until the next clock cycle. A semantic theory based on testing is developed for the new language. This is characterised in terms of *barbs*, a variety of ready traces and also characterised as the initial theory generated by a set of equations.

1. INTRODUCTION

Process algebras are structured high-level description languages for concurrent systems [Mil89, Hoa85, BW90a]. They consist of a small number of constructors or combinators for building processes together with a facility for recursive definitions. They have a range of well-developed semantic theories and related proof systems associated with them and they have been shown to be reasonably succesful for both the specification and verification of concurrent systems, [BW90b]. Intuitively they view processes as objects which are capable of performing "abstract actions" which are usually interpreted as the input and output of values or signals along communication channels. These capabilities are expressed in terms of "next-state relations," $\longrightarrow$, between processes; $p \xrightarrow{a} q$ represents the fact that the process p can perform the action a and thereby evolve into the process q. This is a relatively abstract interpretation of process. For instance, there is no mention of the length of time the action a takes, or when the action occurs or indeed that it actually occurs at all; $p \xrightarrow{a} q$ merely says that the process p has the capability of performing the action a. However, this abstract view turns out to be a major contributing factor to the success of process algebras; it enables one to describe systems at different levels of abstraction and to relate these different descriptions via semantic equivalences. For example one high-level description S could be viewed as a desired specification of a system and a lower-level description I a description of an actual implementation and proving S semantically equivalent to I amounts to showing that the implementation satisfies the required specification.

Time is often an important aspect of the description of many concurrent systems but it is not directly represented in any of the standard process algebras such as CCS, CSP, and ACP. The introduction of aspects of time into the setting of process algebras has received much attention in recent research and not surprisingly, considering the fact that time is a complex subject, there have been many proposals [BB91, DS89, NS94, MT90, Re88, Yu90]. This paper presents another proposal. Our viewpoint may best be explained by contrast with the approaches of say [BB91, Re88]. These papers suggest 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 been 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 σ action. The execution of this σ 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 authors acknowledge the financial support of British Telecom RT6224, the ESPRIT II-BRA project CONCUR, and the SERC.

0890-5401/95 $6.00

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 σ. 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 [Mil83]. We continue to use this assumption; using the syntax of CCS, the process $a.p$ can idle, i.e., it can perform a σ 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 instanteneous is retained in our language since we have a distinguished action σ 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, $(a.p + b.p)|\bar{a}.q$ cannot idle; the communication via the a channel must occur. However we are not simply giving a mild reinterpretation to CCS. Because of the presence of σ in the language we can express processes whose behaviour is, at least to some extent, time-dependent. The new action does not only indicate idleness but also forced delay; $\sigma.a.p$ is a process which can do nothing until the first clock cycle and from that moment on it offers an a action.

Thus our approach to the introduction of time into process algebras may be characterised by five intuitive properties:

1. *discrete time*: in our language time proceeds in discrete steps represented by occurrences of the action σ,
2. *time determinism*: we assume that the passage of time is deterministic,
3. *actions are instantaneous*: time is not associated directly with communication actions but occurs independently,
4. *patience*: processes will wait indefinitely until they can communicate,
5. *maximal progress*: processes communicate as soon as a possibility for communication arises.

Of course none of these assumptions are necessary in a timed process algebra and in our comparison with related work we will discuss languages in which combinations of these assumptions are dropped. However, we hope to convince the reader that their adoption leads to a calculus which

1. On the one hand is mathematically tractable; we demonstrate this by extending the theory of testing from [dNH84, He88] to this timed setting. This theory may be characterised equationally in a manner which differs only slightly from a standard theory of CCS [dNH84]; moreover there is a close connection with the the theory of refusals, [Ph87].

2. On the other hand may be successfully applied to certain application areas; we demonstrate this by treating a relatively simple example of a protocol in which time plays a small but significant role. Further more substantial examples may be found in [Rea91].

As stated previously we do not expect our calculus to be applicable to all manner of timed systems. But we believe it is applicable; moreover it offers the advantage of relative simplicity with a fully developed semantic theory and therefore we hope that it provides a sound basis on which to develop more extensive theories of timed systems.

We end this introduction with an outline of the contents of the remainder of the paper. In the next section we give the syntax of our timed process algebra TPL, which stands for *Timed Process Language*, together with an operational semantics. Using this operational semantics we then define an operational preorder on timed processes bases on the *must* testing from [dNH84, He88]. This is a standard application of the testing scenario from [He88] but here the tests may use the timing constructs from TPL and therefore the power of testing is considerably increased. In the next section, Section 3, we give an alternative characterization of the testing preorder. For the untimed language this alternative characterization is given in terms of *acceptances* [He88] which are of the form sA; here s is a sequence of actions a process can perform to a state in which one of the actions from the finite set A can be performed. Because timed tests are more powerful the alternative characterisation for TPL has to take into account more of the behaviour of processes. It is expressed in terms of *barbs* [Pn85, vG90], which are sequences of the form $s_1 A_1 s_2 \cdots s_k A_k$. Section 4 is devoted to an equational characterisation of the behavioural preorder. This is in terms of a proof system which consists of a set of equations, a slight weakening of the equational theory of CCS from [dNH84] together with an infinitary rule for recursively defined processes, again as used in [dNH84, He88] and a new rule for *patient* processes. In the next section we develop a prototypical example of where we believe our simple assumptions about time can be of use. It is straightforward protocol for transferring messages across a faulty medium. More extensive examples can be found in [Rea91]. In the final section we describe some related work on timed process algebras. The literature in this area of research is quite extensive and so we confine our discussion to approaches which are quite similar to ours.

2. SYNTAX AND BEHAVIOURAL SEMANTICS

In this section we present the process algebra TPL (Temporal Process Language) formally and develop a behavioural theory of these processes based on "must" testing [He88]. We define the language as closed terms built from a set of constructors, give an operational semantics for the language in terms of *labelled transition systems* and finally define a behavioural preorder based on testing.

The abstract syntax of the language is given by the BNF definition

$$t ::= nil \mid \Omega \mid x \mid \sigma.t \mid \lfloor t \rfloor(t) \mid a.t \mid \tau.t \mid$$
$$t + t \mid t|t \mid t[S] \mid t\backslash a \mid recx.t,$$

where a ranges over Act, a set of actions not containing the distinguished actions τ and σ. The operator $recx.\_$ acts in the usual way as a binder for variables and we are mainly interested in closed terms which we call processes. We will use meta-variables p, q, etc. to range over these processes, a, b, c to range over the set of actions Act and Greek letters α, β (but not ω or σ) to range over Act_τ, the union of Act and $\{\tau\}$. We will not often need to talk about a general action from $Act \cup \{\tau\} \cup \{\sigma\}$ and so this will be explicitly stated where necessary.

We give some intuition of these operators, discussing each in turn.

- *nil*. This is the process which is terminated or deadlocked; it can perform no actions from Act_τ but as discussed in the introduction we design our language so that all processes are patient and for this reason *nil* will allow the passage of time, i.e., it can idle indefinitely.

- Ω. This process represents incomplete information or divergence. This incomplete knowledge of a process is catastrophic in that a process whose behaviour is not completely determined will be equivalent to one whose behaviour is completely unknown.

- $a.$. The process $a.p$ can perform an action a and in so doing evolve into the process p. As is usual in CCS style languages there is an overbar or complementary function $Act \rightarrow Act$ which is idempotent and is used to formalize synchronisation. Again because we wish all our processes to be *patient* $a.p$ will be able to idle indefinitely until the a action is requested by that environment.

- $\tau.$. This is the silent or internal action of our language. Since we are imposing the assumption of maximal progress the process $\tau.p$ will not be able to idle in any environment. The τ action represents some internal communication or computation which requires nothing of the environment. When it is possible τ will preempt any 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 $\bar{a}$, we do not want unspecified delay to occur; the communication, the τ move, must fire immediately.

- $\sigma.$. The passage of time is modelled in our system by an occurance of a σ 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 σ as the symbol to represent the passage of time because of its similarity to the 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 τ the operator $+$ behaves in the same way as it does in the CCS setting. The difference comes with the action σ. 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, $+$ is not decided by the action σ. 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 [NS94]. It is similar to the context $\_ + \sigma.\tau.\_$ but is properly decided by the passage of time in favour of the right hand operand. This operator will be used in the complete axiomatisation of the full calculus, although it is also useful in many examples.

- $|$. The parallel bar we use is the handshake and interleaving of CCS. However, σ again behaves differently. When two processes traverse time their composition also does. This is represented by σ being a broadcast event over $|$ and again this is necessary if we wish to ensure that time is deterministic.

- $\backslash a$. This is just the restriction operator of CCS. It is quantified over Act but we often use the shorthand $\backslash A$ to mean $\backslash a_1 \backslash a_2 \backslash a_3 \cdots \backslash a_n$, where $A = \{a_1, a_2, a_3, ..., a_n\}$. Although it has the same syntax as the CSP and LOTOS hiding operators it has a very different operational meaning. For us the context $\_\backslash a$ forbids the action a and $\bar{a}$.

- $[S]$. This is the relabelling operator from CCS. Here S is function from Act to Act which is almost everywhere the identity and which preserves the complement function. In practice we assume that such functions are automatically extended so that $S(\tau) = \tau$ and $S(\sigma) = \sigma$. Relabeling functions enable the reuse of processes in situations demanding the same functionality modulo action names.

From this informal description of the language we see that CCS is a sub-language of TPL and therefore we say that a process from TPL is a CCS process if it does not use the timing constructs σ and $\lfloor\ \rfloor(\ )$.

The operational semantics of processes is given in two parts. The first, in Fig. 1, defines the next state relations, $\xrightarrow{\alpha}$, for each $\alpha \in Act_\tau$. This is a slight generalisation of the standard operational semantics for CCS and the new action σ plays no role. In Fig. 2 the relational $\xrightarrow{\sigma}$ is then defined in terms of these relations. The first rule says that both $a.p$ and nil may delay. This is a perfectly reasonable assumption; if $a.p$ is an environment where no communication via a is possible, then it should be allowed 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 must perform the internal move τ before the next time cycle. The third rule says that $p+q$ may delay if both p and q may delay. Note that the passage of time, i.e., performing a σ action, does not decide between the choice in $p+q$. The fourth rule says that $p \mid q$ may delay if both p and q may delay and no communication between p and q is possible. The other rules are straightforward; the final rule represents the standard methods for handling restriction and recursion while the rule for $\sigma.p$ is perfectly natural.

We now give some examples of processes which may help to explain the influence of σ on the power of the language. In these examples we use the informal notaton of recursive definitions rather than *recx*._. We will also use the standard conventions in writing CCS terms: occurrences of *nil* will often be omitted, action prefixing will have higher precedence than restriction and relabelling, both of which will in turn be higher than $\mid$ which will bind tighter than $+$.

EXAMPLE 2.1. A process that may accept a message and transmit it back to the environment, retransmitting every one time unit until an acknowledgement is received could be written:

$$P \Leftarrow message_{\text{in}}.\overline{message}_{\text{out}}.P'$$

$$P' \Leftarrow ack.P + \sigma.\overline{message}_{\text{out}}.P.$$

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

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

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

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

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

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

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

FIG. 1. Standard operational semantics.

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

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

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

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

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

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

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

FIG. 2. Operational semantics for σ.

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

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

The process Egg_2 is defined so that the egg *will* be unhealthy if left too long before eating:

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

EXAMPLE 2.3. A "leaking counter" is defined informally as

$$C_0 \Leftarrow press.up.C_1$$

$$C_{n+1} \Leftarrow press.up.C_{n+2} + \sigma.down.C_n.$$

It can perform an *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, $(C_0 \mid \overline{press}^k)\backslash press$ 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 $p \xrightarrow{\sigma} q$, $p \xrightarrow{\sigma} q$ respectively. ▮

The informal assumption of *patience* is not as straightforward to capture. Intuitively this should state that if a process p cannot perform a τ action then it must be able to delay, i.e., perform a σ action. But because of the presence of recursive definitions the situation is more complicated. For example, the processes $recx.x$ and Ω can perform no action whatsoever. Intuitively these terms represent underdefined 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

(i) $nil\downarrow, \alpha.p\downarrow, \sigma.p\downarrow$

(ii) $p\downarrow$ implies $(\lfloor p \rfloor(q))\downarrow, (p \mid q)\downarrow, p\backslash a\downarrow, p[S]\downarrow$,

(iii) $p\downarrow, q\downarrow$, implies $(p+q)\downarrow$,

(iv) $t[recx.t/x]\downarrow$ implies $recx.t\downarrow$.

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 ω for reporting success. A test e is applied to a process p by "running" the process $e \mid p$, i.e., allowing it to evolve via τ actions or σ actions. Specifically, a *computation* from $e \mid p$ is a maximal sequence (which may be finite or infinite) of the form

$$e \mid p = e_0 \mid p_0 \mapsto e_1 \mid p_1 \mapsto \cdots$$
$$\mapsto e_i \mid p_i \mapsto \cdots (\text{where} \mapsto = \xrightarrow{\tau} \cup \xrightarrow{\sigma}).$$

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 $e \mid p$,

$$e \mid p = e_0 \mid p_0 \mapsto \cdots \mapsto e_k \mid p_k \mapsto \cdots,$$

there exists some $n \geq 0$ that $e_n \xrightarrow{\omega}$, i.e., e_n can report success, and for every k, $0 \leq k < n$ $e_k \mid p_k\downarrow$. 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 allow occurrences of σ in the tests and these new tests, even when applied to CCS terms, i.e., terms not involving the timing constructs σ and $\lfloor\ \rfloor(\ )$, have more distinguishing power than standard CCS tests. An interesting difference in the power σ vests in testing languages can be found in [La89]:

EXAMPLE 2.7. This example concerns two vending machines (shown diagrammatically in Fig. 3, where σ actions are ignored) with slightly different internal behaviour,

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

and

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

These are equivalent in the standard theory but they can be distinguished by the temporal test $\overline{coin}.(\overline{tea}.\omega + \sigma.\overline{hit}.\overline{tea}.\omega)$, a test which says that if you cannot do a *tea* action immediately after doing a *coin* action then you will be able to do so after performing a *hit* action.

This kind of testing of CCS processes has already been introduced in [Ph87, Ph88] and for LOTOS in [La89]. Indeed, as mentioned, we have borrowed the notation used by Phillips for his stability operator in [Ph88] for our new delay action, although there is a significant difference: his delay operator decides + whereas ours does not. It should be emphasized that both authors introduce these operators

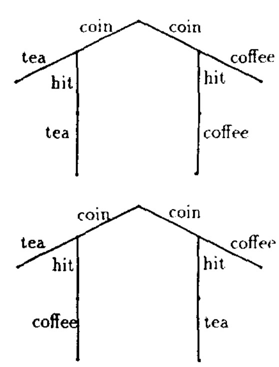


FIG. 3. Langerak's vending machines.

into the test language only and not into the process language.

The preorder $\sqsubseteq$ is not a congruence with respect to the operators $+$ and $\lfloor\ \rfloor(\ )$. For $+$ the example is the usual one: $a.nil \sqsubseteq \tau.a.nil$ but $b.nil + a.nil \sqsubseteq b.nil + \tau.a.nil$ is not true. One can also check that $\lfloor \tau.a \rfloor(b) \simeq \tau.a$ but $\lfloor a \rfloor(b)$ is obviously equivalent to $\tau.a$. However, as we will see, the standard approach to generating a precongruence from $\sqsubseteq$ will also work for our language: let $p \sqsubseteq^{+} q$ if for some a not occurring in p and q $a.nil + p \sqsubseteq a.nil + q$. In the next section we will prove that $p \sqsubseteq^{+} q$ is the largest preorder contained in $\sqsubseteq$ which is preserved by all the operators of the language, nil, $\mu.$, $+$, $\lfloor\ \rfloor(\ )$, $|$, $\backslash a$, and $[S]$.

So now we have a fully fledged process language endowed with a behavioural preorder. In the next section we present an alternative characterisation in terms of barbs [Pn85].

3. ALTERNATIVE CHARACTERISATION

In this section we look at an alternative characterisation of $\sqsubseteq$ over TPL. The corresponding alternative characterisation for the untimed language in [He88] is in terms of "acceptances" of the form sA where s is a sequence of actions a process can perform to arrive at a state and A is the set of next possible actions which can be performed from that state. However, because of the presence of the timing constructs in the tests for TPL, the characterisation now needs to be more complicated. The necessary behavioural information can be encoded in a manner similar to the barbs of [Pn85] which are closely related to the *failure traces* of [vG90]. However, because of the presence of σ in processes and the treatment of divergence, care must be taken in the definition of barbs and how they are associated with processes.

DEFINITION 3.1 (Barbs). Let the set of barbs be the least set satisfying:

1. Ω is a barb
2. if A is a finite subset of Act then A is a barb
3. if $\underline{b}$ is a barb and $a \in Act \cup \{\sigma\}$ then $a\underline{b}$ is a barb
4. if $\underline{b}$ is a barb and A is a finite subset of Act then $A\underline{b}$ is a barb.

Thus a barb may be viewed as a sequence of the form

$$s_1 A_1 s_2 A_2 \cdots s_k A_k$$

or

$$s_1 A_1 s_2 A_2 \cdots s_k \Omega$$

with $1 \leqslant k$, 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. $\Omega \ll \underline{b}$.
2. $A \subseteq B$ implies $A \ll B$.
3. $\underline{b} \ll \underline{b}'$ implies $a\underline{b} \ll a\underline{b}'$.
4. $\underline{b} \ll \underline{b}'$ and $A \subseteq A'$ imply $A\underline{b} \ll A'\underline{b}'$.

This ordering is lifted to sets of barbs by defining

$$A \ll B \Leftrightarrow \forall \underline{b} \in B . \exists \underline{a} \in A . \underline{a} \ll \underline{b}.$$

In order to associate barbs with processes we have to introduce some notation. First the relations $\xrightarrow{\mu}$ are extended to $\xrightarrow{s}$, for $s \in (Act \cup \{\sigma, \tau\})^*$, in the obvious way. Also, $\xRightarrow{\lambda}$, $\lambda \in Act \cup \{\sigma\}$, is used to denote $\xrightarrow{\tau}{}^* \xrightarrow{\lambda} \xrightarrow{\tau}{}^*$ and this is also extended to $\xRightarrow{s}$, $s \in (Act \cup \{\sigma\})^*$, in the natural way. Let

$$S(p) = \{a : p \xrightarrow{a}, a \in Act\}$$
$$Sort(p) = \{a : p \xRightarrow{\lambda} p' \xrightarrow{a}, s \in (Act \cup \{\sigma\})^*, a \in Act\}.$$

We next generalise the strong convergence predicate $\downarrow$ to take internal actions into account: Let $\Downarrow$ be the least predicate on TPL which satisfies

$$p\downarrow \text{ and } \forall p'.(p \xrightarrow{\tau} p' \Longrightarrow p' \Downarrow) \text{ imply } p \Downarrow.$$

We use $p \Downarrow$ to denote the negation of $p \Uparrow$. Finally we say p is *stable* if $p\downarrow$ and $p \not\xrightarrow{\tau}$, i.e., for no p' is $p \xrightarrow{\tau} p'$. Sometimes we will just want to say p cannot perform a τ move, in which case we will call p τ-stable. So, for example, $a.p$ is both stable and τ-stable, while $a.p + \Omega$ is just τ-stable.

Now consider a barb of the form

$$s_1 A_1 s_2 A_2 \cdots s_k B,$$

where B is either Ω or another finite subset of Act. This barb can be generated by the process p if there exists a derivation of stable processes $p_1, p_2, ..., p_k$,

$$p \xRightarrow{s_1} p_1 \xRightarrow{s_2} p_2 \cdots \xRightarrow{s_k} p_k$$

with $A_i = S(p_i)$ for $1 \leqslant i \leqslant k-1$, and if B is Ω then $p_k \Uparrow$ and otherwise p_k is also stable with $B = S(p_k)$. Let $Barbs(p)$ be the set of barbs generated by the process p.

DEFINITION 3.3. For TPL processes p and q let $p \ll^{r} q$ iff $Barb(p) \ll Barb(q)$.

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 σ 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 the 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 $d.(a.nil + c.nil \mid \bar{c}.nil)\backslash c$. Then both p and $d.nil$ have exactly the same simple barbs, namely prefixes of $\varepsilon\{d\}\ d\varnothing$. However, they can be distinguished by the test $\bar{d}.(\bar{a}.nil + \sigma\omega)$.

The theorem is also not true for the entire language TPL, as barbs are too discriminating. For example,

$$\tau.(b.nil + \sigma.a.nil) + \tau.(a.nil + c.nil) \not\ll^r a.nil + b.nil,$$

because the barb $\{a, b\}\, a$ distinguishes them although (as we will soon be able to verify) they are related via $\sqsubseteq$.

To characterise $\sqsubseteq$ over TPL we need to restrict ourselves to *standard* barbs, i.e., barbs of the form

$$s_1 A_1 \sigma s_2 A_2 \sigma s_3 \cdots s_k A_k$$

or

$$s_1 A_1 \sigma s_2 A_2 \sigma s_3 \cdots s_k \Omega,$$

where each s_i is now a member of the set Act^*, i.e., they do not contain occurrences of σ.

Let $SBarb(p)$ be the standard barbs associated with the process p and (by overloading notation) $\ll$ the modification to $\ll^r$ which restricts attention to standard barbs. We may now stake the main result of this section:

THEOREM 3.5 (Alternative Characterisation for TPL). *For p, q in TPL, $p \sqsubseteq q$ if and only if $p \ll q$.*

The remainder of this section is devoted to proving this characterisation of $\sqsubseteq$ over TPL. The proof of the corresponding characterisation for CCS, Theorem 3.4, is omitted as it is very similar.

PROPOSITION 3.6. *For p, q in TPL, $p \ll q$ implies $a \sqsubseteq q$.*

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

$$C \equiv e \mid q = e_0 \mid q_0 \mapsto e_1 \mid q_1 \mapsto \cdots e_i \mid q_i \mapsto \cdots \qquad (1)$$

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

$$C \equiv e \mid q = e_0 \mid q_0 \xrightarrow{\tau}{}^* f_1 \mid r_1 \xrightarrow{\sigma} \xrightarrow{\tau}{}^* f_2 \mid r_2 \cdots e_i \mid q_i \xrightarrow{\sigma} \xrightarrow{\tau}{}^* \cdots, \qquad (2)$$

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

$$\begin{array}{l} q_0 \stackrel{s_1}{\Longrightarrow} r_1 \xrightarrow{\sigma} \stackrel{s_2}{\Longrightarrow} r_2 \xrightarrow{\sigma} \cdots \\ e_0 \stackrel{\bar{s}_1}{\Longrightarrow} f_1 \xrightarrow{\sigma} \stackrel{\bar{s}_2}{\Longrightarrow} f_2 \xrightarrow{\sigma} \cdots \end{array} \qquad (3)$$

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. It gives rise to the sequence of barbs from $SBarb(q)$:

$$\begin{array}{l} \underline{b_1} : s_1 S(r_1) \\ \underline{b_2} : s_1 S(r_1)\, \sigma s_2 S(r_2) \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ \underline{b_k} : s_1 S(r_1)\, \sigma s_2 S(r_2)\, \sigma s_3 \cdots s_k S(r_k) \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \end{array}$$

This sequence may terminate with a barb $\underline{b_m} = s_1 S(r_1)\, \sigma s_2 S(r_2)\, \sigma s_3 \cdots s_m \Omega$ or it may well be infinite. Note that it cannot terminate with a barb $\underline{b_m} = s_1 S(r_1)\, \sigma s_2 S(r_2)\, \sigma s_3 \cdots s_m S(r_m)$, since this would imply that both the experiment and the q are in a stable position with no communication possible and that q is in a convergent state. But this would imply that the computation may proceed via a σ move.

Now suppose that for some $m \geqslant 0$ $S\ Barb(p)$ contains a barb $\underline{a_n} = s_1 S'_1 \sigma s_2 S'_2 \sigma s_3 \cdots s'_n \Omega$, with $n < m$ and s'_n a prefix of s_n, which satisfies $\underline{a_n} \ll \underline{b_n}$. Then this would lead to a derivation from p of the form

$$p = p_0 \stackrel{s_1}{\Longrightarrow} g_1 \xrightarrow{\sigma} \stackrel{s_2}{\Longrightarrow} \cdots \stackrel{s_n}{\Longrightarrow} g_n,$$

where $S(g_i) = S'_i$ for $i < n$ and $g_n \Uparrow$. This could be zipped together with the derivation from e in (3) to obtain a computation from $e \mid p$, which only uses the test states $e_0, e_1, e_2, \ldots$,

$$e \mid p \mapsto e_1 \mid p_1 \mapsto \cdots \mapsto e_l \mid p_l = e_l \mid g_n \xrightarrow{\tau} e_l \mid g_{n+1} \xrightarrow{\tau} \cdots,$$

which will either be infinite or terminate at some $e_l \mid g_{n+j}$ with $g_{n+j} \uparrow$ depending on why $g_n \Uparrow$.

Since p *must* e it follows that for some e_i, $e_i \xrightarrow{\omega}$. Therefore the original computation (1) is succesful. We know that the zipping together of these derivations works because in each state where the $\mapsto$ move is the result of a σ action the relevant S'_i is contained in S_i.

So we may assume that $SBarb(p)$ does not contain any such open barbs. It follows that the original computation (1) and the sequence of barbs $\underline{b}_1, \underline{b}_2 \cdots$ are infinite; a maximal barb b_m would be open and since $p \ll q$, $S\,Barb(p)$ would have to contain an $\underline{a}$ such that $\underline{a} \ll \underline{b}_k$ and necessarily $\underline{a}$ would be open.

So let us consider the infinite sequence of barbs $\underline{b}_1, \underline{b}_2, \ldots$. For each $\underline{b}_k$ we can obtain a barb of p of the form $\underline{a}_k = s_1 S'_1 \sigma s_2 S'_2 \sigma s_3 \cdots s_k S'_k$ with $s'_i \subseteq S(r_i)$ for each $1 \leqslant i \leqslant k$. This gives us the derivation from p

$$p = p_0 \overset{s_1}{\Longrightarrow} g_1 \xrightarrow{\sigma} \overset{s_2}{\Longrightarrow} \cdots \overset{s_k}{\Longrightarrow} g_k,$$

where $S(g_i) = S'_i$ for $1 \leqslant i \leqslant k$. From this we wish to deduce the existence of an infinite derivation from p

$$p = p_0 \overset{s_1}{\Longrightarrow} g_1 \xrightarrow{\sigma} \overset{s_2}{\Longrightarrow} \cdots \xrightarrow{\sigma} \overset{s_k}{\Longrightarrow} g_k \cdots. \tag{4}$$

However, this requires the assumption that the transition system generated by p, based on the "weak moves" $\overset{\lambda}{\Longrightarrow}$, is finite branching. For if it were not, p might have a branch to match each of these barbs while having no infinite branch to match them all. To prove this assumption true we can show that, in the terminology of [Ab91], it is weakly finite branching—that is, for each q accessible from $p\{p' : \exists \mu . q \xrightarrow{\mu} p'\}$ is finite—and also that $\{p' : q \xrightarrow{\tau} p'\}$ is finite. The proof depends on the fact that $q \downarrow$ for each such q and is very similar to the corresponding proof in [dNH84] and is therefore omitted.

The derivation (4) can be combined with the computation from e in (3) to obtain an infinite computation from $e \mid p$ which only uses the test states $e_0, e_1, \ldots$. Again, using p *must* e, we can conclude that the original computation (1) is successful.

This leaves the case when some $e_n \uparrow$, which we leave to the reader. It is sufficient to consider the barb from $S\,Barb(q)$ which characterises the contribution of q up to the appearance of e_n and use the corresponding barb from p to obtain some e_k with $k \leqslant n$ and $e_k \xrightarrow{\omega}$. ∎

We prove the converse by showing that in some sense the ability to generate a particular barb may be captured by an associated test. For every barb $\underline{b}$ and finite set of actions L define the test $e(\underline{b}, L)$ by induction on $\underline{b}$ as follows:

1. $e(\Omega, L) = \tau.\omega$
2. $e(A, L) = \lfloor \sum_{x \in L \setminus A} x.\omega \rfloor (nil)$
3. $e(c\underline{b}', L) = \tau.\omega + \bar{c}.e(\underline{b}', L)$
4. $e(A\sigma\underline{b}', L) = \lfloor \sum_{x \in L \setminus A} x.\omega \rfloor (e(\underline{b}', L))$.

We leave the reader to check the following property of these tests:

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*

$$p \text{ must } e(\underline{b}, L) \Leftrightarrow \exists \underline{a} \in S\,Barb(p)\ \underline{a} \ll \underline{b}.$$

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 $\underline{b} \in S\,Barb(q)$ $\underline{b}' \ll \underline{b}$ for no $\underline{b}'$ in $S\,Barb(p)$. In this case we employ Lemma 3.7, where L is chosen 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 3.7 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 $\underset{\sim}{\sqsubset}^{+}$ 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 be true of the language under examination. Completeness is often harder to prove since it requires that all truths in the language be provable in the proof system.

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

The proof system is defined in Fig. 6. It is essentially inequational reasoning with extra rules for recursive terms, REC and ω – *Induction*. In the latter $App(t)$ denotes the set of finite approximations to t, $\{t^n : n \geqslant 0\}$, 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 $\vdash_E t \leqslant u$ denote that $t \leqslant u$ is derivable in this proof system, $t \leqslant_E u$ that $t \leqslant u$ is derivable with purely inequational reasoning, and finally $t \leqslant_{Er} u$ that $t \leqslant u$ is derivable using *in addition the unfolding rule* REC.

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

$$
\begin{array}{rcllrcll}
x + x & = & x & +1 & \tau.x + x & = & \tau.x & \tau 1 \\
x + y & = & y + x & +2 & \tau.x + y & = & \tau.(\tau.x + y) & \tau 2 \\
x + (y + z) & = & (x + y) + z & +3 & a.x + a.y & = & a.(\tau.x + \tau.y) & \tau 3 \\
x + nil & = & x & +4 & \tau.x + y & \leq & \tau.x & \tau 4 \\
 & & & & \tau.x + \tau.y & \leq & \tau.(x + y) & \tau 5 \\
 \\
nil \setminus a & = & nil & res1 & nil[S] & = & nil & rel1 \\
\alpha.x \setminus a & = & nil \quad \alpha \in \{a, \bar{a}\} & res2 & (\alpha.x)[S] & = & S(\alpha).x[S] & rel2 \\
\alpha.x \setminus a & = & \alpha.(x \setminus a) \quad \alpha \notin \{a, \bar{a}\} & res3 & (x + y)[S] & = & x[S] + y[S] & rel3 \\
(x + y) \setminus a & = & x \setminus a + y \setminus a & res4 & & & & \\
\lfloor x \rfloor(y) \setminus a & = & \lfloor x \setminus a \rfloor(y \setminus a) & res5 & \lfloor x \rfloor(y)[S] & = & \lfloor x[S] \rfloor(y[S]) & rel4 \\
 \\
\tau.\Omega & = & \Omega & \Omega 1 & \Omega \setminus a & = & \Omega & \Omega 4 \\
x + \Omega & = & \Omega & \Omega 2 & \Omega[S] & = & \Omega & \Omega 5 \\
x | \Omega & = & \Omega & \Omega 3 & \lfloor \Omega \rfloor(x) & = & \Omega & \Omega 6
\end{array}
$$

FIG. 4. The inequation system E.

equivalences, derivation congruence and observational congruence [Mil90].

DEFINITION 4.1. *Derivation congruence* is the largest equivalence satisfying

$$
\langle p, q \rangle \in S \Rightarrow \begin{array}{ll} \text{(i)} & \forall p'.p \xrightarrow{\lambda} p'.q \xrightarrow{\lambda} p' \\ \text{(ii)} & p \Downarrow \Leftrightarrow q \Downarrow \end{array}
$$

where $\lambda \in Act \cup \{\tau, \sigma\}$.

We write $p \sim q$ to say that p and q are derivation congruent.

DEFINITION 4.2. *Observational equivalence* is the largest equivalence satisfying

$$
\begin{array}{rcll}
\sigma.x & = & \lfloor nil \rfloor(x) & \sigma 1 \\
a.x & = & \lfloor a.x \rfloor(a.x) & \sigma 2 \\
\lfloor \lfloor x \rfloor(y) \rfloor(z) & = & \lfloor x \rfloor(z) & \sigma 3 \\
\lfloor x \rfloor(y) + \lfloor u \rfloor(v) & = & \lfloor x + u \rfloor(y + v) & \sigma 4 \\
 \\
\lfloor \tau.x \rfloor(y) & = & \tau.x & \sigma\tau 1 \\
\tau.\lfloor x \rfloor(y) & = & \tau.\lfloor x \rfloor(\tau.y) & \sigma\tau 2 \\
x + \tau.\lfloor y \rfloor(z) & \leq & \tau.\lfloor x + y \rfloor(z) & \sigma\tau 3
\end{array}
$$

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

FIG. 5. Extra inequations for system E.

1. *Reflexivity*		$\dfrac{}{t \le t}$	
2. *Transitivity*		$\dfrac{t \le t',\ t' \le t''}{t \le t''}$	
3. *Substitution*	(a)	$\dfrac{t \le t'}{f(t) \le f(t')}$	for every $f \in \{\mu.,+,\mid,\backslash a,[S]\}$
	(b)	$\dfrac{t \le t'}{rec\,x.t \le rec\,x.t'}$	
4. *Instantiation*		$\dfrac{t \le t'}{t\rho \le t'\rho}$	for every substitution ρ
5. *Inequations*		$\dfrac{}{t \le t'}$	for every inequation $t \le t'$ in E
6. Ω – *Rule*		$\dfrac{}{\Omega \le t}$	
7. *REC*		$\dfrac{}{rec\,x.t = t[\overline{rec\,x.t}/x]}$	
8. ω – *Induction*		$\dfrac{\text{for every } d \in App(t), d \le t'}{t \le t'}$	
9. *Stability* – *Rule*		$\dfrac{}{\sum_{i \in I} \alpha_i.t_i \le rec\,x.\lfloor \sum_{i \in I} \alpha_i.t_i \rfloor(x)}$	$\alpha_i \in Act_\tau$

FIG. 6. The proof system.

$$\langle p, q \rangle \in S \Rightarrow \begin{array}{ll} \text{(i)} & \forall p'.p \stackrel{\lambda}{\Longrightarrow} p'.\exists q'.q \stackrel{\lambda}{\Longrightarrow} q'.\langle p', q' \rangle \in S \\ \text{(ii)} & p \Downarrow \Leftrightarrow q \Downarrow \end{array}$$

where $\lambda \in Act \cup \{\sigma, \varepsilon\}$.

It is well known that $\approx$ is not preserved by $+$ [Mil90] but if, as usual, we define $p \approx^+ q$ if for some a not appearing in p or q $p + a \approx q + a$, then $\approx^+$ is a congruence with respect to all our operators.

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 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 $+1$, $+2$, $+3$, $+4$ are called the monoid laws (although computativity idempotence is not required of monoids). In [Mil90] they are shown to be sound with respect to $\sim$. The equations *res*1, *res*2, *res*3, *res*4, *rel*1, *rel*2, *rel*3, and *e*1 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 that $x + y \xrightarrow{\sigma} x' + y'$. 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 $\sigma 1$, ..., $\sigma 4$, $\sigma\tau 1$, and $e2$ are all true of $\sim$. As examples we consider the two equations $\sigma 4$ and $e2$.

- $\lfloor x \rfloor(y) + \lfloor u \rfloor(v) = \lfloor x + u \rfloor(y + v)$. First we examine the σ transition possible if x and u cannot perform a τ move. $THEN_2$ gives $\lfloor x + u \rfloor(y + v) \xrightarrow{\sigma} y + v$. It also gives $\lfloor x \rfloor(y) \xrightarrow{\sigma} y$ and $\lfloor u \rfloor(v) \xrightarrow{\sigma} v$, so by SUM_3 the only available σ move from $\lfloor x \rfloor(y) + \lfloor u \rfloor(v)$ is $\lfloor x \rfloor(y) + \lfloor u \rfloor(v) \xrightarrow{\sigma} y + v$. The other transitions from $\lfloor x + u \rfloor(y + v)$ must, by $THEN_1$, come from $x + u$. By SUM_1 and SUM_2 these must be from $x \xrightarrow{\alpha} x'$ or $u \xrightarrow{\alpha} u'$, giving $\lfloor x + u \rfloor(y + v) \xrightarrow{\alpha} x'$ or $\lfloor x + u \rfloor(y + v) \xrightarrow{\alpha} y'$. Now the initial moves (i.e., non-σ moves) of $\lfloor x \rfloor(y) + \lfloor u \rfloor(v)$ are derived from SUM_1 and SUM_2, i.e., from $\lfloor x \rfloor(y)$ and $\lfloor u \rfloor(v)$. $THEN_1$ gives these as the result of $x \xrightarrow{\alpha} x'$ or $u \xrightarrow{\alpha} u'$. So $\lfloor x \rfloor(y) + \lfloor u \rfloor(v) \xrightarrow{\alpha} x'$ or $\lfloor x \rfloor(y) + \lfloor u \rfloor(v) \xrightarrow{\alpha} u'$ as before.

- 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 $x \mid y = \lfloor \sum_I \mu_i.(x_i \mid y) + \sum_J \gamma_j.(x \mid y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i \mid y_j) \rfloor (x_\sigma \mid y_\sigma)$. 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 $x \xrightarrow{\sigma} x'$, $y \xrightarrow{\sigma} y'$ and $x \mid y \not\xrightarrow{\tau}$. $THEN_2$ translates the first two conditions into $\tau \notin (\{\mu_i : i \in I\} \cup \{\gamma_j : j \in J\})$, giving $x \xrightarrow{\sigma} x_\sigma$ and $y \xrightarrow{\sigma} y_\sigma$. The third condition implies (by COM_3) that $\{\mu_i : i \in I\} \cap \{\overline{\gamma_j} : j \in J\} = \varnothing$. These conditions also imply that $\lfloor \sum_I \mu_i.(x_i \mid y) + \sum_J \gamma_j.(x \mid y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i \mid y_j) \rfloor (x_\sigma \mid y_\sigma)$ is $\lfloor \sum_I \mu_i.(x_i \mid y) + \sum_J \gamma_j.(x \mid y_j) \rfloor (x_\sigma \mid y_\sigma)$, and by $THEN_2$ we have $\lfloor \sum_I \mu_i.(x_i \mid y) + \sum_J \gamma_j.(x \mid y_j) \rfloor (x_\sigma \mid y_\sigma) \xrightarrow{\sigma} x_\sigma \mid y_\sigma$. 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. Suppose $x \mid y \xrightarrow{u} x_a \mid y$ by COM_1. Then by repeated use of SUM_1 we have $\sum_I \mu_i.(x_i \mid y) + \sum_J \gamma_j.(x \mid y_j) + \sum_{\mu_i = \overline{\gamma_j}} \tau.(x_i \mid y_j) \xrightarrow{u} x_a \mid y$. $THEN_1$ then gives the desired move from the right.

The axioms $\tau 1$, $\tau\sigma 2$, $\Omega 1$, $\Omega 4$, and $\Omega 5$ are all true of observational congruence. The only non-trivial axiom to check is $\tau\sigma 2$, which relates the congruence to stability: for any two processes p, q, $\tau.\lfloor p \rfloor(q)$ and $\tau.\lfloor p \rfloor(\tau.q)$ are obviously observationally equivalent because $\lfloor p \rfloor(q)$ and $\lfloor p \rfloor(\tau.q)$ are observationally equivalent and the extra condition for congruence is easily checked. The standard τ-laws of CCS may also be derived from ours. $\tau 1$ is the second τ-law of CCS (see for example [Mil90]). The first τ-law, $\alpha.\tau.x = \alpha.x$, follows from our $\tau 2$ (with $y = \tau.x$) and $\tau 3$ (with $y = x$). The third, $\alpha.(x + \tau.y) + \alpha.y = \alpha.(x + \tau.y)$ is derivable as follows:

$$\begin{aligned} \alpha.(x + \tau.y) + \alpha.y &= \alpha.(\tau.(x + \tau.y) + \tau.y) && \tau 3 \\ &= \alpha.(x + \tau.y + \tau.y) && \tau 2 \\ &= \alpha.(x + \tau.y) && +1. \end{aligned}$$

The remaining equations, $\tau 2$, $\tau 3$, $\tau 4$, $\tau 5$, $\sigma\tau 3$, $\Omega 2$, and $\Omega 3$ have to be justified directly in terms of $\ll^c$. We look at two examples:

- $\tau.x + \tau.y \leqslant \tau.(x+y)$. This is not straightforward, but it embodies the idea that nondeterminism is decreased as one moves up the preorder. We prove this sound by induction on the length of barbs.

— $\Omega \in S\ Barb(\tau.(x+y))$. Then $(x+y)\Uparrow$ and so either $x\Uparrow$ or $y\Uparrow$ and so $(\tau.x+\tau.y)\Uparrow$ and $\Omega \in S\ Barb(\tau.x+\tau.y)$.

— $A \in S\ Barb(\tau.(x+y))$. Then either $x \stackrel{\varepsilon}{\Longrightarrow} x'$ with $A = S(x')$, $y \stackrel{\varepsilon}{\Longrightarrow} y'$ with $A = S(y')$, or $A = S(x) \cup S(y)$. In any case this can be matched by $\tau.x + \tau.y$.

— $a\underline{b} \in S\ Barb(\tau.(x+y))$. Then $\tau.(x+y) \stackrel{a}{\Longrightarrow} z$ with $\underline{b} \in S\ Barb(z)$. But then either $x \stackrel{a}{\Longrightarrow} z$ or $y \stackrel{a}{\Longrightarrow} z$ and so $a\underline{b} \in S\ Barb(\tau.x+\tau.y)$.

— $A\sigma\underline{b} \in S\ Barb(\tau.(x+y))$. Now if either $A\sigma\underline{b} \in S\ Barb(x)$ or $A\sigma\underline{b} \in S\ Barb(y)$ we have $A\sigma\underline{b} \in S\ Barb(\tau.x+\tau.y)$. Suppose not, that is, $x \stackrel{\sigma}{\longrightarrow} x'$ and $y \stackrel{\sigma}{\longrightarrow} y'$ with $A = S(x) \cup S(y)$ and $\underline{b} \in S\ Barb(x'+y')$. Then $\underline{b} \in S\ Barb(\tau.(x'+y'))$ and so by induction $\underline{a} \in S\ Barb(\tau.x'+\tau.y')$ with $\underline{a} \ll \underline{b}$. But $S\ Barb(\tau.x'+\tau.y') = S\ Barb(x') \cup S\ Barb(y')$ and so $\underline{a} \in S\ Barb(x')$ or $\underline{a} \in S\ Barb(y')$. Without loss of generality assume $\underline{a} \in S\ Barb(x')$. Then $S(x)\sigma\underline{a} \in S\ Barb(x) \subseteq S\ Barb(\tau.x+\tau.y)$ as required.

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

— $x \stackrel{\tau}{\Longrightarrow} x'$ with $A = S(x')$ and $\underline{b} \in S\ Barb(x')$. Then $x + \tau.\lfloor y \rfloor(z) \stackrel{\tau}{\Longrightarrow} x'$ and $\underline{b} \in S\ Barb(x+\tau.\lfloor y \rfloor(z))$.

— $y \stackrel{\tau}{\Longrightarrow} y'$ with $A = S(y')$ and $\underline{b} \in S\ Barb(y')$. Then $x + \tau.\lfloor y \rfloor(z) \stackrel{\tau}{\Longrightarrow} y'$ and $\underline{b} \in S\ Barb(x+\tau.\lfloor y \rfloor(z))$.

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

We have just shown:

PROPOSITION 4.4. *All the inequations in Fig. 4 and 5 are sound with respect to* $\sqsubseteq^c$.

At this point it is convenient to ignore the soundness of the proof system and instead address completeness.

In common with most completeness proofs in the process algebra literature we start by defining the notion of normal form. These are gleaned from the behaviour of processes and the mechanics of the proof that every term indeed has a normal form. In the following definition of normal forms we make the distinction between stable and unstable processes. If the process is stable then either it changes under the passage of one unit of time or it does not. If the process is unstable then either it is divergent or it has a number of actions available to it before nondeterministically resolving its instability in favour of a stable normal form.

DEFINITION 4.5. A *normal form* (nf for short) is a term of the following inductively defined form.

1. Ω is a normal form.
2. $\lfloor \sum_A a.n_a \rfloor(n_\sigma)$ is a normal form if each n_a is a normal form and n_σ is a normal form.
3. $\sum_B b.n_b + \sum_I \tau.n_i$ is a normal form if each n_b is a normal form and each n_i is a stable normal form.

Taking I to be empty gives normal forms $\sum_A a.n_a$ and also taking A to be empty gives the normal form *nil*. It may also be worth clarifying the notation $\sum_X f(x)$. This is intended as a shorthand for $\sum_{x \in X} f(x)$. We will use and abuse this notation liberally. We also denote by n_a the unique m such that the normal form n can evolve to by performing an a action, i.e., $n \stackrel{a}{\longrightarrow} m$.

In proving that every term can be reduced to a normal form we need a measure on which to perform induction.

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

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

The depth of a term is supposed to represent the maximum length of a trace from that term, ignoring τ and σ actions. To avoid complication $|d\backslash a|$ is defined as $|d|$ when obviously it could be much less. The reason for ignoring σ 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 occurrences of the construct $\lfloor\ \rfloor(\ )$ in p.

We write $d \preccurlyeq f$ when either $d \prec f$ or $|d| = |f|$ and $M_\sigma(d) = M_\sigma(f)$, that is, when neither the depth nor M_σ is greater in d than in 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 exist 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* $\sum_{A \cup B} c.r_c \preccurlyeq \sum_A a.p_a + \sum_B b.q_b$.

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. $n' \preccurlyeq n$.

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. $n_3 \preccurlyeq \tau.n_1 + \tau.n_2$.

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

$$\sum_A a.p_a + \sum_B b.q_b =_E \sum_{A \backslash B} a.p_a + \sum_{B \backslash A} b.q_b + \sum_{A \cap B} c.(\tau.p_c + \tau.q_c) \quad \text{by } \tau 3$$

$$\sum_A a.p_a + \sum_B b.q_b =_E \sum_{A \backslash B} a.p_a + \sum_{B \backslash A} b.q_b + \sum_{A \cap B} c.nf(\tau.p_c + \tau.q_c) \quad \text{by above} \ \blacksquare$$

THEOREM 4.9 (Normal Form Theorem). *Every finite term p has an equationally equivalent normal form* $nf(p)$ *with* $nf(p) \preccurlyeq p$.

Proof. We proceed by induction on $\preccurlyeq$. There are several cases to consider, depending on the structure of p, but we examine only $p + q$ here.

- $nf(p) =_E \lfloor \sum_A a.p_a \rfloor (p_\sigma)$, $nf(q) =_E \lfloor \sum_B b.q_b \rfloor (q_\sigma)$:

$$p + q =_E nf(p) + nf(q) \quad \text{by substitution}$$

$$=_E \left\lfloor \sum_A a.p_a \right\rfloor (p_\sigma) + \left\lfloor \sum_B b.q_b \right\rfloor (q_\sigma) \quad \text{by substitution}$$

$$=_E \left\lfloor \sum_A a.p + \sum_B b.q_b \right\rfloor (p_\sigma + q_\sigma) \quad \text{by } \sigma 4$$

$$=_E \left\lfloor \sum_A a.p_a + \sum_B b.q_b \right\rfloor (nf(p_\sigma + q_\sigma)) \quad \text{by induction on } \preccurlyeq.$$

The result then follows by Lemma 4.8.

- $nf(p) =_E \sum_A a.p_a$, $nf(q) =_E \lfloor \sum_B b.q_b \rfloor (q_\sigma)$:

$$p + q =_E nf(p) + nf(q) \quad \text{by substitution}$$

$$=_E \sum_A a.p_a + \left\lfloor \sum_B b.q_b \right\rfloor (q_\sigma) \quad \text{by substitution}$$

$$=_E \left\lfloor \sum_A a.p_a + \sum_B b.q_b \right\rfloor \left(\sum_A a.p_a + q_\sigma\right) \quad \text{by } \sigma 2 \ \& \ \sigma 4$$

$$=_E \left\lfloor \sum_A a.p_a + \sum_B b.q_b \right\rfloor \left(nf\left(\sum_A a.p_a + q_\sigma\right)\right) \quad \text{by induction on } \preccurlyeq.$$

The result then follows as above by Lemma 4.8.

- $nf(p) =_E \lfloor \sum_A a.p_a \rfloor (p_\sigma)$, $nf(q) =_E \sum_B b.q_b + \sum_{J \neq \varnothing} \tau.q_j$:

$$p + q =_E nf(p) + nf(q) \quad \text{by substitution}$$

$$=_E \left\lfloor \sum_A a.p_a \right\rfloor (p_\sigma) + \sum_B b.q_b + \sum_{J \neq \varnothing} \tau.q_j \quad \text{by substitution}$$

$$=_E \sum_A a.p_a + \sum_B b.q_b + \sum_{J \neq \varnothing} \tau.q_j \quad \text{by } \sigma\tau 3.$$

Again the result follows from Lemma 4.8.

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

$$p + q =_E nf(p) + nf(q) \quad \text{by substitution}$$

$$=_E \sum_A a.p_a + \sum_{I \neq \varnothing} \tau.p_i + \sum_B b.q_b + \sum_{J \neq \varnothing} \tau.q_j \quad \text{by substitution.}$$

The result follows as before from Lemma 4.8. $\blacksquare$

It will be convenient in the completeness theorem to be able to further reduce normal forms. These we call *strong normal forms*.

DEFINITION 4.10. A normal form with the structure $\sum_A a.d_a + \sum_{I \neq \varnothing} \tau.d_i'$, where each d_i is a stable normal form (i.e., $\sum_{B_i} b.d^i_b$ or $\lfloor \sum_{B_i} b.d^i_b \rfloor (d^i_\sigma)$), is a *strong normal form* if

1. each B_i is contained in A and
2. for each a in $A \cap B_i$, $\tau.d_a + \tau.d^i_a =_{\mathrm{E}} d_a$.

We use $snf(d)$ to denote the strong normal form of a term d. We did not include this information in the definition of normal forms, as the translation of a τ-stable process into its associated strong normal form may *increase* the depth of the term. However, we can prove the following lemma.

LEMMA 4.11. *For every normal form* $d = \sum_A a.d_a + \sum_{I \neq \varnothing} \tau.d_i$ *with* $d_i = \sum_{B_i} b.d^i_b$ *or* $d_i = \lfloor \sum_{B_i} b.d^i_b \rfloor (d^i_\sigma)$, *there exists a strong normal form* $snf(d) = \sum_{A'} a.d'_a + \sum_{I \neq \varnothing} \tau.d_i$ *such that* $d =_{\mathrm{E}} snf(d)$ *and* $d'_a \prec d$.

Proof. We proceed by direct equational manipulation of d. We will use the derived axiom $\tau.\lfloor x \rfloor(y) = x + \tau.\lfloor x \rfloor(y)$ (this follows from $\tau 1$ and the easily derivable equation $\lfloor x \rfloor(y) + \tau.z = x + \tau.z$):

$$d =_{\mathrm{E}} \sum_A a.d_a + \sum\nolimits_{I \neq \varnothing} \tau.d_i \qquad \text{by def}^n$$

$$=_{\mathrm{E}} \sum_A a.d_a + \sum_J \tau.\sum_{B_j} b.d^j_b + \sum_K \tau.\left\lfloor \sum_{B_k} b.d^k_b \right\rfloor (d^k_\sigma)$$

by def^n

$$=_{\mathrm{E}} \sum_A a.d_a + \sum_J \sum_{B_j} b.d^j_b + \sum_J \tau.\sum_{B_j} b.d^j_b$$

$$+ \sum_K \tau.\left\lfloor \sum_{B_k} b.d^k_b \right\rfloor (d^k_\sigma) \qquad \text{by } \tau 1$$

$$=_{\mathrm{E}} \sum_A a.d_a + \sum_J \sum_{B_j} b.d^j_b + \sum_K \sum_{B_k} b.d^k_b$$

$$+ \sum_J \tau.\sum_{B_j} b.d^j_b + \sum_K \tau.\left\lfloor \sum_{B_k} b.d^k_b \right\rfloor (d^k_\sigma) \qquad \text{from above}$$

$$=_{\mathrm{E}} \sum_A a.d_a + \sum_J \sum_{B_j} b.d^j_b + \sum_K \sum_{B_k} b.d^k_b + \sum_{I \neq \varnothing} \tau.d_i$$

by def^n

We now examine the term $\sum_A a.d_a + \Sigma_J \sum_{B_j} b.d^j_b + \sum_K \sum_{B_k} b.d^k_b$ in isolation: we aim to translate it into the form $\sum_{A'} a.d'_a$. Suppose $a.d_1$ and $a.d_2$ are both summands of $\sum_A a.\ d_a + \sum_J \sum_{B_j} b.d^j_b + \sum_K \sum_{B_k} b.d^k_b$. Then by $\tau 3$ we have $a.d_1 + a.d_2 a.(\tau.d_1 + \tau.d_2)$. As in Lemma 4.8, we can transform $a.d_1 + a.d_2$ into $a.nf(\tau.d_1 + \tau.d_2)$ with no increase in depth. We repeat this procedure until no such duplicated prefixed action appear in the sum. It is then straightforward to check that the transformation of d is both a strong normal form and satisfies the requirements of the lemma. ∎

It is also necessary to develop a partial normal form for infinite terms, i.e., those involving recursion. These are called head normal forms.

DEFINITION 4.12. A *head normal form* is a term of the following form.

1. $\lfloor \sum_A a.p_a \rfloor (p_\sigma)$ is in head normal form.
2. $\sum_A a.p_a + \sum_{I \neq \varnothing} \tau.p_i$ is in head normal form if each p_i is in a stable head normal form.

We use $hnf(p)$ to denote the head normal form of a term p.

THEOREM 4.13 (Head Normal Form Theorem). *For any term p such that $p \Downarrow$ there exists a head normal form $hnf(p)$ such that $p =_{\mathrm{Er}} hnf(p)$.*

Proof. The proof is similar to that of the Normal Form Theorem except that the induction used is on the length of the proof that $p \Downarrow$. ∎

In the next theorem, the heart of the completeness theorem, we use various simple facts about $\sqsubseteq^c$ which are summarised in the following lemma. The proofs are straightforward and are left to the reader.

LEMMA 4.14. 1. *(τ-preservation)* $p \ll q \Rightarrow \tau.p \ll^c \tau.q$

2. *(stability) for convergent p* $p \ll^c q \wedge p \xrightarrow{\tau} \Rightarrow q \xrightarrow{\tau}$
3. *(σ-property)*

$$\left.\begin{array}{l} p \ll^c q \\ p \xrightarrow{\sigma} p_\sigma \\ q \xrightarrow{\sigma} q_\sigma \end{array}\right\} \Rightarrow p_\sigma \ll^c q_\sigma.$$

THEOREM 4.15 (Partial Completeness). *For any finite process d and any process q*

$$d \sqsubseteq^c q \Rightarrow \vdash_{\mathrm{E}} d \leqslant q$$

Proof. We proceed by induction on the order $\prec$ over d and its subterms. For convenience we abbreviate $\vdash_{\mathrm{E}} d \leqslant q$ to $d \leqslant_{\mathrm{E}} q$ within the confines of this proof although essential use is made of the extra rules. We may assume d is in normal form. If $d \Uparrow$ then it is easy to prove by structural induction on d that $d =_{\mathrm{E}} \Omega$ and the result follows immediately. So we may further assume that $d \Downarrow$ and therefore $q \Downarrow$; in particular we may now assume q is in head normal form.

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

First we prove that $n_\sigma \leqslant_{\mathrm{E}} q_\sigma$. $d_\sigma \ll^c q_\sigma$ follows directly from the σ-property, Part 3 of Lemma 4.14, and so by induction we have $d_\sigma \leqslant_{\mathrm{E}} q_\sigma$.

Now we prove $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_B b.q_b$. Any barb $b\underline{v} \in S\,Barb(q)$ must be matched by one in $S\,Barb(d)$, so $B \subseteq A$. Any barb $B\sigma\underline{v} \in S\,Barb(q)$ must be matched by one in

$S\ Barb(d)$, so $A \subseteq B$, i.e., $A = B$. Also, for all $a \in A$, $d_a \ll q_a$. For consider $\underline{v} \in S\ Barb(q_a)$. Then $a\underline{v} \in S\ Barb(q)$ and so by $d \ll^c q$ we have $a\underline{u} \in S\ Barb(d)$ with $a\underline{u} \ll a\underline{v}$. Hence $\underline{u} \ll \underline{v}$ with $\underline{u} \in S\ Barb(d_a)$. By the τ-preservation property, Part 1 of Lemma 4.14, $d_a \ll q_a \Rightarrow \tau.d_a \ll^c \tau.q_a$, and so by induction for all $a \in A$ $\tau.d_a \leqslant_{\mathrm{E}} \tau.q_a$. Hence for all $a \in A$ $a.\tau.d_a \leqslant_{\mathrm{E}} a.\tau.q_a$ and so $\sum_A a.\tau.d_a \leqslant_{\mathrm{E}} \sum_A a.\tau.q_a$ which, by $\tau 3$, gives $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_A a.q_a$, i.e., $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_B b.q_b$.

Combining these results we have

$$d \equiv \left\lfloor \sum_A a.d_a \right\rfloor (d_\sigma) \leqslant_{\mathrm{E}} \left\lfloor \sum_B b.q_b \right\rfloor (q_\sigma) \equiv q.$$

- $d \equiv \sum_B b.d_b + \sum_I \tau.d_i$.

There are several sub-cases to consider.

— $I = \varnothing$, i.e., $d \equiv \sum_A a.d_a$.

Note that this includes the case when $A = \varnothing$, that is, $d \equiv nil$. We define the term $e = recx.\lfloor \sum_A a.d_a \rfloor (x)$ and prove that

$$\forall n. \forall p. d \ll^c p \Rightarrow e^n \leqslant_{\mathrm{E}} p.$$

We continue in this subproof by induction on n.

* $n = 0$. $e^0 = \Omega$ and so by the $\Omega - Rule$ we have $e^0 \leqslant_{\mathrm{E}} p$.

* $n = k + 1$. By d's stability we know that p has the stable head normal form $p =_{\mathrm{E}} \lfloor \sum_B b.p_b \rfloor (p_\sigma)$. First note that $d \ll^c p_\sigma$ follows directly by the σ-property and by induction on n we have $e^k \leqslant_{\mathrm{E}} p_\sigma$.

Now we prove that $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_B b.p_b$. Any barb $b\underline{v} \in S\ Barb(p)$ must be matched by one in $S\ Barb(d)$ so $B \subseteq A$. Any barb $B\sigma\underline{v} \in S\ Barb(p)$ must be matched by one in $S\ Barb(d)$, so $A \subseteq B$, i.e., $A = B$. Also, for all $a \in A$, $d_a \ll p_a$. For consider $\underline{v} \in S\ Barb(p_a)$. Then $a\underline{v} \in S\ Barb(p)$ and so by $d \ll^c p$ we have $a\underline{u} \in S\ Barb(d)$ with $a\underline{u} \ll a\underline{v}$. Hence $\underline{u} \ll \underline{v}$ with $\underline{u} \in S\ Barb(d_a)$. By τ-preservation $d_a \ll p_a \Rightarrow \tau.d_a \ll^c \tau.p_a$ and so by induction for all $a \in A$, $\tau.d_a \leqslant_{\mathrm{E}} \tau.p_a$. Hence for all $a \in A$, $a.\tau.d_a \leqslant_{\mathrm{E}} a.\tau.p_a$, and so $\sum_A a.\tau.d_a \leqslant_{\mathrm{E}} \sum_A a.\tau.p_a$ which, by $\tau 3$, gives $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_A a.p_a$, i.e., $\sum_A a.d_a \leqslant_{\mathrm{E}} \sum_B b.p_b$.

Combining these results we obtain $\lfloor \sum_A a.d_a \rfloor (e^k) \leqslant_{\mathrm{E}} \lfloor \sum_B b.p_b \rfloor (p_\sigma)$. But $e^{k+1} = \lfloor \sum_A a.d_a \rfloor (e^k)$ and so $e^{k+1} \leqslant_{\mathrm{E}} p$ as required.

Instantiating the p above to be the q of this theorem we have $\forall n.e^n \leqslant_{\mathrm{E}} q$ and by $\omega - Induction$ we get $e \leqslant_{\mathrm{E}} q$. By the $Stability - Rule$ $d \leqslant_{\mathrm{E}} e$ and so $d \leqslant_{\mathrm{E}} q$ as required.

— $I \neq \varnothing$.

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:

$$d \equiv \sum_A a.d_a + \sum_{I \neq \varnothing} \tau.d_i,$$

where

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

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

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

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 τ-preservation $\tau.d_c \ll^c \tau.q_c$. By induction we have $\tau.d_c \leqslant_{\mathrm{E}} \tau.q_c$ and therefore $c.d_c \leqslant_{\mathrm{E}} c.q_c$. This means that for each such c, $d \leqslant_{\mathrm{E}} d + c.q_c$ and, because of $\tau 4$, to complete the theorem it is sufficient to prove $d \leqslant_{\mathrm{E}} d + \tau.\lfloor \sum_{E_j} e.q_e^j \rfloor (q_\sigma^j)$ for each j in J. Let a typical such q^j be of the form $\lfloor \sum_E e.q_e \rfloor (q_\sigma)$. We actually show that $d' \leqslant_{\mathrm{E}} \tau.\lfloor \sum_{E_j} e.q_e^j \rfloor (q_\sigma^j)$, where d' is $\sum_E e.d_e + \sum_{I'} \tau.d_i$ with $I' = \{i \in I : B_i \subseteq E\}$. Since $d \ll^+ q$ it follows that I' is not empty. We use induction on its size.

* $|I'| = 1$. Exactly how we proceed depends on the form of d'; it has either the form $\sum_E e.d_e + \tau.\sum_B b.d_b^1$ or $\sum_E e.d_e + \tau.\lfloor \sum_B b.d_b^1 \rfloor (d_\sigma^1)$. We consider the latter case in detail as the former is dealt with in a similar manner to the case above when $d = \sum_B b.d_b$.

We first show that $d_\sigma^1 \ll q_\sigma$. To any barb $\underline{b} \in S\ Barb(q_\sigma)$ there corresponds a barb $E\sigma\underline{b} \in S\ Barb(q)$. This must be matched by a barb from d and the only candidates are those of the form $B\sigma\underline{b'}$ where $\underline{b'} \in S\ Barb(d_\sigma^1)$. Now by τ-preservation we have $\tau.d_\sigma^1 \ll^c \tau.q_\sigma$ and therefore by induction $\tau.d_\sigma^1 \leqslant_{\mathrm{E}} \tau.q_\sigma$.

It is also easy to establish that $d_e \ll q_e$ by considering the possible barbs of q_e and using the fact that d is a strong normal form. Again, using τ-preservation, we have $d_e \leqslant_{\mathrm{E}} q_e$ for each $e \in E$.

We now have the required ingredients to prove $d' \leqslant_{\mathrm{E}} \tau.\lfloor \sum_E e.q_e \rfloor (q_\sigma)$:

$$\begin{aligned} &\sum_E e.d_e + \tau.\left\lfloor \sum_B b.d_b^1 \right\rfloor (d_\sigma^1) \\ &\leqslant_{\mathrm{E}} \tau.\left\lfloor \sum_E e.d_e + \sum_B b.d_\sigma^1 \right\rfloor (d_\sigma^1) && \text{by } \sigma\tau 3 \\ &\leqslant_{\mathrm{E}} \tau.\left\lfloor \sum_E e.d_e \right\rfloor (d_\sigma^1) && \text{by } \tau 3, \tau 4 \text{ since } B \subseteq E \\ &=_{\mathrm{E}} \tau.\left\lfloor \sum_E e.d_e \right\rfloor (\tau.d_\sigma^1) && \text{by } \sigma\tau 2 \end{aligned}$$

$$\leqslant_{\mathrm{E}} \tau.\left\lfloor \sum_E e.q_e \right\rfloor (\tau.q_\sigma)$$

$$=_{\mathrm{E}} \tau.\left\lfloor \sum_E e.q_e \right\rfloor (q_\sigma) \qquad \text{by } \sigma\tau 2.$$

This ends the proof when $|I'|=1$.

* $|I'|>1$.

We suppose without loss of generality that $I'=\{1,2,3,...,k\}$. We define a new term d'' which is a term lying between d' and $\tau.q_j$,

$$d'' \equiv \sum_E e.d'_e + \sum_{2\leqslant i\leqslant k} \tau.d'_i,$$

where for $3\leqslant i\leqslant k$ we define $d'_i\equiv d_i$. The definition of d'_2 depends on the structure of d_1 and d_2 as does the rest of the proof.

· $d_1=\sum_{B_1} b.d^1_b,\ d_2=\sum_{B_2} b.d^2_b$

In this case we define

$$d'_2 \equiv \sum_{B_1\backslash B_2} b.d^1_b + \sum_{B_2\backslash B_1} b.d^2_b + \sum_{B_1\cap B_2} b.nf(\tau.b^1_b+\tau.b^2_b).$$

Then since $B_1\cup B_2\subseteq E$ we have $d''\ll^c \tau.q_j$ and by induction $d''\leqslant_{\mathrm{E}} \tau.q_j$ as required.

· $d_1=\sum_{B_1} b.d^1_b,\ d_2=\lfloor\sum_{B_2} b.d^2_b\rfloor(d^2_\sigma)$

In this case we define

$$d'_2 \equiv \left\lfloor \sum_{B_1\backslash B_2} b.d^1_b + \sum_{B_2\cap B_2} b.d^2_b + \sum_{B_1\cap B_2} b.nf(\tau.b^1_b+\tau.b^2_b)\right\rfloor \left(nf\left(\tau.\sum_{B_1} b.d^1_b+\tau.d^2_\sigma\right)\right).$$

Again $d''\ll^c \tau.q_j$ and by induction $d''\leqslant_{\mathrm{E}}\tau.q_j$. By $\sigma 2$ and $\sigma\tau 2$ $d'\leqslant_{\mathrm{E}} d''$ and so $d'\leqslant_{\mathrm{E}}\tau.q_j$ as required.

· $d_1=\lfloor\sum_{B_1} b.d^1_b\rfloor(d^1_\sigma),\ d_2=\lfloor\sum_{B_2} b.d^2_b\rfloor d^2_\sigma)$

In this case we define

$$d'_2 \equiv \left\lfloor \sum_{B_1\backslash B_2} b.d^1_b + \sum_{B_2\backslash B_1} b.d^2_b + \sum_{B_1\cap B_2} b.nf(\tau\cdot b^1_b+\tau.b^2_b)\right\rfloor (nf(\tau.d^1_\sigma+\tau.d^2_\sigma)).$$

Again $d''\ll^c\tau.q_j$ and by induction $d''\leqslant_{\mathrm{E}}\tau.q_j$. By $\sigma\tau 2$ $d''\leqslant_{\mathrm{E}} d''$ and so $d'\leqslant_{\mathrm{E}}\tau.q_j$ as required.

This completes the induction on $|I'|$ and hence we have shown that $d'\leqslant_{\mathrm{E}}\tau.q_j$.

We now need to show that $d'\leqslant_{\mathrm{E}} d''$. Fortunately this follows directly from $\tau 4$.

That ends the final case in our partial completeness proof. ∎

As an immediate corollary we have a completeness proof for arbitrary closed terms.

THEOREM 4.16 (Completeness). *For arbitrary closed terms p, q, $p \sqsubseteq^c q$ implies $\vdash_{\mathrm{E}} p\leqslant q$.*

Proof. Suppose $p\ll^+ q$. In order to establish that $\vdash_{\mathrm{E}} p\leqslant q$, using ω – Induction, it is sufficient to show that $\vdash_{\mathrm{E}} d\leqslant q$ for an arbitrary finite approximation d of p. But $p\ll^+ q$ implies that $d\ll^+ q$ and therefore $\vdash_{\mathrm{E}} d\leqslant q$ follows from the previous result. ∎

To finish this section let us now address the soundness of the system.

THEOREM 4.17 (Soundness). *For arbitrary closed terms p, q, $\vdash_{\mathrm{E}} p\leqslant q$ implies $p\sqsubseteq^c q$.*

Proof. We have already shown that the inequations are sound and there are only two nontrivial rules:

1. The Stability Rule. For any set of closed processes $\{p_i : i\in I\}$ it is easy to check that $\sum_I \alpha_i.p_i \sim recx.\lfloor\sum_I\alpha_i.p_i\rfloor(x)$, from which the soundness follows.

2. ω-Induction. The proof of soundness of this rule is similar in spirit to that in [He88]. It is sufficient to establish that for any experiment e, p *must* e implies d *must* e for some finite approximation d of p. In [He88] it was sufficient to prove this for finite experiments e and induction was used on the size of e. Here we cannot use this measure of induction, since it may be possible that $e\xrightarrow{\sigma} e$. Instead we use another measure which does not depend on the fact that e is finite.

Let us abbreviate the computation

$$e\mid p=e_0\mid p_0\mapsto e_1\mid p_1\mapsto\cdots e_k\mid p_k$$

to

$$e\mid p\mapsto^n e_k\mid p_k$$

if

(a) for every $i\geqslant 0$, e_i cannot report success

(b) in the derivation above the inferences

$$e\xrightarrow{a} e',\ p\xrightarrow{\bar a} p' \text{ implies } e\mid p\mapsto e'\mid p'$$

or

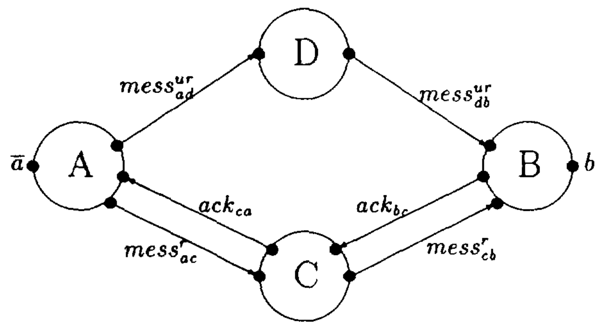
$$e\xrightarrow{\sigma} e',\ p\xrightarrow{\sigma} p' \text{ implies } e\mid p\mapsto e'\mid p'$$

are used n times.

One can show that if p *must* e then the set $\{n\mid e\mid p\mapsto^n e'\mid p'\}$ is finite. One can now mimic the corresponding proof in [He88], Lemma 4.5.6, but using induction on the maximal element of this set. ∎

5. EXAMPLE

We present in this section a description of a very simple "Security Costs Protocol." The Security Costs Protocol describes the transition of a message between two distributed ports. Transmission of a message across a secure medium is considered expensive while acknowledgements travel freely. The protocol initially sends the message across an unsecure medium only resending across the secure medium if an acknowledgement has not arrived before timeout.



Accept:

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

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:

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

$System \Leftarrow (A \mid B \mid C \mid D)\backslash S$ where $S = Sort(A) \cup Sort(B) \cup Sort(C) \cup Sort(D)\backslash\{a, b\}$.

The message is received by the protocol on port $\bar{a}$. 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 the port mes^{ur}_{db}, or it loses the message. Upon the possible receipt of the message from D, B will send and acknowledgment to A via the reliable medium along the 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 on to 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 send 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" while the second ack_{bc} represents "message delivered to environment, reset to accept a new message."

We can now prove equationally that

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

$$\begin{aligned}
System &= (A \mid B \mid C \mid D)\backslash S \qquad \text{by definition}\\
&= \bar{a}.(mess^{ur}_{ad}.(\overline{ack_{ca}}.\overline{ack_{ca}}.A\\
&\quad + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A) \mid B \mid C \mid D)\backslash S \qquad \text{by } d1\\
&= \bar{a}.\tau.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A) \mid B \mid C \mid\\
&\quad (\tau.D + \tau.mess^{ur}_{db}.D))\backslash S \qquad \text{by } d1\\
&= \bar{a}.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A) \mid B \mid C \mid\\
&\quad (\tau.D + \tau.mess^{ur}_{db}.D))\backslash S \qquad \text{by } d5\\
&= \bar{a}.(X + Y) \qquad \text{by } d4
\end{aligned}$$

where

$$\begin{aligned}
X &= \tau.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A) \mid B \mid C \mid D)\backslash S\\
Y &= \tau.((\overline{ack_{ca}}.\overline{ack_{ca}}.A\\
&\quad + \sigma.mess^{r}_{ca}.\overline{ack_{ca}}.A) \mid B \mid C \mid mess^{ur}_{db}.D)\backslash S.
\end{aligned}$$

Now

$$\begin{aligned}
X &= \tau.\sigma.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + mess^{r}_{ac}.\overline{ack_{ca}}.A) \mid B \mid C \mid D)\backslash S\\
&\quad \text{by } d2\\
&= \tau.\sigma.\tau.(\overline{ack_{ca}}.A \mid B \mid mess^{r}_{cb}.C \mid D)\backslash S \qquad \text{by } d1\\
&= \tau.\sigma.\tau.\tau.(\overline{ack_{ca}}.A \mid b.ack_{bc}.B \mid C \mid D)\backslash S \qquad \text{by } d1\\
&= \tau.\sigma.\tau.\tau.b.(\overline{ack_{ca}}.A \mid ack_{bc}.B \mid C \mid D)\backslash S \qquad \text{by } d1\\
&= \tau.\sigma.\tau.\tau.b.\tau.(\overline{ack_{ca}}.A \mid B \mid ack_{ca}.C \mid D)\backslash S \qquad \text{by } d1\\
&= \tau.\sigma.\tau.\tau.b.\tau.\tau.(A \mid B \mid C \mid D)\backslash S \qquad \text{by } d1\\
&= \tau.\sigma.b.System \qquad \text{by } d5, d6
\end{aligned}$$

And

$$\begin{aligned}
Y &= \tau.((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)\\
&\quad \mid B \mid C \mid mess^{ur}_{db}.D)\backslash S \qquad \text{by definition}\\
&= \tau.\tau((\overline{ack_{ca}}.\overline{ack_{ca}}.A + \sigma.mess^{r}_{ac}.\overline{ack_{ca}}.A)\\
&\quad \mid ack_{bc}.b.ack_{bc}.B \mid C \mid D)\backslash \qquad \text{by } d3
\end{aligned}$$

$$
\begin{aligned}
&= \tau.\tau.\tau.((\overline{ack}_{ca}.\overline{ack}_{ca}.A + \sigma.mess^{r}_{ac}.\overline{ack}_{ca}.A) \\
&\quad |b.ack_{bc}.B \mid ack_{ca}.C \mid D)\backslash S \qquad \text{by } d3 \\
&= \tau.((\overline{ack}_{ca}.\overline{ack}_{ca}.A + \sigma.mess^{r}_{ac}.\overline{ack}_{ca}.A) \\
&\quad |b.ack_{bc}.B \mid ack_{ca}.C \mid D)\backslash S \qquad \text{by } d5 \\
&= \tau.(U + V) \quad \text{by } d3
\end{aligned}
$$

where

$$
\begin{aligned}
U &= b.((\overline{ack}_{ca}.\overline{ack}_{ca}.A + \sigma.mess^{r}_{ac}.\overline{ack}_{ca}.A) \\
&\quad |ack_{bc}.B \mid ack_{ca}.C \mid D)\backslash S \\
V &= \tau.(\overline{ack}_{ca}.A \mid b.ack_{bc}.B \mid C \mid D)\backslash S.
\end{aligned}
$$

Again

$$
\begin{aligned}
U &= b.\tau.(\overline{ack}_{ca}.A \mid ack_{bc}.B \mid C \mid D)\backslash S && \text{by } d3 \\
&= b.\tau.\tau.(\overline{ack}_{ca}.A \mid B \mid ack_{ca}.C \mid D)\backslash S && \text{by } d1 \\
&= b.\tau.\tau.\tau.(A \mid B \mid C \mid D)\backslash S && \text{by } d1 \\
&= b.System && \text{by } d5
\end{aligned}
$$

and

$$
\begin{aligned}
V &= \tau.(\overline{ack}_{ca}.A \mid b.ack_{bc}.B \mid C \mid D)\backslash S && \text{by definition} \\
&= \tau.b.(\overline{ack}_{ca}.A \mid ack_{bc}.B \mid C \mid D)\backslash S && \text{by } d1 \\
&= \tau.b.\tau.(\overline{ack}_{ca}.A \mid B \mid ack_{ca}.C \mid D)\backslash S && \text{by } d1 \\
&= \tau.b.\tau.\tau.(A \mid B \mid C \mid D)\backslash S && \text{by } d1 \\
&= \tau.b.System && \text{by } a2.
\end{aligned}
$$

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

$$
\begin{aligned}
& w = \textstyle\sum_A a.w_a + \sigma.w_\sigma,\ x = \sum_B b.x_b,\ y = \sum_C c.y_c,\ z = \sum_D d.z_d \\
(w|x|y|z) \setminus E &= \sigma.(((\textstyle\sum_A a.w_a + w_\sigma)|x|y|z) \setminus E) \\
&\quad \text{if } A \cap \overline{B} = \emptyset \wedge A \cap \overline{C} = \emptyset \wedge A \cap \overline{D} = \emptyset \wedge \\
&\quad B \cap \overline{C} = \emptyset \wedge B \cap \overline{D} = \emptyset \wedge C \cap \overline{D} = \emptyset \\
&\quad \text{and } A \cup B \cup C \cup D \subseteq E && d2
\end{aligned}
$$

$$
\begin{aligned}
& w = \textstyle\sum_A a.w_a + \sigma.w_\sigma,\ x = \sum_B b.x_b,\ y = \sum_C c.y_c,\ z = \sum_D d.z_d \\
(w|x|y|z) \setminus E &= ext + int \\
&\text{where} \\
ext &= \textstyle\sum_{A\setminus E} a.((w_a|x|y|z) \setminus E) + \sum_{B\setminus E} b.((w|x_b|y|z) \setminus E) + \\
&\quad \textstyle\sum_{C\setminus E} c.((w|x|y_c|z) \setminus E) + \sum_{D\setminus E} d.((w|x|y|z_d) \setminus E) \\
int &= \textstyle\sum_{A\cap\overline{B}} \tau.((w_a|x_b|y|z) \setminus E) + \sum_{A\cap\overline{C}} \tau.((w_a|x|y_c|z) \setminus E) + \\
&\quad \textstyle\sum_{A\cap\overline{D}} \tau.((w_a|x|y|z_d) \setminus E) + \sum_{B\cap\overline{C}} \tau.((w|x_b|y_c|z) \setminus E) + \\
&\quad \textstyle\sum_{B\cap\overline{D}} \tau.((w|x_b|y|z_d) \setminus E) + \sum_{C\cap\overline{D}} \tau.((w|x|y_c|z_d) \setminus E) \\
&\quad \text{if } A \cap \overline{B} \neq \emptyset \vee A \cap \overline{C} \neq \emptyset \vee A \cap \overline{D} \neq \emptyset \vee \\
&\quad B \cap \overline{C} \neq \emptyset \vee B \cap \overline{D} \neq \emptyset \vee C \cap \overline{D} \neq \emptyset && d3
\end{aligned}
$$

$$
\begin{aligned}
(\tau.x + \tau.y)|z &= \tau.x|z + \tau.y|z && d4 \\
a\tau.x &= a.x && d5 \\
\tau.\sigma.\tau.x &= \tau.\sigma.x && d6
\end{aligned}
$$

FIG. 7. Derived equations.

So finally

$$
\begin{aligned}
System &= \bar{a}.(\tau.\sigma.b.System + \tau.(b.System + \tau.b.System)) \\
&\qquad \text{from above} \\
&= \bar{a}.(\tau.\sigma.b.System + \tau.b.System) \qquad \text{by } \tau 1,\ d5.
\end{aligned}
$$

Figure 7 shows the new equations used is this proof. All of these except $d4$ are derived equations in our proof system; their derivations are straightforward but tedious. Every closed instance of $d4$ can also be derived but the axiom itself cannot. Its use is inessential but we employ it to make the proof more readable. We leave the reader to check its soundness using the alternative characterisation.

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 referred to [Je91a], 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, [BB91] presents a real-time extension of ACP, [Re88] contains an extension of CSP called *Timed CSP*, while CCS is the starting point for [MT90], 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 [BB91] while the operational viewpoint, which underlies much of the research on CCS, is reflected in [MT90]. 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 [Sch95] 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 [BB91, 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. Nevertheless these languages have been very influential. They are very expressive, have sound semantic theories based on either forms of bisimulation equivalence

[Mil89] or *Refusals* [Hoa85], and have been seen to be useful in real-time applications.

The other major approach to representing time is to introduce special actions to represent the passage of time, which the current paper shares with [Gr89, MT90, NS94, Yi90, Yi91], although the basis for all those proposals may be found in [BC88]. All of the languages presented in these papers share many of the underlying informal assumptions of TPL outlined in the introduction. For example, they all continue to assume that actions are instantaneous and only the extension of ACP presented in [Gr89] does not incorporate *time determinism*; however, *maximal progress* is less popular as an assumption and *patience* is even rarer. Although each of these proposals uses a different syntax for its timed version of process algebra, it is of more interest to classify them according to the assumptions they impose on the special "time" actions.

In [Gr89], the simplest proposal, time is just like any other action except that it must synchronise across parallel bar. This fits in very neatly with the general synchronisation mechanism of ACP and an axiomatisation of weak bisimulation for finite terms in an extension of ACP with this timed action is given. In [MT90], a similar action is introduced into CCS but it assumes more of the characteristics of time; time determinism is assumed but they are uncommitted as to whether time is discrete or continuous. They give a complete axiomatisation of strong bisimulation for finite terms in a rather expressive language. The language, and operational semantics, in [NS94] is similar in spirit to TPL but is based on a different algebra ACP. In fact, it is from this language that the $\lfloor \_ \rfloor (\_)$ 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 [NS94, MT90] assumes 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 assuming time determinism, maximal progress, and patience. However, as with [Gr89, MT90, NS94], 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

$$a(t).P(t),$$

which represents a process which can perform the action a at time t and then act like the process $P(t)$; so the behaviour of processes can in some sense be parameterised on the time when actions are performed.

Thus the approach we have taken has much in common with that of [Gr89, MT90, NS94, Yi90, Yi91]. A major feature of this common approach is that the action representing time has special features which are incorporated into the operational semantics of the various languages using some form of prioritisation of the actions. Indeed, it is shown in [Jef92] that many timed languages which take this approach can be translated into untimed languages where actions have priorities associated with them. However, our semantic theory is based on testing and as far as we know the problem of developing a testing based semantic theory for timed processes has not been tackled before although, as we have previously mentioned, a construct similar to σ has been used in [Ph87, La89] to describe so-called "refusal" tests. We have deliberately chosen a rather simple notion of time and in this choice we were very influenced by the preliminary exploration in [Ste88] carried out as part of the FORMAP project.

But now that a firm basis has been laid for a testing based theory we hope to be able to extend it to languages with more complicated constructs. The extension to the constructs of [Yi91] which are parameterised on time should be straightforward but to handle processes which are not patient will require a reworking of the notion of barb. Finally, extending the theory of tests for a language where time is not discrete will be a major challenge.

Received April 24, 1991; final manuscript received May 10, 1993

REFERENCES

[Ab91] Abramsky, S. (1991), A domain equation for bisimulation, *Inform. and Comput.* **92**, 161–218.

[AH92] Arun-Kumar, S., and Hennessy, M. (1992), An efficiency preorder for processes, *Acta Informat.* **29**, 737–760.

[BB91] Baeten, J., and Bergstra, J. (1991), Real time process algebra, *Formal Aspects Comput. Sci.* **3**, 142–188.

[BW90a] Baeten, J., and Weijland, W. (1990), "Process Algebra," Cambridge Univ. Press, London/New York.

[BW90b] Baeten, J., and Weijland, W. (1990), "Applications of Process Algebra," Cambridge Univ. Press, London/New York.

[BC88] Berry, G., and Cosserat, L. (1988), "The ESTEREL Synchronous Programming Language and Its Mathematical Semantics," Technical Report 842, INRIA, Sophia-Antipolis.

[BW89] Burns, A., and Wellings, A. (1989), "Real-Time Systems and their Programming Languages," Addison-Wesley, Reading, MA.

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

[CH90] Cleaveland, R., and Henessey, M. (1990), Priorities in process algebras, *Inform. and Comput.* **87**, 58–77.

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

[DS89] Davies, J., and Schneider, S. (1991), "An Introduction to Timed CSP," Technical Report, PRG, Oxford.

[dNH84] De Nicola, R., and Hennessy, M. (1984), Testing equivalence for processes, *Theoret. Comput. Sci.* **34**, 83–133.

[GB86] Gert, R., and Boucher, A. (1986), A timed failures model for extended communication processes, *in* "Lecture Notes in Computer Science," Vol. 267, pp. 95–114, Springer-Verlag, Berlin/New York.

[vG90] van Glabbeek, R. (1990), The linear time-branching time spectrum, *in* "Proceedings, CONCUR90," pp. 278–297, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.

[Gr89] Groote, J. F. (1989), "Specification and Verification of Real Time Systems in ACP," Technical Report CS-R9015, CWI, Amsterdam. An extended abstract appeared *in* "Proceedings 10th International Symposium on Protocol Specification, Testing and Verification, Ottawa, 1990" (L. Logrippa, R. L. Probert, and H. Urel, Eds.), pp. 261–274.

[He81] Hennessy, M. (1981), A term model for synchronous processes, *Inform. and Control* **51**, 58–75.

[He83] Hennessy, M. (1983), Synchronous and asynchronous experiments on processes, *Inform. and Control* **59**, 36–83.

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

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

[Hoa85] Hoare, C. A. R. (1985), "Communicating Sequential Processes," Prentice-Hall, Englewood Cliffs, NJ.

[HdR89] Hooman, J. J. M., and deRoever, W. P. (1989), Design and verification in real-time distributed computing: An introduction to compositional methods, *in* "Proceedings of the Ninth International Conference on Protocol Specification, Testing and Verification," North-Holland, Amsterdam.

[Je91a] Jeffrey, A. (1991), "Discrete Timed CSP," Technical Report PMG 79, Chalmers University of Technology, Sweden.

[Je91b] Jeffrey, A. (1991), "Timed Process Algebra ≠ Time × Process Algebra," Technical Report PMG 78, Chalmers University of Technology, Sweden.

[Je91c] Jeffrey, A., A linear time process algebra, *in* "CAV91."

[Jef92] Jeffrey, A. (1992), Translating timed process algebra into untimed process algebra, *in* "Proceedings, Formal Techniques in Real-Time and Fault-Tolerant Systems," Lecture Notes in Computer Science, Vol. 571, Springer-Verlag, Berlin/New York.

[Jo89] Joseph, M. (1989), Time and real-time in programs, *in* "FST & TCS 9, Bangolore," Lecture Notes in Computer Science, Vol. 405, Springer-Verlag, Berlin/New York.

[Kl91] Klusener, A. S. (1991), Completeness in real time process algebra, *in* "CONCUR 91," pp. 96–110, Lecture Notes in Computer Science, Vol. 527, Springer-Verlag, Berlin/New York.

[Kn75] Knuth, D. (1975), "Fundamental Algorithms," Addison-Wesley, Reading, MA.

[La89] Langerak, R. (1989), A testing theory for LOTOS using deadlock detection, *in* "Proceedings of the Ninth International Conference on Protocol Specification, Testing and Verification," North-Holland, Amsterdam.

[Mil83] Milner, R. (1983), Calculi for synchrony and asynchrony, *Theoret. Comput. Sci.* 267–310.

[Mil89] Milner, R. (1989), "Calculus for Communication and Concurrency," Prentice-Hall, London.

[Mil90] Milner, R. (1990), Operational and algebraic semantics of concurrent processes, *in* "Handbook of Theoretical Computer Science," North-Holland, Amsterdam.

[MT90] Moller, F., and Tofts, C. (1990), A Temporal Calculus of Communicating Systems, *in* "Proceedings of CONCUR 90," pp. 401–415, Lecture Notes in Computer Science, Vol. 458, Springer-Verlag, Berlin/New York.

[MT91a] Moller, F., and Tofts, (1991), "Relating Processes With Respect To Speed," Technical Report, LFCS, Edinburgh.

[MT91b] Moller, F., and Tofts, C. (1991), Relating processes with respect to speed, *in* "CONCUR91," pp. 424–438, Lecture Notes in Computer Science, Vol. 527, Springer-Verlag, Berlin/New York.

[Mur90] Murphy, D. V. J. (1990), "Time, Casuality, and Concurrency," Ph.D. Thesis, University of Glasgow, CSC 90/R32, 1990.

[NS94] Nicollin, X., and Sifakis, J. (1994), The algebra of timed processes, **ATP**: Theory and application, *Inform. and Comput.* **114**, 131–178.

[Ph87] Phillips, I. (1987), Refusal Testing, *Theoret. Comput. Sci.* **50**, 241–284.

[Ph88] Phillips, I. (1988), "CCS With Broadcast Stability," Draft Technical Report, Imperial College.

[Pn85] Pneuli, A. (1985), Linear and branching structures in the semantics and logics of reactive systems, pp. 15–32, Lecture Notes in Computer Science, Vol. 194, Springer-Verlag, Berlin/New York.

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

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

[Reg91] Regan, T. (1991), "Process Algebras for Real-Time Systems," Ph.D. Thesis, University of Sussex.

[RR86] Reed, G. M., and Roscoe, A. (1986), A timed model for communicating sequential processes, pp. 314–323, Lecture Notes in Computer Science, Vol. 226, Springer-Verlag, Berlin/New York.

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

[Sch86] Schmidt, D. A., (1986), "Denotational Semantics," Allyn & Bacon, Needham Heights, MA.

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

[Sch95] Schneider, S. A. (1995), An operational semantics for timed CSP, *Inform. and Comput.* **116**, 193–213.

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

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

[Yi90] Yi, W. (1990), Real-time behavior of asynchronous agents, *in* "Lecture Notes in Computer Science," Vol. 458, pp. 502–520, Springer-Verlag, Berlin/New York.

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

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