An Operational Semantics for Timed CSP[1]

Steve Schneider

Programming Research Group, Oxford University Computing Laboratory,
11 Keble Road, Oxford OX1 3QD, UK

Abstract

An operational semantics is defined for the language of timed CSP, in terms of two relations: an evolution relation, which describes when a process becomes another simply by allowing time to pass; and a timed transition relation, which describes when a process may become another by performing an action at a particular time. It is shown how the timed behaviours used as the basis for the denotational models of the language may be extracted from the operational semantics. Finally, the failures model for timed CSP is shown to be equivalent to may-testing, and thus to trace congruence.

1 Introduction

An operational semantics for a computer programming language defines the meaning of programs written in that language in terms of how a machine is intended to execute them step by step. It therefore offers a direct intuition of how program constructs are intended to behave, in contrast with denotational approaches, which often abstract away from such considerations, and with algebraic approaches, where operators are defined in terms of the algebraic laws they satisfy. In providing an operational semantics for a language which is already endowed with a denotational semantics, our intention is to provide an operational intuition for the language constructs, and hence gain some insight into the design decisions that were incorporated into the denotational definitions; in this way we may highlight those aspects of behaviour which derive from our philosophy of how programs should be executed, and expose those design decisions that have viable alternatives.

Timed Communicating Sequential Processes is an extension of the CSP language of (Hoare, 1985) to include timing constructs. It first appeared in (Reed and Roscoe, 1986), and a variety of denotational semantic models have since been provided for the language (Reed and Roscoe, 1987), (Reed 1988); the report (Davies and Schneider, 1989) provides an introduction to these models. More recently the language and models have been refined to take a more abstract view of time. The alterations are chronicled in

[1]to appear in *Information and Computation*

(Davies and Schneider, 1992); we will use the timed model discussed there for the purposes of this paper. All of these approaches define the meaning of a CSP program to be a set of possible behaviours. The nature of a behaviour differs between models, the simplest kind being simply sequences of events (untimed traces). Behaviours in the more sophisticated timed models contain times at which events are performed and refused.

In order to verify that these denotational models are consistent with some intuition of how timed CSP programs should be executed, we present an operational semantics for timed CSP. We consider the execution of a timed CSP program as a sequence of timed transitions: timed communication events (drawn from the alphabet Σ) and timed internal events (τ). Inference rules are presented which define the transitions which a process may perform, which for composite programs are given in terms of the possible transitions of the constituents; whether the constituents may evolve over time without performing any transitions, given in terms of an *evolves* relation; and what the components may perform initially, given by a function *init*.

Full abstraction will be established for the timed failures model $\mathcal{M}_{TF}$, by extracting timed traces and timed refusals from the transition sequences of a program, and establishing that the pertinent information agrees with the semantics given directly by the denotational semantic function. This result is used to show that equivalence under may-testing is the same as equality in the timed failures model; it follows that the timed failures model is the weakest model that implies trace congruence (the largest congruence smaller than trace equivalence). This illustrates the fact that timed traces alone are not sufficient to provide an adequate model for timed CSP, and that a model containing refusal information is required.

2 Notation

The set Σ is the set of visible events. The event τ is a special internal event; $\tau \notin \Sigma$. The set Σ^+ denotes $\Sigma \cup \{\tau\}$. Variables a, b, c are taken to range over Σ, and μ ranges over Σ^+. The variables t and u range over $\mathbf{R}^+$, the set of non-negative real numbers. Variable s ranges over $(\mathbf{R}^+ \times \mathbf{\Sigma})^*$, the finite sequences of timed visible events; w ranges over $(\mathbf{R}^+ \times \mathbf{\Sigma}^+)^*$, the finite sequences of timed visible and internal events. We use $\aleph \subseteq \mathbf{R}^+ \times \mathbf{\Sigma}$ to represent a timed refusal, a set of timed visible events.

We use the following operations on sequences of (visible and internal) events: $\#w$ is the length of the sequence w; $w_1 \frown w_2$ denotes the concatenation of w_1 and w_2. We define the beginning and end of a sequence as follows: $begin(\langle(t,\mu)\rangle \frown w) = t$, $end(w \frown \langle(t,\mu)\rangle) = t$, and for convenience $begin(\langle\rangle) = \infty$ and $end(\langle\rangle) = 0$. The notation $w_1 \preceq w_2$ means that w_1 is a subsequence of w_2. The following projections on sequences are defined by list comprehension:

$$\begin{aligned} w \lhd t &= \langle (u,a) \mid (u,a) \leftarrow w, u \leq t \rangle \\ w \upharpoonright t &= \langle (u,a) \mid (u,a) \leftarrow w, u < t \rangle \end{aligned}$$

$$\begin{aligned}
w \rhd t &= \langle (u,a) \mid (u,a) \leftarrow w, u \geq t \rangle \\
w \uparrow t &= \langle (u,a) \mid (u,a) \leftarrow w, u = t \rangle \\
w \upharpoonright A &= \langle (u,a) \mid (u,a) \leftarrow w, a \in A \rangle \\
w \setminus A &= \langle (u,a) \mid (u,a) \leftarrow w, a \notin A \rangle \\
w - t &= \langle (u-t,a) \mid (u,a) \leftarrow w, u \geq t \rangle \\
w + t &= \langle (u+t,a) \mid (u,a) \leftarrow w \rangle \\
\sigma(w) &= \{ a \mid w \upharpoonright \{a\} \neq \langle\rangle \}
\end{aligned}$$

We also define a number of projections on refusal sets:

$$\begin{aligned}
\aleph \upharpoonleft t &= \{ (u,a) \mid (u,a) \in \aleph, u < t \} \\
\aleph \rhd t &= \{ (u,a) \mid (u,a) \in \aleph, u \geq t \} \\
\aleph \upharpoonright A &= \{ (u,a) \mid (u,a) \in \aleph, a \in A \} \\
\aleph - t &= \{ (u-t,a) \mid (u,a) \in \aleph, u \geq t \} \\
\sigma(\aleph) &= \{ a \mid (u,a) \in \aleph \} \\
end(\aleph) &= sup\{ u \mid (u,a) \in \aleph \}
\end{aligned}$$

We will use $(s, \aleph) - t$ as an abbreviation for $(s - t, \aleph - t)$.

3 Transition Systems

We begin by defining the kind of transition system we will use for modelling executions of programs. A timed transition system is a triple $\langle NODES, A_\Sigma, \longrightarrow \rangle$, where

- $NODES$ is a set of nodes, which we will use to represent programs.
- A_Σ is the set of timed events; for our purposes it is the set $\mathbf{R}^+ \times \mathbf{\Sigma}^+$.
- $\rightarrow$ is a ternary relation, a subset of $NODES \times A_\Sigma \times NODES$. Intuitively, we interpret $N_1 \xrightarrow{(t,\mu)} N_2$ as the information that it is possible for N_1 to evolve for length of time t, perform μ, and become N_2. In a computing context, we may consider it to mean that the execution of the program represented by N_1 may begin by waiting for length of time t, performing event μ, and then becoming the program represented by N_2; the execution continues from N_2.

We make the following abbreviations:

$$\begin{aligned}
N \xrightarrow{(t,\mu)} \quad &\widehat{=} \quad \exists\, N' : NODES \bullet N \xrightarrow{(t,\mu)} N' \\
N \overset{\mu}{\rightarrow} \quad &\widehat{=} \quad \exists\, t : \mathbf{R}, \mathbf{N'} : \mathbf{NODES} \bullet \mathbf{N} \xrightarrow{(\mathbf{t},\mu)} \mathbf{N'} \\
N \overset{(t,\mu)}{\not\longrightarrow} \quad &\widehat{=} \quad \neg(N \xrightarrow{(t,\mu)}\ ) \\
N \overset{\mu}{\not\rightarrow} \quad &\widehat{=} \quad \neg(N \overset{\mu}{\rightarrow}\ )
\end{aligned}$$

We extend the transition notation to allow finite sequences of (visible and internal) events w. If $w \in (A_\Sigma)^*$, $w = \langle (t_0, \mu_0) \ldots (t_{n-1}, \mu_{n-1}) \rangle$, then

$$\begin{aligned} N_0 \xrightarrow{w} N_n \quad &\widehat{=} \quad \exists\, N_1 \ldots N_{n-1} : NODES \bullet \forall\, i < n \bullet N_i \xrightarrow{(t_i, \mu_i)} N_{i+1} \\ N \xrightarrow{w} \quad &\widehat{=} \quad \exists\, N' : NODES \bullet N \xrightarrow{w} N' \end{aligned}$$

In order to define the transition relation, we will need first to define an evolution relation $\rightsquigarrow$, which is also a ternary relation, a subset of $NODES \times \mathbf{R}^+ \times \mathbf{NODES}$. We interpret $N_1 \overset{t}{\rightsquigarrow} N_2$ to mean that N_1 may evolve to N_2 in t units of time.

4 Timed CSP

Timed CSP *terms* are defined by the following Backus-Naur form:

$$\begin{aligned} P \quad ::= \quad & STOP \mid SKIP \mid WAIT\ t \mid a \longrightarrow P \mid P\,;P \\ & \mid P \overset{t}{\triangleright} P \mid P \,\Box\, P \mid P \sqcap P \mid \textstyle\bigsqcap_{i \in I} P_i \mid a : A \longrightarrow P_a \\ & \mid P\ {}_A\|_A\ P \mid P \mathrel{|||} P \\ & \mid P \setminus A \mid f(P) \mid f^{-1}(P) \mid X \mid \mu\, X \circ P \end{aligned}$$

We have that a ranges over Σ, A ranges over $\mathbf{P}\,\Sigma$, the powerset of Σ, f is a function from Σ to Σ, X ranges over $\mathcal{V}$, a set of process variables and I is a subset of $\mathcal{I}$, a set of indexes. The P_i and the P_a are sets of terms indexed by I and A respectively. For technical reasons (discussed in (Roscoe 1988)) to ensure that this BNF defines a minimal set of terms closed under all of the operations we require that the cardinality of set I must always be less than some regular cardinal κ, and that $\mathcal{I}$ has cardinality at least κ; but as there are arbitrarily large regular cardinals, this does not amount to a practical restriction. The variable t ranges over the reals; we will also allow arithmetic expressions, and consider syntactic equivalence to be modulo equal arithmetic expressions, identifying for example $WAIT\ 5$ and $WAIT\ (7-2)$.

We use the following rules to define when a timed CSP term P is t-guarded for variable X:

- For any X and t:
  1. $STOP$, $SKIP$ and $WAIT\ u$ are all t-guarded for X
  2. X is 0-guarded for X
  3. $Y \neq X$ is t-guarded for X
  4. $\mu\, X \circ P$ is t-guarded for X
- If P is t-guarded for X:

1. $a \longrightarrow P$, $\mu\, Y \circ P$, $SKIP\, ; P$, $P \setminus A$, $f(P)$, and $f^{-1}(P)$ are all t-guarded for X
2. $WAIT\ u\, ; P$ is $t + u$-guarded for X
3. P is u-guarded for X, for any $u \leq t$

- If P is t-guarded for X and Q is u-guarded for X:

1. $P \,\Box\, Q$, $P \sqcap Q$, $P\, ; Q$, $P \parallel\!\mid Q$, $P \;{}_A\|_B\; Q$ are all $\min\{t, u\}$-guarded for X
2. $P \overset{t_0}{\rhd} Q$ is $\min\{t, u + t_0\}$-guarded for X

- If for each $i \in I$ we have P_i is t-guarded for X, then so is $\bigsqcap_{i \in I} P_i$
- If for each $a \in A$ we have P_a is t-guarded for X, then so is $a : A \longrightarrow P_a$

We use $TCSP_0$ to denote the set of terms for which every subterm of the form $\mu\, X \circ P$ has some $t > 0$ for which P is t-guarded for X. This ensures that all recursions are guarded by some non-zero delay. Timed CSP *programs* are those members of $TCSP_0$ with no free process variables; these are denoted by $TCSP$.

5 Evolution and Transitions

Consider the nature of an execution of a timed process. In any given state P, it may withhold the performance of events, allowing time to pass; in this case we say it is *evolving*, and it may evolve in time t to a state P'. We will use the relation $P \overset{t}{\leadsto} P'$ to describe this possibility.

It is also possible that the process in some state will be obliged to perform a particular event, being refused permission to withold it; it is therefore unable simply to allow time to pass while doing nothing. Internal actions (which we will denote by τ) fall into this category, since no process is permitted to defer their performance. A process cannot therefore evolve beyond the time it is committed to an internal event. We will write $P \xrightarrow{(t,\tau)} P'$ to denote the possibility that P will perform an internal event at time t, becoming P'. We prove later (Corollary 5.8) that any process has at most one time at which it can perform an internal event.

In addition to performing internal events, the process may also wish to communicate with its environment, by performing an external event at a particular time. We write $P \xrightarrow{(t,a)} P'$ to denote that P can perform event $a \in \Sigma$ at time t and enter state P'. However, since it requires the cooperation of its environment in order to do this, it remains a possibility that the event will not be performed, since the environment may not be willing. It must therefore be possible for the process to evolve past communication events, since it cannot guarantee to perform them; it can at most guarantee to offer them.

In the operational semantics below, it turns out that the set of external events in which a process is prepared to engage does not change over the course of evolution, though of course it may alter when the state changes via the performance of an internal or external action. Hence the set of communications a process is prepared to engage in up to the next τ action is exactly the set it is prepared for at time *0*; that set will remain available until either one of its members or an internal event is performed. We will need to identify this set in order to define the operational semantics of the hiding and sequential composition operators. The function *init* identifies this set from the syntax of its argument process; we prove later (Theorem 5.5) that this concurs with the set given directly by the operational semantics.

Both the evolution relation $\rightsquigarrow$ and the transition relation $\rightarrow$ are defined in the standard way (Plotkin 1981) to be the smallest relations closed under all the monotonic inference rules given below. Such rules are of the form

$$\frac{\begin{array}{l} \textit{antecedent} \\ \ldots \\ \textit{antecedent} \end{array}}{\textit{conclusion}} \quad [\ \textit{side condition}\ ]$$

We will use the convention that antecedents will be assertions about the evolution and transition relations (on the components of a program), and the side condition will be some assertion not concerned with evolutions or transitions.

Operational Semantics Definitions

We give a set of operational rules for TCSP programs. We will use the standard notation $P[Q/X]$ to denote the term P with the term Q substituted for every free occurrence of X in P.

Deadlock

The deadlocked process can initially perform nothing.

$$\textit{init}(STOP) \quad = \quad \{\}$$

It permits any amount of time to pass, remaining deadlocked.

$$\frac{}{STOP \overset{t}{\rightsquigarrow} STOP}$$

It can perform no transitions.

Successful Termination

The process *SKIP* is able to terminate immediately. It does so by communicating the special termination event $\surd$ to its environment. It is therefore initially prepared to perform

$\surd$.

$$\mathit{init}(\mathit{SKIP}) \quad = \quad \{\surd\}$$

It is prepared to wait for any length of time, without changing its state.

$$\frac{}{\mathit{SKIP} \overset{t}{\rightsquigarrow} \mathit{SKIP}}$$

It is prepared to perform $\surd$ at any time; having terminated, it will do nothing further:

$$\frac{}{\mathit{SKIP} \xrightarrow{(t,\surd)} \mathit{STOP}}$$

Delay

The delay process $\mathit{WAIT}\ u$ will wait for u time units, after which it will change state (via an internal event) to become the terminating process SKIP. It is initially able to perform nothing.

$$\mathit{init}(\mathit{WAIT}\ u) \quad = \quad \{\}$$

It may evolve to a WAIT process with a smaller argument during the course of its delay. However, it may not evolve beyond the end of its delay.

$$\frac{}{\mathit{WAIT}\ u \overset{t}{\rightsquigarrow} \mathit{WAIT}\ (u-t)} \quad [\ t \le u\ ]$$

At the end of its delay it performs an internal action to permit termination.

$$\frac{}{\mathit{WAIT}\ u \xrightarrow{(u,\tau)} \mathit{SKIP}}$$

Event Prefixing

The process $a \longrightarrow P$ is initially prepared to engage in event $a \in \Sigma$.

$$\mathit{init}(a \longrightarrow P) \quad = \quad \{a\}$$

It is prepared to wait for its environment for any length of time, without changing its state.

$$\frac{}{a \longrightarrow P \overset{t}{\rightsquigarrow} a \longrightarrow P}$$

When it performs its initial event, it will begin to behave as P.

$$\frac{}{a \longrightarrow P \xrightarrow{(t,a)} P}$$

This operator differs from the prefix operator described in (Reed and Roscoe, 1986), where there was a non-zero delay of δ following the performance of the first event. This is now written as $a \longrightarrow \mathit{WAIT}\ \delta\ ;\ P$.

Sequential Composition

The sequential composition of two programs is executed by running its left hand argument until it terminates, which is modelled by the occurrence of an internal $\surd$ event. At this point control is passed to its right hand argument. The composite process $P \; ; \; Q$ may initially perform any external event that process P may perform with the possible exception of the $\surd$ event, which P is still permitted to perform but which becomes an internal event.

$$init(P \; ; \; Q) \quad = \quad init(P) \setminus \{\surd\}$$

If P is unable to terminate, and it can evolve through time t, then so can $P;Q$. Otherwise, it cannot evolve beyond time 0.

$$\frac{P \overset{t}{\rightsquigarrow} P'}{P \; ; \; Q \overset{t}{\rightsquigarrow} P' \; ; \; Q} \quad [\; \surd \notin init(P) \vee t = 0 \;]$$

If P is unable to terminate, and it can perform a timed event, then so can $P \; ; \; Q$.

$$\frac{P \xrightarrow{(t,\mu)} P'}{P \; ; \; Q \xrightarrow{(t,\mu)} P' \; ; \; Q} \quad [\; \surd \notin init(P) \;]$$

If P is able to perform a non termination event at time 0, then $P \; ; \; Q$ can also perform that event at time 0, even if P can also terminate. Further, if P can perform $\surd$ at time 0, then the process may perform an internal event and become Q.

$$\frac{P \xrightarrow{(0,\mu)} P'}{P \; ; \; Q \xrightarrow{(0,\mu)} P' \; ; \; Q} \quad [\; \mu \neq \surd \;] \qquad \frac{P \xrightarrow{(0,\surd)} P'}{P \; ; \; Q \xrightarrow{(0,\tau)} Q}$$

Choice

There are two forms of choice in CSP, nondeterministic or internal, and deterministic or external. It should be noted that the CSP forms of choice differ from the approach taken in CCS (Moller and Tofts 1990), (Wang Yi, 1991), in which an argument performing an internal τ event may thereby resolve the choice. Here, in the case of external choices, we have that internal events do not resolve the choice; and in the case of internal choices, although we will mark the resolution of the choice with an internal action, this resolution is completely independent of the capabilities of the component processes.

The timeout operator is a form of external choice. The process $P \overset{u}{\triangleright} Q$ initially executes P, but if no synchronisation has been performed by time u, then a timeout occurs, and control is passed to Q. Initially it may perform the same events as P.

$$init(P \overset{u}{\triangleright} Q) \quad = \quad init(P)$$

It may evolve as P evolves, but not beyond time u.

$$\frac{P \overset{t}{\rightsquigarrow} P'}{P \overset{u}{\triangleright} Q \overset{t}{\rightsquigarrow} P' \overset{u-t}{\triangleright} Q} \quad [\, t \leq u \,]$$

It may perform any events that P may perform before time u; external events resolve the choice in favour of P, internal ones do not.

$$\frac{P \xrightarrow{(t,a)} P'}{P \overset{u}{\triangleright} Q \xrightarrow{(t,a)} P'} \quad [\, t \leq u \,] \qquad \frac{P \xrightarrow{(t,\tau)} P'}{P \overset{u}{\triangleright} Q \xrightarrow{(t,\tau)} P' \overset{u-t}{\triangleright} Q} \quad [\, t \leq u \,]$$

Furthermore, the timeout may occur at time u.

$$\frac{P \overset{u}{\rightsquigarrow} P'}{P \overset{u}{\triangleright} Q \xrightarrow{(u,\tau)} Q}$$

An external choice of the form $P \Box Q$ is resolved by its first external communication event, in favour of the argument process that performs the event. The choice may initially engage in events that either of its argument processes may perform.

$$\mathit{init}(P \Box Q) \quad = \quad \mathit{init}(P) \cup \mathit{init}(Q)$$

If both components of the choice may evolve for a particular length of time, then so may the choice.

$$\frac{P \overset{t}{\rightsquigarrow} P' \qquad Q \overset{t}{\rightsquigarrow} Q'}{P \Box Q \overset{t}{\rightsquigarrow} P' \Box Q'}$$

Internal events do not resolve deterministic choice.

$$\frac{P \xrightarrow{(t,\tau)} P' \qquad Q \overset{t}{\rightsquigarrow} Q'}{P \Box Q \xrightarrow{(t,\tau)} P' \Box Q'} \qquad \frac{P \overset{t}{\rightsquigarrow} P' \qquad Q \xrightarrow{(t,\tau)} Q'}{P \Box Q \xrightarrow{(t,\tau)} P' \Box Q'}$$

External events resolve the choice.

$$\frac{P \xrightarrow{(t,a)} P' \qquad Q \overset{t}{\rightsquigarrow} Q'}{P \Box Q \xrightarrow{(t,a)} P'} \qquad \frac{P \overset{t}{\rightsquigarrow} P' \qquad Q \xrightarrow{(t,a)} Q'}{P \Box Q \xrightarrow{(t,a)} Q'}$$

A nondeterministic choice is made internally, and the environment of the process has no control over this choice. Operationally, we may think of it as a choice that is resolved internally at time 0; no external event may be performed before the choice is made. The binary form is as follows:

$$init(P \sqcap Q) \quad = \quad \{\}$$

No evolution is possible before the choice is made.

$$\frac{}{P \sqcap Q \overset{0}{\rightsquigarrow} P \sqcap Q}$$

The choice is made at time 0, and either of its arguments may be chosen.

$$\frac{}{P \sqcap Q \xrightarrow{(0,\tau)} P} \qquad \frac{}{P \sqcap Q \xrightarrow{(0,\tau)} Q}$$

General nondeterministic choice, where any process from a set may be chosen, acts in much the same way:

$$init(\textstyle\bigsqcap_{i \in I} P_i) \quad = \quad \{\}$$

No evolution is possible before the choice is made.

$$\frac{}{\bigsqcap_{i \in I} P_i \overset{0}{\rightsquigarrow} \bigsqcap_{i \in I} P_i}$$

The choice is made at time 0, and any of its arguments may be chosen.

$$\frac{}{\bigsqcap_{i \in I} P_i \xrightarrow{(0,\tau)} P_i} \quad [\ i \in I\ ]$$

There are technical restrictions on when this construct has a denotational semantics (see Appendix B), but there are no operational difficulties with it.

There is also a special form of infinite deterministic choice, which we term prefix choice. A set of events is offered by the process, and the choice is resolved when the first event occurs. Its set of initial events is the set being offered:

$$init(a : A \longrightarrow P_a) \quad = \quad A$$

It is able to evolve for any length of time.

$$\frac{}{a : A \longrightarrow P_a \overset{t}{\rightsquigarrow} a : A \longrightarrow P_a}$$

It may perform its first event b at any time; its subsequent behaviour is that of P_b.

$$\frac{}{a : A \longrightarrow P_a \xrightarrow{(t,b)} P_b} \quad [\ b \in A\ ]$$

For the same reasons as the general non-deterministic choice, there are technical restrictions on when this construct has a well-defined denotational semantics.

Parallel Combination

We consider here only the general parameterised parallel operator ${}_A\|_B$, where both A and B are sets of external events. Note that this is not the same as Hoare's original $\|$ operator presented in (Hoare, 1985), which assumes that the argument processes have alphabets. Here, we make explicit the interface through which each process is intended to interact, by supplying each interface explicitly as an argument A or B to the parallel operator. The general parallel operator permits its left hand argument to perform external events only from the set A, and its right hand argument only external events from the set B; they must cooperate on events in the intersection of those two sets, but not on other events. The events that $P\ {}_A\|_B\ Q$ may initially perform will be those events in $A \cap B$ that both P and Q may perform, together with those events in $A \setminus B$ that P may perform, and those in $B \setminus A$ that Q may perform.

$$\begin{aligned} \mathit{init}(P\ {}_A\|_B\ Q) &= (\mathit{init}(P) \cap A \setminus B) \cup (\mathit{init}(Q) \cap B \setminus A) \\ &\quad \cup (\mathit{init}(P) \cap \mathit{init}(Q) \cap A \cap B) \end{aligned}$$

If the component processes may both evolve, then so may their parallel composition.

$$\frac{P \overset{t}{\leadsto} P' \qquad Q \overset{t}{\leadsto} Q'}{P\ {}_A\|_B\ Q \overset{t}{\leadsto} P'\ {}_A\|_B\ Q'}$$

Components of a parallel composition may progress separately on internal events and disjoint events.

$$\frac{P \xrightarrow{(t,\mu)} P' \qquad Q \overset{t}{\leadsto} Q'}{P\ {}_A\|_B\ Q \xrightarrow{(t,\mu)} P'\ {}_A\|_B\ Q'} \quad [\ \mu \in A \setminus B \cup \{\tau\}\ ]$$

$$\frac{P \overset{t}{\leadsto} P' \qquad Q \xrightarrow{(t,\mu)} Q'}{P\ {}_A\|_B\ Q \xrightarrow{(t,\mu)} P'\ {}_A\|_B\ Q'} \quad [\ \mu \in B \setminus A \cup \{\tau\}\ ]$$

If both components can engage in a common timed event, then so can the parallel combination.

$$\frac{P \xrightarrow{(t,a)} P' \qquad Q \xrightarrow{(t,a)} Q'}{P\ {}_A\|_B\ Q \xrightarrow{(t,a)} P'\ {}_A\|_B\ Q'} \quad [\ a \in A \cap B\ ]$$

The other parallel operator is asynchronous. Processes combined using the $|||$ constructor run simultaneously and completely independently. The resulting process can perform any events that either of its components can perform.

$$\mathit{init}(P \ |||\ Q) \ = \ \mathit{init}(P) \cup \mathit{init}(Q)$$

It may evolve when both of its component processes may do so.

$$\frac{P \overset{t}{\rightsquigarrow} P' \qquad Q \overset{t}{\rightsquigarrow} Q'}{P \interleave Q \overset{t}{\rightsquigarrow} P' \interleave Q'}$$

Interleaved processes progress independently.

$$\frac{P \xrightarrow{(t,\mu)} P' \qquad Q \overset{t}{\rightsquigarrow} Q'}{P \interleave Q \xrightarrow{(t,\mu)} P' \interleave Q'} \qquad \frac{P \overset{t}{\rightsquigarrow} P' \qquad Q \xrightarrow{(t,\mu)} Q'}{P \interleave Q \xrightarrow{(t,\mu)} P' \interleave Q'}$$

Abstraction

The abstraction operator internalises all occurrences of events from the set of events being hidden. The set of initial external events of $P \setminus A$ is therefore those external events that P can initially perform that are not now internal.

$$\mathit{init}(P \setminus A) \quad = \quad \mathit{init}(P) \setminus A$$

If P can evolve, and can perform no internalised events, then $P \setminus A$ is also able to evolve. Otherwise, it will not be able to evolve beyond time $\mathit{0}$.

$$\frac{P \overset{t}{\rightsquigarrow} P'}{P \setminus A \overset{t}{\rightsquigarrow} P' \setminus A} \quad [\, A \cap \mathit{init}(P) = \{\} \vee t = \mathit{0} \,]$$

An abstracted process can progress normally if it cannot perform any internal events.

$$\frac{P \xrightarrow{(t,\mu)} P'}{P \setminus A \xrightarrow{(t,\mu)} P' \setminus A} \quad [\, A \cap \mathit{init}(P) = \{\} \,]$$

Abstracted events are labelled by τ, and must happen at time $\mathit{0}$ unless an external event happens at that time.

$$\frac{P \xrightarrow{(\mathit{0},a)} P'}{P \setminus A \xrightarrow{(\mathit{0},a)} P' \setminus A} \quad [\, a \notin A \,] \qquad \frac{P \xrightarrow{(\mathit{0},\mu)} P'}{P \setminus A \xrightarrow{(\mathit{0},\tau)} P' \setminus A} \quad [\, \mu \in A \cup \{\tau\} \,]$$

Event Renaming

If $f : \Sigma \longrightarrow \Sigma$ is an alphabet transformation, then $f(P)$ is able to perform $f(a)$ whenever P may perform a, and $f^{-\mathit{1}}(P)$ can perform a whenever P can perform $f(a)$.

$$\begin{aligned} \mathit{init}(f(P)) &= f(\mathit{init}(P)) \\ \mathit{init}(f^{-\mathit{1}}(P)) &= f^{-\mathit{1}}(\mathit{init}(P)) \end{aligned}$$

If a process can evolve, then so can its image under direct or inverse alphabet transformation.

$$\frac{P \overset{t}{\rightsquigarrow} P'}{f(P) \overset{t}{\rightsquigarrow} f(P')} \qquad \frac{P \overset{t}{\rightsquigarrow} P'}{f^{-1}(P) \overset{t}{\rightsquigarrow} f^{-1}(P')}$$

By convention, internal events remain unaltered under alphabet renamings, so we extend f so that $f(\tau) = \tau$. Thus we obtain

$$\frac{P \xrightarrow{(t,\mu)} P'}{f(P) \xrightarrow{(t,f(\mu))} f(P')} \qquad \frac{P \xrightarrow{(t,f(\mu))} P'}{f^{-1}(P) \xrightarrow{(t,\mu)} f^{-1}(P')}$$

Recursion

No rule is given for the term X, since this is not a process.

A recursively defined process must immediately unwind (via an internal action) before it is able to perform any visible actions.

$$\mathit{init}(\mu\, X \circ P) \quad = \quad \{\}$$

It may not evolve beyond the time of its internal action.

$$\frac{}{\mu\, X \circ F(X) \overset{0}{\rightsquigarrow} \mu\, X \circ F(X)}$$

It unwinds the recursion once at time 0.

$$\frac{}{\mu\, X \circ P \xrightarrow{(0,\tau)} P[(\mu\, X \circ P))/X]}$$

We are able to obtain a number of theorems which confirm that the operational semantics concurs with our intuition concerning the execution of processes. In each case the result follows from a straightforward structural induction on P. Example proofs are presented in Appendix A. The properties captured in Theorems 5.2, 5.4, 5.6 and 5.9 also appear in (Wang Yi, 1991) and (Nicollin and Sifakis 1990); we will follow Wang's terminology in these theorems.

Evolution

We establish the following results concerning the evolution relation $\rightsquigarrow$, and its interaction with the *init* function. The first theorem provides us with a result which we would expect to hold of the $\rightsquigarrow$ relation.

Theorem 5.1 A process can always evolve to itself in no time.

$$P \overset{0}{\rightsquigarrow} P$$

□

The next theorem indicates that the passage of time does not introduce any non-determinism into the execution of a process; hence any non-determinism that does arise must do so as a result of the occurrence of transitions.

Theorem 5.2 Time determinacy:

$$P \overset{t}{\rightsquigarrow} P_1 \wedge P \overset{t}{\rightsquigarrow} P_2 \Rightarrow P_1 = P_2$$

□

Furthermore, the passage of time cannot alter the communication events in which a process is prepared to engage.

Theorem 5.3 The set of possible initial events $init(P)$ remains constant under evolution.

$$P \overset{t}{\rightsquigarrow} P' \Rightarrow init(P) = init(P')$$

□

Finally, if a process is able to evolve to a certain time t, then it must evolve through every time before t; since time is continuous, this says that the process evolves continuously.

Theorem 5.4 Time continuity

$$P \overset{t''+t'}{\rightsquigarrow} P' \Leftrightarrow \exists\, P'' \bullet P \overset{t''}{\rightsquigarrow} P'' \wedge P'' \overset{t'}{\rightsquigarrow} P'$$

□

Transitions

The first theorem confirms that the set given by $init(P)$ indeed corresponds to precisely those events that P can initially perform, given by the transition relation $\longrightarrow$.

Theorem 5.5 The set of events P can initially perform is $init(P)$:

$$a \in init(P) \Leftrightarrow a \in \Sigma \wedge P \xrightarrow{(0,a)}$$

□

The next theorem concerns the interaction of the evolution relation and internal events; it confirms that if an internal event is possible for a process, then that process cannot evolve beyond that time; internal events must be performed as soon as possible. This is called τ-urgency in (Nicollin and Sifakis 1990).

Theorem 5.6 Maximal progress:

$$P \xrightarrow{(t',\tau)} P' \land t > t' \Rightarrow \neg \exists P'' \bullet P \overset{t}{\rightsquigarrow} P''$$

$\Box$

Conversely, if the process is unable to perform an internal action, then it can evolve for all time; if it is able to perform an internal action, then it can evolve up to the time that action is to be performed.

Theorem 5.7 A process can always evolve up to the time of the next τ action:

$$\begin{array}{rcl} P \overset{\tau}{\not\rightarrow} & \Rightarrow & \forall t \exists P' \bullet P \overset{t}{\rightsquigarrow} P' \\ P \xrightarrow{(t',\tau)} & \Rightarrow & \forall t \leq t' \exists P' \bullet P \overset{t}{\rightsquigarrow} P' \end{array}$$

$\Box$

From these two theorems the following corollary is immediate:

Corollary 5.8 If $P \xrightarrow{(t_1,\tau)}$ and $P \xrightarrow{(t_2,\tau)}$ then $t_1 = t_2$. $\Box$

Evolution cannot alter the set of possible communication events. If P is able initially to perform event a, and it is also able to evolve through time t, then it is able to perform event a at time t. Conversely, if P is able to perform a at time t, then it must be able to perform a at time 0, and to evolve for length of time t. Hence any occurrence of a timeout must be accompanied by an internal action, since timeout cannot occur purely through evolution.

Theorem 5.9 Persistency:

$$(P \xrightarrow{(0,a)} \land \exists P' \bullet P \overset{t}{\rightsquigarrow} P') \Leftrightarrow P \xrightarrow{(t,a)}$$

$\Box$

Finally, we see that our processes satisfy a finite variability property: that they cannot perform infinitely many events within a finite period of time.

Theorem 5.10 Finite variability: if there is a sequence of processes $\{P_i\}_{i=1}^{\infty}$ and a sequence of transitions $\{(t_i,\mu_i)\}_{i=1}^{\infty}$ such that

$$\forall\, i \bullet P_i \xrightarrow{(t_i,\mu_i)} P_{i+1}$$

then $\Sigma_{i=1}^{\infty} t_i = \infty$. □

It immediately follows that a process may reach any time merely by performing internal events and evolving, since evolution can always occur up to the next τ event (Theorem 5.7), and only a finite number of τ events can occur by any given time.

Corollary 5.11 Given any process P_0 and time T, there exists a sequence of processes $\{P_i\}_{i=1}^{n+1}$ and a sequence of times $\{t_i\}_{i=0}^{n}$ such that

$$\forall\, i < n \bullet P_i \xrightarrow{(t_i,\tau)} P_{i+1} \wedge P_n \overset{t_n}{\rightsquigarrow} P_{n+1} \wedge \Sigma_{i=0}^{n} t_i = T$$

□

These theorems give us a picture of the execution of a process. It will be a sequence of states (processes) connected by timed events; before these events occur, the set of events being offered by the process (given by *init*) remains constant.

Example

The following execution of a timeout process modelled using choice and sequential composition offers event a for 4 time units. If that event is not performed by that time, then the offer is withdrawn. This change of state is accompanied by τ actions, and an alternative course of action is offered and eventually pursued.

$$\begin{array}{lll}
 & (a \longrightarrow STOP \;\Box\; WAIT\ 4)\,;\, b \longrightarrow STOP & (init = \{a\}) \\
\xrightarrow{(4,\tau)} & (a \longrightarrow STOP \;\Box\; SKIP)\,;\, b \longrightarrow STOP & (init = \{a\}) \\
\xrightarrow{(0,\tau)} & b \longrightarrow STOP & (init = \{b\}) \\
\xrightarrow{(3,b)} & STOP & (init = \{\})
\end{array}$$

6 Mapping to $\mathcal{M}_{TF}$

The timed failures model $\mathcal{M}_{TF}$ for TCSP, discussed in (Davies and Schneider, 1992) and reproduced in Appendix B, gives the meaning of a TCSP program in terms of the timed traces and timed refusals that it may exhibit. A timed trace s is a finite sequence of timed events drawn from $\mathbf{R}^+ \times \mathbf{\Sigma}$ such that the times are non-decreasing. Note that the internal action τ is not recorded in a trace. We denote the set of timed traces by TT.

$$TT \;=\; \{s \in (\mathbf{R}^+ \times \mathbf{\Sigma})^* \mid \langle(\mathbf{t_1},\mathbf{a_1}),(\mathbf{t_2},\mathbf{a_2})\rangle \preceq \mathbf{s} \Rightarrow \mathbf{t_1} \leq \mathbf{t_2}\}$$

A refusal token is a set of timed events consisting of the cartesian product of a finite half open interval I and a set of events $A \subseteq \Sigma$, describing the continuous refusal of all events in A throughout the interval I. The set of refusal tokens is given by

$$RT \quad = \quad \{[t_1, t_2) \times A \mid 0 \leq t_1 < t_2 < \infty \wedge A \subseteq \Sigma\}$$

A timed refusal is a set of timed events corresponding to a record of events refused at particular times during an execution. It consists of a finite union of refusal tokens. The set of timed refusals TR is given by

$$TR \quad = \quad \{\bigcup R \mid R \subseteq RT \wedge R \text{ is finite}\}$$

A timed refusal may be thought of as a step function which maps times to sets of events.

An observation in this model is a pair consisting of a timed trace, and a timed refusal. A TCSP program P is associated with a set of such observations. The model $\mathcal{M}_{TF}$ consists of those sets that meet a number of axioms, or consistency conditions (such as prefix closure on traces). The semantic function $\mathcal{F}_{TF}$ maps each program to the corresponding set of behaviours. The value of $\mathcal{F}_{TF}$ applied to a composite program is defined in terms of the value of $\mathcal{F}_{TF}$ on its constituents. It is defined in Appendix B. We think of the set $\mathcal{F}_{TF}[\![P]\!]$ as the set of observations that may be made of some execution of the program P.

We will define a mapping Φ from transition systems to the power set of timed failures $\mathbf{P}(TF)$. The mapping Φ extracts the timed failure information from the transition system associated with a program. We will establish that Φ provides us with an equivalence, in the sense that $\Phi(P) = \mathcal{F}_{TF}[\![P]\!]$ for all TCSP processes P. The equivalence reduces to showing that the denotational semantics of program P given directly by the semantic function $\mathcal{F}_{TF}$ is the same as the semantics obtained by considering P as a node of the transition system and calculating its set of behaviours, using Φ, from the set of possible sequences of transitions starting at P.

Traces

We will need to translate between the sequences of timed transitions generated by the operational semantics (where the time associated with each event is the time since the last one) and the absolute time traces used in the denotational semantics (where the time associated with each event is the time since the trace started). We define two functions, $abs : (\mathbf{R}^+ \times \mathbf{\Sigma}^+)^* \longrightarrow \mathbf{T}(\mathbf{\Sigma}^+)^*_{\leq}$, and $rel : T(\Sigma^+)^*_{\leq} \longrightarrow (\mathbf{R}^+ \times \mathbf{\Sigma}^+)^*$, by the following equations:

$$\begin{aligned} abs(\langle\rangle) &= \langle\rangle \\ abs(\langle(t,\mu)\rangle^\frown w) &= \langle(t,\mu)\rangle^\frown abs(w+t) \end{aligned}$$

$$\begin{aligned} rel(\langle\rangle) &= \langle\rangle \\ rel(\langle(t,\mu)\rangle^\frown w) &= \langle(t,\mu)\rangle^\frown rel(w-t) \end{aligned}$$

Then the traces of a node may be defined as follows:

$$traces(N) \quad = \quad \{abs(w) \mid P \overset{w}{\rightarrow} \}$$

Refusals

Since the set of possible events from a node does not vary before a τ action is possible, it makes sense to say that a node N can refuse a set X precisely when it can perform no event from that set at time 0, since it will then be unable to perform any element of that set constantly up to the next τ action. We define the relation $\underline{\text{ref}}$ between nodes and subsets of Σ as follows:

$$N \ \underline{\text{ref}}\ X \quad \widehat{=} \quad \forall\, a \in X \bullet N \overset{(0,a)}{\not\longrightarrow}$$

A pair of nodes (N, N') is said to refuse a timed refusal set $\aleph$ on a given trace w (which may contain τ actions) if there is a sequence of nodes from N to N' such that it may follow the path between them by performing w, with successive nodes refusing the respective part of $\aleph$. It may be defined in terms of $\underline{\text{ref}}$:

$$\begin{aligned}
(N, N')\ \underline{\text{fail}}\ (\langle\rangle, \aleph) \quad &\widehat{=} \quad N = N' \wedge N\ \underline{\text{ref}}\ \sigma(\aleph) \wedge (t < end(\aleph) \Rightarrow N \overset{(t,\tau)}{\not\longrightarrow}\ ) \\
(N, N')\ \underline{\text{fail}}\ (\langle(t,\mu)\rangle^\frown w, \aleph) \quad &\widehat{=} \quad N\ \underline{\text{ref}}\ \sigma(\aleph \upharpoonleft t) \wedge \exists\, N'' \bullet N \overset{(t,\mu)}{\longrightarrow} N'' \\
&\qquad \wedge (N'', N')\ \underline{\text{fail}}\ (w, \aleph) - t \\
N\ \underline{\text{fail}}\ (w, \aleph) \quad &\widehat{=} \quad \exists\, N' \bullet (N, N')\ \underline{\text{fail}}\ (w, \aleph)
\end{aligned}$$

Observe that if $t = 0$ then $\aleph \upharpoonleft t = \{\}$, and so is trivially refused by N.

Definition 6.1 If $TS = \langle NODES, A_\Sigma, \ \longrightarrow\rangle$, then the corresponding function $\Phi : NODES \longrightarrow \mathcal{M}_{TF}$ is defined as follows:

$$\Phi(N) \quad = \quad \{(w \setminus \tau, \aleph) \mid N\ \underline{\text{fail}}\ (w, \aleph)\}$$

□

Equivalence for TCSP

It is straightforward to prove (by induction on the structure of timed CSP terms) that the behaviours of the transition system (produced by the function Φ) are identical to those provided by the denotational semantics, thus establishing equivalence.

Definition 6.2 A *partial-binding* η is a total mapping from timed CSP variables $\mathcal{V}$ to timed CSP terms. A *binding* η is a partial-binding whose range is contained in the set of TCSP processes. □

Definition 6.3 The value $(P)\eta$ of a timed CSP term P in a binding η is the term obtained by replacing all occurrences of free variables X by its value $\eta(X)$ in binding η. It may therefore be defined on the BNF for timed CSP terms as follows

$$\begin{aligned} (X)\eta &\mathrel{\widehat{=}} \eta(X) \\ (\mu X \circ P)\eta &\mathrel{\widehat{=}} \mu X \circ ((P)\eta_{[X \to X]}) \end{aligned}$$

Here we define partial-binding $\eta_{[X\to X]}$ to be that partial-binding which maps process variable X to term X, and $\eta_{[X\to X]}[Y] = \eta(Y)$ when $Y \neq X$.

The other clauses are defined in the obvious way; η distributes over all of the other operators.

□

Theorem 6.4 If V is a set of variables, and η is a partial-binding that acts as the identity on V and maps variables not in V to processes, then all the free variables of $(P)\eta$ are contained in V. □

Corollary 6.5 If η maps all variables to TCSP processes, then $(P)\eta$ is a process □

Definition 6.6 If η is a binding, then we define $\overline{\eta}$ to be the binding for $\mathcal{M}_{TF}$ (a mapping $\mathcal{V} \longrightarrow \mathcal{M}_{TF}$) by the equivalence $\overline{\eta}[X] = \Phi(\eta(X))$ □

We have finally reached a position where we can establish the equivalence result.

Theorem 6.7 For all timed CSP terms P and bindings η we have $\Phi((P)\eta) = \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$. □

Corollary 6.8 For all TCSP processes we have $\Phi(P) = \mathcal{F}_{TF}[\![P]\!]$ □

Testing Equivalence and the Failures Model

A test is a TCSP process T whose alphabet has been augmented with a special success event ω. Testing consists of running T in parallel with the process being tested, and hiding all events except ω. We write Σ^ω to denote $\Sigma \cup \{\omega\}$. We say that P $\underline{\text{may}}$ T if there is an execution of the process $(P \;{}_{\Sigma}\|_{\Sigma^\omega}\; T) \setminus \Sigma$ that contains a state in which the event ω may be performed. Formally, P $\underline{\text{may}}$ T if there is a sequence of states $Q_1, \ldots Q_{n+1}$ such that

$$(P \;{}_{\Sigma}\|_{\Sigma^\omega}\; T) \setminus \Sigma \xrightarrow{(t_0,\tau)} Q_1 \xrightarrow{(t_1,\tau)} Q_2 \ldots \xrightarrow{(t_n,\tau)} Q_{n+1} \quad \wedge \quad Q_{n+1} \xrightarrow{(0,\omega)}$$

The processes P and Q are equivalent under may-testing if for all tests T, we have P $\underline{\text{may}}$ $T \Leftrightarrow Q$ $\underline{\text{may}}$ T. A denotational semantics is fully abstract with respect to a notion of testing (and an operational semantics) if two processes are equivalent under testing precisely when they have the same denotational semantics.

It turns out that the timed failures semantic model is fully abstract with respect to the notion of testing given here. We first obtain that any two processes with differing denotational semantics in the failures model may be distinguished by some test. Consider $(s, \aleph) \in \mathcal{F}_{TF}[\![P]\!]$ and $(s, \aleph) \notin \mathcal{F}_{TF}[\![Q]\!]$. Then let $s = \langle (t_1, a_1), \ldots, (t_n, a_n) \rangle$ and $\aleph = \bigcup_{i=1}^{m} [t'_i, t''_i) \times A_i$.

A useful derived operator allows interleaving on some events, and enforces synchronisation on others. We define

$$P \underset{A}{\|} Q \quad = \quad h(f(P)\ {}_{f(\Sigma)}\|_{g(\Sigma)}\ g(Q))$$

where the alphabet mappings f, g, and h are defined by

$$\begin{array}{rcll} f(x) & = & c.x & \text{if } x \in A \\ & & l.x & \text{if } x \notin A \\ g(x) & = & c.x & \text{if } x \in A \\ & & r.x & \text{if } x \notin A \\ h(d.x) & = & x & \text{if } d \in \{l, r, c\} \\ & & d.x & \text{otherwise} \end{array}$$

This operator may be generalised as follows:

$$\big|\big|_A \{P_i\} \quad = \quad (P_1\ \big|\big|[A](P_2\ \big|\big|[A] \ldots P_n) \ldots)$$

and where $\big|\big|_A$ applied to a singleton set is simply the process contained in that set.

The following test $T_{(s,\aleph)}$ will then distinguish P from Q. We first define a test T_s which is (nondeterministically) able to perform trace s, and only then may it perform ω. This will be placed in parallel with a process which may perform ω only after offering everything in the refusal set $\aleph$ and having no offer accepted.

$$\begin{array}{rcl} T_{\langle\rangle} & = & \omega \longrightarrow STOP \\ T_{\langle (t,a) \rangle ^\frown s} & = & WAIT\ t\,;((a \longrightarrow T_{s-t}) \overset{0}{\triangleright} STOP) \end{array}$$

If $\aleph$ is the empty set, then we take $T_{(s,\aleph)} = T_s$. Otherwise, we define $T_{(s,\aleph)}$ as follows:

$$\begin{array}{rcl} T_{(s,\aleph)} & = & T_s \\ & & \underset{\{\omega\}}{\|} \\ & & \big|\big|_{\{\omega\}} \{ WAIT\ t'_i\,;((a : A_i \longrightarrow STOP) \overset{t''_i - t'_i}{\triangleright} \omega \longrightarrow STOP) \mid 1 \leq i \leq m \} \end{array}$$

Then P $\underline{\text{may}}$ $T_{(s,\aleph)}$, since P may perform a sequence of transitions corresponding (under Φ) to $(s, \aleph)$, which enable it to pass the test. Conversely, if such a sequence is not possible, then the test cannot be passed, and so $\neg(Q \ \underline{\text{may}}\ T_{(s,\aleph)})$. Hence there is a test that distinguishes P from Q.

To obtain full abstractness, we must also show that programs with identical semantics are not distinguishable by some test. This follows from the fact that $P \ \underline{\text{may}}\ T \Leftrightarrow \mathcal{F}_{TF}[\![(P\ {}_{\Sigma}\|_{\Sigma^\omega}\ T) \setminus \Sigma]\!] \neq \mathit{STOP}$. (I am grateful to Alan Jeffrey for this observation.) If $\mathcal{F}_{TF}[\![P]\!] = \mathcal{F}_{TF}[\![Q]\!]$, it follows for any TCSP process T that $P \ \underline{\text{may}}\ T \Leftrightarrow Q \ \underline{\text{may}}\ T$, as required. Thus $\mathcal{M}_{TF}$ is the weakest model of TCSP that includes trace congruence.

7 Comparisons and Discussion

Specific Comparison

The work presented here is essentially an operational semantics for real-time concurrency with a continuous time domain. The approach is in some ways similar to that presented by Wang in (Wang Yi, 1991), where an operational semantics for a real-time version of CCS is presented. The work involved in forcing internal events to occur as soon as possible is also present in the timed CCS work, though the point at which it appears is in the definition of parallelism (as we would expect) since it is at that point that visible events may become internalised. We have used the same technique in this paper as was previously used by Wang in order to identify the initial events of a process; the initial events P may perform was called $\mathcal{S}ort(P)$. Thus $P \mid Q$ is permitted to evolve beyond time 0 only if P and Q cannot communicate internally: if $\mathcal{S}ort(P) \cap \overline{\mathcal{S}ort(Q)} = \{\}$. We have also established properties of processes (agents) that correspond to the properties of maximal progress, determinacy, continuity, and persistency given in (Wang Yi, 1991).

There are of course cosmetic differences between the two approaches. Where we use the evolution relation $P \overset{t}{\rightsquigarrow} P'$, Wang uses a time passing event $\varepsilon(t)$, and writes $P \xrightarrow{\varepsilon(t)} P'$ to say that P becomes P' under the passage of t units of time. The advantage of this approach is that it separates the passage of time from the performance of events: either time will pass, or an event will happen at the current moment. Hence events do not need to be labelled with times. Where we write $P \xrightarrow{(t,a)} P'$, Wang will write $P \xrightarrow{\varepsilon(t)} P'' \xrightarrow{a} P'$ (for some P''). The two approaches appear to be equivalent in that they contain the same information, since Theorem 5.9 establishes that evolutions and events performed at the current moment are sufficient to define all the transitions of a process. The motivation for our approach is that by ensuring evolutions by themselves do not appear in the transition system the mapping to the denotational semantics is more direct. Furthermore, in keeping the evolutions distinct from the transitions we sidestep the danger that infinite sequences of evolutions may generate difficulties: if a process P_0 may evolve up to time 2, then by continuity there will be an infinite sequence of processes P_n such that $P_n \xrightarrow{\varepsilon(2^{-n})} P_{n+1}$. This does not cause problems in (Wang Yi, 1991), but a notion of equivalence based on testing would have to avoid treating it as a maximal computation, which may lead to undesirable complications.

General Comparisons

There now exist a wide variety of formalisms for describing and analysing real-time systems. These all support the modelling of time-critical behaviour and timeout, although they differ in much the same way as programming languages differ: although they all have (roughly) the same expressive power, they vary with regard to how easily they support higher level conceptual tools such as abstraction and compositionality, and also how comprehensible programs are. They also treat the underlying notion of time in different ways.

Discrete versus dense time

The time domain in some approaches is discrete, and in others is continuous, or at least dense. It has been argued in (Joseph and Goswami 1988) that different kinds of computation problems are best addressed using different notions of time, and hence that there is no 'best' uniform approach. Some formalisms (eg (Moller and Tofts 1990), (Wang Yi, 1991),(Gerth and Boucher, 1987)) may use either a dense or a discrete time domain.

The march of time

It is generally accepted that time should be able to pass for ever for any real system. Many algebras, such as those presented in (Hennessy and Regan, 1991), (Wang Yi, 1991), (Nicollin and Sifakis 1990), (Moller and Tofts 1990), (Baeten and Bergstra, 1991), and (Liang Chen, 1991) permit programs that may perform an infinite number of internal or visible events by a particular time, in effect preventing time from passing beyond that point. Further, in (Moller and Tofts 1990) and (Liang Chen, 1991) there are processes which neither perform any action nor proceed through time. For these approaches, the passage of time seems rather weak compared with the execution of processes. It has been observed by Jeffrey that the advantage of allowing such time-stopping processes in a language is that a complete axiomatization often becomes possible (Jeffrey 1992) (although he shows in (Jeffrey 1990) that it is possible to do without such processes). It also permits reasoning about broken processes, since a process such as an infinite loop where cycles take no time should be considered as dangerous, in that it requires that time should stop (an impossible condition to fulfil in practice) in order to execute successfully. At some stage during verification there should therefore be an obligation to establish that the system does not require the stoppage of time in order to execute. Any such system are termed *well-timed* systems in (Nicollin and Sifakis 1990).

Other formalisms, such as those described in (Ortega-Mallén and de Frutos-Escrig 1990), (Hooman and Widom, 1989), (Barringer, Kuiper, and Pnueli, 1985), (Jeffrey 1990), and (Murphy 1990), as well as Timed CSP, have a stronger notion of time passing, in that no process which requires the stoppage of time can be written in the language. This is usually accomplished either by insisting that recursive calls are guarded by some delay in a recursively defined process (as in (Ortega-Mallén and de Frutos-Escrig 1990) and (Jeffrey 1990)) or insisting that the recursive call itself takes some time, which ensures time-guardedness. In (Barringer, Kuiper, and Pnueli, 1985) the semantic domain itself precludes the possibility of this form of divergence by ensuring that all computations within the domain of discourse satisfy an axiom of finite variability which forbids the performance of infinitely many actions in a finite time; the result is that no process can exhibit such behaviour, not even an unguarded loop.

Underlying Semantics

Arguably the most important distinction between the various approaches is the ease with which correctness may be established. This will of course be bound up with the nature of the process language, since the simplicity of the system's description in that language will have a bearing on the ease with which it may be verified. However, the underlying semantics will also be critical, since it provides the foundation for the verification method to be used in establishing correctness; different methods will be appropriate for different kinds of problem.

The algebraic approach to semantics (Baeten and Weijland, 1990) is to define the meaning of a language by giving a complete set of algebraic laws concerning the relationships between different programs. Two processes are said to be identical if it is possible to establish that they are equal by means of the laws, and otherwise they are distinct. This is the approach taken in (Baeten and Bergstra, 1991). Specifications are simply processes written in the process algebra, and a verification that a system meets that specification will be a calculation using the laws that the process algebra description of the system is equal to the specifying process.

The operational approach defines the meaning of a program in terms of how an ideal machine would execute it, for example in terms of transition systems (as in (Hansson 1991), (Hennessy and Regan, 1991), (Quemada and Fernandez 1987), (Moller and Tofts 1990), (Nicollin and Sifakis 1990), and (Ho-Stuart, Zedan, Fang, and Holt (1992)), or Petri nets (see (Coolahan and Roussopoulos, 1985), (Leveson and Stolzy. 1987)). Equivalence of processes is usually defined either in terms of bisimulation (see (Milner 1989)) or in terms of testing equivalence (see (Hennessy, 1988)). Equivalence of processes may be established equationally, by means of a complete axiomatisation for the language, permitting the use of processes as specifications as in the algebraic approach. Alternatively, a logical language (e.g. Hennessy-Milner logic in (Hennessy and Milner, 1985)) may be used to capture specific properties which may be treated as specifications; many different processes may all be shown to meet this sort of specification.

The denotational approach defines the meaning of a process to be a particular object in the domain of possible program meanings; usually the meaning of a program will be the set of possible 'behaviours' that it may perform (as in (Gerth and Boucher, 1987), (Barringer, Kuiper, and Pnueli, 1985), (Gerber, Lee, and Zwarico, 1988), (Jeffrey 1990), (Reed 1988) (Koymans, Shyamasundar, de Roever, Gerth, and Arun-Kumar 1988), and

(Hooman and Widom, 1989)). A denotational semantics is compositional, in the sense that the meaning of any composite program is entirely determined by the meanings of its constituent parts and the way they are combined. A specification is usually given by a predicate on behaviours, and a process meets that specification if all of its possible behaviours meet the predicate. The nature of the behaviours underlying the semantic model will of course influence the choice of specification language, but different specification languages may be used for different applications (see e.g. (Jackson 1991)), and this supports different levels of abstraction, essential for the analysis of large systems. The compositional nature of the semantics often makes it possible to construct a complete proof system for the language, permitting the reduction of proof obligations on compound processes to proof obligations on their component processes, as in (Davies and Schneider, 1990a) and (Hooman and Widom, 1989). However, there is no guarantee that such a proof system will exist; for example, the timed failures-stability model for Timed CSP cannot have such a proof system (see (Schneider, 1990)).

Of course these three brands of semantic approach are not mutually exclusive, but rather are complementary. In (Jeffrey 1991), Jeffrey provides a failures semantics for his process algebra APA, and establishes that it is equivalent to extracting the failures information directly from the transition graph given by the operational semantics. He also establishes that trace congruence (the largest congruence smaller than trace equivalence) is the same as failures equivalence, for time-stop free processes in a time domain with no maximal elements; the theory presented in this paper is one such example. Jeffrey's treatmGent is more general than that presented in this paper, permitting partially ordered time. Another example is provided by the process algebra presented in (Baeten and Bergstra, 1991), which is endowed with an operational semantics, permitting the application of operational techniques to such processes; processes defined operationally are given complete axiomatizations which enable algebraic reasoning, and may also be provided with a denotational semantics (as this paper illustrates); and processes defined denotationally may also give rise to complete axiomatisations (as in CSP) or fully abstract operational semantics.

Discussion

In laying bare the operational intuitions underlying timed CSP we are now able to address questions concerning which features of the language are consequences of its nature, and which features are simply the consequence of design decisions.

Instant causality

The original models for Timed CSP (Reed and Roscoe, 1986), (Reed and Roscoe, 1987), (Reed 1988) did not permit instant causality between visible events, insisting rather that simultaneous events may be observed in any order. This philosophy conflicts with the operational treatment of sequential composition $P\ ;\ Q$ given above, where events from P

must be observed before events from Q, even if the last event of P and the first event of Q occur simultaneously. In order to reflect the original denotational sequential composition, we must allow P's final events to occur after control has passed to Q. Within the present framework, this may most easily be accomplished by using the first rule below in place of the original rule concerned with transfer of control through sequential composition. A new construct $(Q, P)@\mathit{0}$ is also required: it behaves as Q but also allows events from P at time $\mathit{0}$; it may not evolve beyond 0 unless it first disposes of P (via a τ action), and its initial events will be the union of those of P and of Q.

$$\frac{P \xrightarrow{(\mathit{0},\checkmark)} P'}{P\ ;\ Q \xrightarrow{(\mathit{0},\tau)} (Q, P)@\mathit{0}} \qquad \frac{}{(Q, P)@\mathit{0} \xrightarrow{(\mathit{0},\tau)} Q}$$

$$\frac{P \xrightarrow{(\mathit{0},\mu)} P'}{(Q, P)@\mathit{0} \xrightarrow{(\mathit{0},\mu)} (Q, P')@\mathit{0}} \qquad \frac{Q \xrightarrow{(\mathit{0},\mu)} Q'}{(Q, P)@\mathit{0} \xrightarrow{(\mathit{0},\mu)} (Q', P)@\mathit{0}}$$

The original models for TCSP are fully abstract (for the same notions of testing) with respect to an operational semantics with this definition of sequential composition.

Other operators concerned with causality are the event prefix operator and recursion. It makes little difference to the operational rules whether or not there is a delay following the occurrence of an event, and whether or not recursion should take time to unwind. Although its operational effects are negligible, instant prefixing has disproportionate consequences for the more abstract denotational semantics, since simultaneous causally connected events are now possible. The original models for Timed CSP cannot model such behaviour (see (Davies and Schneider, 1990b)), because the information contained in a trace is simply a record of bags of events occurring at particular times; there is an assumption that all events occurring in such a bag are causally independent. This assumption is made explicit by the inclusion in each of the original models of an axiom that states that simultaneous events may occur in a trace in any order. The model used here (see Appendix B) omits this axiom, thus permitting instant prefixing.

Hiding

In common with some other approaches (e.g. (Hennessy and Regan, 1991)) we insist on *maximal progress*: that internal events happen as soon as possible. This is guaranteed by preventing a process from evolving past the time of a hidden action.

There are two principle ways in which maximal progress may be dropped. The first is simply to allow internal events to be deferred, allowing evolution past the times at which they are available, in the same way as visible events are currently treated. In this case, it is reasonable for a machine to idle for ever, never making any internal progress.

The semantics for this form of hiding is straightforward: all side conditions concerning *init* are removed, from the evolution rules and the transition rules given earlier. The side conditions prevented the progress of time if there were internal events available, the mechanism by which maximal progress was obtained.

In the absence of some other mechanism for insisting that events occur, we no longer have the ability to analyse liveness properties of systems, since programs need not progress: the process obtained by hiding event a in the process $a \longrightarrow P$ would never be guaranteed to begin executing $P \setminus a$, since it need never perform the internal a. A denotational semantics for these operators would find refusal information superfluous, since processes may always refuse events, preferring to evolve; in this case $init(P)$ would still signify those events that P could initially perform. Timed traces would thus be sufficient to model processes in the absence of maximal progress. Observe that it is only the hiding operator and sequential composition that introduced maximal progress, and thus the need for refusal information. The rules for the other operators define what is possible for a composite program purely in terms of what is possible for its components, so those operators all respect trace equivalence; and so the traces model is well-defined for them. If none of the operators introduce maximal progress, then trace equivalence is a congruence.

The second approach, taken in (Baeten and Bergstra, 1991), is to insist that events must happen at one of the times they are available, but not necessarily at their earliest moment of availability. Maximal progress is again absent, and choices are between possible executions. Thus the program which is prepared to perform an event over an interval may be executed by performing the event at any point of that interval. A corresponding hiding operator would insist that an event be performed at some point when it is available, but not necessarily at its earliest time (even if there is one). In contrast to the previous case, however, evolution beyond the end of the interval is not permitted.

Other models for timed CSP

The infinite timed failures model, and must-testing

This model is an extension of the timed failures model, developed to overcome the limitations of that model in addressing problems associated with unbounded nondeterminism. In the infinite timed failures model, traces may be either finite or infinite in length, and refusal sets may be recorded over all time. These enhanced behaviours remain bound by the finite variability constraint, which requires here that the set of times recorded in an infinite trace must be unbounded, and also that the projection of an infinite refusal set to any initial finite interval $[0, t)$ must be an element of $RSET$. This ensures that the refusal set changes only finitely often in a finite interval.

Processes are modelled as sets of these behaviours, subject to a number of consistency conditions. The fixed point theory is not straightforward, as the space is neither a complete partial order, nor a complete metric space. Details of the model may be found in

(Schneider, 1992). This model allows the distinction of processes such as

$$\sqcap_{n \in \mathbf{N}} \mathit{WAIT}\ n\ ;\ a \longrightarrow \mathit{STOP} \quad \text{and} \quad \mathit{STOP} \sqcap (\sqcap_{n \in \mathbf{N}} \mathit{WAIT}\ n\ ;\ a \longrightarrow \mathit{STOP})$$

where the first is able to refuse the event a for any length of time, but not forever; the second may also refuse it forever. These processes are identified in the timed failures model, but are distinguished by the infinite behaviour $(\langle\rangle, [0, \infty) \times \{a\})$, which is possible for the second, but not for the first.

It is shown in (Schneider, 1992) that the infinite failures information may be extracted from the operational semantics, using the same approach as was taken in this paper. Furthermore, it was shown that a test could be constructed which would fail for a given infinite timed failure, but would succeed otherwise. More precisely, given an infinite timed failure $(s, \aleph)$, a process $T'_{s,\aleph}$ could be constructed with the following property: if a maximal execution of $(P\ {}_{\Sigma}\|_{\Sigma^{\omega}}\ T'_{s,\aleph}) \setminus \Sigma)$ passes through a particular sequence of states, then one of those states must have $\xrightarrow{(0,\omega)}$ as a possible transition, unless $(s, \aleph)$ is an observation of P's contribution to the execution.

It follows that equivalence in this model amounts to must-testing: P <u>must</u> T if every execution of $(P\ {}_{\Sigma}\|_{\Sigma^{\omega}}\ T) \setminus \Sigma$ passes through a state in which $\xrightarrow{(0,\omega)}$ is possible. Processes distinguished in the infinite model will have some behaviour, and hence some test, that will distinguish them. In the example above, the test $a \longrightarrow \omega \longrightarrow \mathit{STOP}$ distinguishes the two processes. Furthermore, if two processes P_1 and P_2 have the same behaviours, then if P_1 fails T on some execution, then P_2 has some execution for which T passes through the same sequence of states: an execution corresponding to the same behaviour.

Since the infinite failures model is strictly finer than the standard failures model, it follows that equivalence under must-testing is strictly stronger than equivalence under may-testing. This result follows because we have disallowed time-stopping processes, and insisted on finite variability. Certainly if such features were allowed in the language, then must-testing is no longer stronger than may-testing. Consider a process $\mathit{TIMESTOP}$ which is unable to evolve beyond 0, or to perform any transitions. Any process P will be equivalent under may-testing to $P \sqcap \mathit{TIMESTOP}$, but the must-testing equivalence does not hold for any process that allows time to progress beyond 0; since the test $\mathit{WAIT}\ t\ ;\ \omega \longrightarrow \mathit{STOP}$ makes the distinction whenever P is guaranteed to allow t units of time to pass. The absence of finite variability also allows the passage of time to be curtailed. For example, if we permit the improper recursion $\mu X \circ a \longrightarrow X$, then hiding a yields a process which performs an infinite sequence of internal actions at time 0, and never allows time to progress. The same effect is achieved by the recursion $\mu X \circ X$. Both of these divergent processes are equivalent under must-testing to $\mathit{TIMESTOP}$. More interestingly, the process $\mu X \circ (X \,\square\, a \longrightarrow \mathit{STOP})$ is also equivalent to these two under must-testing; performance of the event a cannot be guaranteed, and the process may unfold its recursion for ever.

Finally, it is worth noting that the infinite timed failures model is equivalent to the standard untimed failures model in the absence of the two infinite choice operators. The infinite behaviours of programs will be given by the finite ones: an infinite behaviour will be possible precisely when all of its finite approximations are. For such programs, equivalence under may-testing and must-testing coincide.

The timed failures-stabilities model

This model, presented in (Reed and Roscoe, 1991), associates a unique *stability value* α with each trace-refusal pair, yielding a timed failure-stability triple $(s, \alpha, \aleph)$. The value $\alpha \in \mathbf{R}^+ \cup \{\infty\}$ is the earliest time after which it may be guaranteed that no further internal progress is possible, or else it is ∞ if there is no such time. The processes $STOP$, $WAIT\ 1\ ; STOP$ and $\mu X \circ WAIT\ 1\ ; X$ all have the same failure sets, but they differ on stability values, associating the values 0, 1, and ∞ with (for example) the failure $(\langle\rangle, [0, 2) \times \{a\})$. The choice $STOP \sqcap WAIT\ 1\ ; STOP$ has the triple $(\langle\rangle, 1, [0, 2) \times \{a\})$ as a possible behaviour, since stability (no further internal action) cannot be guaranteed before time 1; even though some execution consistent with that failure will make no internal transitions beyond time 0, this cannot be said for all such possible executions.

It is straightforward to extract the stability information from the operational semantics. Given a process P and a failure $(s, \aleph) \in \mathcal{F}_{TF}[\![P]\!]$, we define

$$stab_P(s, \aleph) \quad = \quad \sup\{end(w) \mid P \ \underline{\text{fail}}\ (w, \aleph) \wedge w \setminus \tau = s\}$$

where the supremum of an unbounded set is ∞.

There does not seem to be a natural form of test which corresponds to equivalence in the model with stabilities. Concurrent interactions between processes are defined purely by their failure sets, and so a test to distinguish processes which differ only on stability values would require access to internal transitions. How this might best be achieved is a topic for future research.

section*Acknowledgements

Thanks are due to Jim Davies, Dave Jackson, Mike Reed, Bill Roscoe, Tony Hoare, Mike Mislove, Alan Jeffrey, Tim Regan, Brian Scott, Paritosh Pandya, David Murphy, Wang Yi, Uno Holmer, and members of the SPEC project, for a number of useful discussions and comments on this work and related topics. The full abstractness result was obtained with the help of Tim Regan and Alan Jeffrey, and the encouragement of David Murphy. I would also like to thank the anonymous referees for many valuable comments and suggestions. Thanks are also due to the organisers of the Chalmers Workshop in Concurrency for inviting me to the May 1991 meeting, where an early version of this work was first presented.

I am grateful to the ESPRIT BRA SPEC project, the U.S. Office of Naval Research, and latterly to the Science and Engineering Research Council (fellowship B91/RFH/312), for funding this research.

References

BAETEN, J. C. M., AND BERGSTRA, J. A. (1991), Real time process algebra, *Formal Aspects of Computing* **3**, pp 142–188.

BARRINGER, H., KUIPER, R., AND PNUELI, A. (1985), A really abstract concurrent model and its temporal logic, *in* "Proceedings, 13th ACM Symposium on the Principles of Programming Languages," pp 173–183.

BAETEN, J. C. M., AND WEIJLAND, W. P. (1990), Process Algebra, Cambridge University Press.

LIANG CHEN (1991), An interleaving model for real-time systems, report ECS-LFCS-91-184, University of Edinburgh.

COOLAHAN, J. E., AND ROUSSOPOULOS, N. (1985), A timed Petri net methodology for specifying real-time system timing constraints, *in* "Proceedings, International Workshop on Timed Petri Nets," Torino, Italy.

DAVIES, J. W. (1991), Specification and proof in Timed CSP, D. Phil thesis, University of Oxford.

DAVIES, J. W., AND SCHNEIDER, S. A. (1989), An introduction to timed CSP, technical monograph PRG-75, University of Oxford.

DAVIES, J. W., AND SCHNEIDER, S. A. (1990a), Factorising proofs in Timed CSP, *in* "Proceedings, 5th Conference on Mathematical Foundations of Programming Semantics," pp 129–159, Lecture Notes in Computer Science, Vol 442, Springer-Verlag.

DAVIES, J. W., AND SCHNEIDER, S. A. (1990b), Waiting for timed CSP, technical report TR-3-90, University of Oxford.

DAVIES, J. W., AND SCHNEIDER, S. A. (1992), A brief history of timed CSP, technical monograph PRG-96, University of Oxford.

GERBER, R., LEE, I., AND ZWARICO, A. (1988), A complete axiomatization of real-time processes, technical report, University of Pennsylvania.

GERTH, R., AND BOUCHER, A. (1987), A timed model for extended communicating processes, *in* "Proceedings, 14th International Colloquium on Automata, Languages and Programming," pp 95–114, Lecture Notes in Computer Science, Vol 267, Springer-Verlag.

HENNESSY, M. (1988) Algebraic Theory of Processes, MIT Press.

HANSSON, H. (1991), A calculus for communicating systems with time and probabilities, Ph.D. thesis, University of Uppsala, HENNESSY, M., AND MILNER, R. (1985), Algebraic laws for nondeterminism and concurrency, *Journal of the ACM* **32** pp 137–161.

HOARE, C. A. R. (1985), Communicating Sequential Processes, Prentice-Hall.

HENNESSY, M., AND REGAN, T. (1991), A process algebra for timed systems, report 5/91, University of Sussex.

HOOMAN, J., AND WIDOM, J. (1989), A temporal-logic based compositional proof system for real-time message passing, *in* "Proceedings, Parallel Architectures and Languages Europe," volume II, pp 424–441, Lecture Notes in Computer Science, Vol 366, Springer-Verlag.

HO-STUART, C., ZEDAN, H. S. M., FANG, M., AND HOLT, C. M. (1992), PARTY: A process algebra with real-time from York, report YCS 177, University of York.

JACKSON, D. M. (1991), A temporal logic proof system for Timed CSP technical report TR 2-91, University of Oxford.

JEFFREY, A. (1990), Discrete Timed CSP PMG Memo 78, Chalmers University of Technology, Sweden.

JEFFREY, A. (1991), Observation spaces and timed processes, D. Phil thesis, University of Oxford, 1991.

JEFFREY, A. (1991), Translating timed process algebra into prioritized process algebra, *in* "Proceedings, Nijmegen symposium on real-time and fault-tolerant systems," pp 493–506, Lecture Notes in Computer Science, Vol 571, Springer-Verlag.

JOSEPH, M., AND GOSWAMI, A. (1988), Relating computation and time, Research report 138, University of Warwick.

KOYMANS, R., SHYAMASUNDAR, R. K., DE ROEVER, W.P., GERTH, R., AND ARUN-KUMAR, S. (1988), Compositional semantics for real-time distributed computing, *Inform. and Comput.* **79**, pp 210–256.

LEVESON, N. G., AND STOLZY, J. L. (1987), Safety analysis using Petri nets, *in* IEEE Transactions on Software Engineering, **SE-13**, pp 386–397.

MILNER, R. (1989), Communication and Concurrency, Prentice-Hall.

MOLLER,F., AND TOFTS, C. (1990), A temporal calculus of communicating systems, *in* "Proceedings, CONCUR 1990," pp 401–415, Lecture Notes in Computer Science, Vol 458, Springer-Verlag.

MURPHY, D. V. J. (1990), Time, causality, and concurrency, Ph.D. thesis, University of Surrey.

Nicollin, X., and Sifakis, J. (1990), The algebra of timed processes ATP: theory and application, RT-C26, Projet SPECTRE, Laboratoire de Génie Informatique de Grenoble.

Ortega-Mallén, Y., and de Frutos-Escrig, D. (1990), Timed observations: a semantic model for real-time concurrency, *in* “IFIP-TC2 — Working Conference on Programming Concepts and Methods,” North-Holland.

Peterson, J. L. (1977), Petri nets, ACM Computing Surveys, **9**, pp 223–252.

Plotkin, G. D. (1981), A structural approach to operational semantics, Report DAIMI-FN-19, Århus University, Denmark.

Quemada, J., and Fernandez, A. (1987), Introduction of quantitative relative time into LOTOS, *in* “Protocol Specification, Testing and Verification VII” North Holland.

Reed, G. M. (1988), A uniform mathematical theory for real-time distributed computing, D.Phil thesis, University of Oxford.

Roscoe A. W. (1988), Unbounded nondeterminism in CSP, technical monograph PRG-67, University of Oxford.

Reed, G. M., and Roscoe, A. W. (1986), A timed model for communicating sequential processes, *in* “Proceedings, 13th International Colloquium on Automata, Languages and Programming,” pp 314–323, Lecture Notes in Computer Science, Vol 226, Springer-Verlag.

Reed, G. M., and Roscoe, A. W. (1987), Metric spaces as models for real-time concurrency, *in* “Proceedings, Workshop on the Mathematical Foundations of Programming Language Semantics,” pp 331–343, Lecture Notes in Computer Science, Vol 298, Springer-Verlag.

Reed, G. M., and Roscoe, A. W. (1991), Analysing $\mathcal{M}_{TFS}$: A study of nondeterminism in real-time concurrency, *in* “Proceedings, Concurrency: Theory, Language, and Architecture,” pp 36–63, Lecture Notes in Computer Science, Vol 491, Springer-Verlag.

Schneider, S. A. (1990), Communication and correctness in real-time distributed computing, D.Phil thesis, University of Oxford.

Schneider, S. A. (1992), Unbounded nondeterminism in timed CSP, technical report TR-13-92, University of Oxford.

Wang Yi (1991) A calculus of real time systems, Ph.D. Thesis, Chalmers University of Technology, Sweden.

A Example cases in structural induction proofs

Each of the theorems in sections 5 and 6 is proved by means of a structural induction on the syntax of timed CSP. The cases are usually straightforward, and so only example cases from each proof are presented here. The difficult cases are all presented. Minimality of $\rightsquigarrow$ and $\rightarrow$ under their respective inference rules is often used (see e.g. the proof of Theorem 5.2 below): by minimality, any instance of $P \overset{t}{\rightsquigarrow} P'$ must be a conclusion of one of the rules, since if it were not, then the relation E', defined to be exactly the same as $\rightsquigarrow$ except that $E'(P, P', t)$ does not hold, is strictly smaller than $\rightsquigarrow$ and is still closed under the inference rules, contradicting minimality of $\rightsquigarrow$. The same is true for the $\rightarrow$ relation.

Theorem 5.1: A process can always evolve to itself in no time.

$$P \overset{0}{\rightsquigarrow} P$$

Proof

Case $P \mathbin{|||} Q$

Assume $P \overset{0}{\rightsquigarrow} P$ and $Q \overset{0}{\rightsquigarrow} Q$. It follows immediately from the rule for $\rightsquigarrow$ on the interleaving operator, that $P \mathbin{|||} Q \overset{0}{\rightsquigarrow} P \mathbin{|||} Q$.

Case $P \setminus A$

Assume $P \overset{0}{\rightsquigarrow} P$. Then by the rule for $\rightsquigarrow$ on hiding, it follows immediately that $P \setminus A \overset{0}{\rightsquigarrow} P \setminus A$.

□

Theorem 5.2: Time determinacy

$$P \overset{t}{\rightsquigarrow} P_1 \wedge P \overset{t}{\rightsquigarrow} P_2 \Rightarrow P_1 = P_2$$

Proof

Case $P \;{}_A\|_B\; Q$.

Assume inductively that the theorem holds for P and for Q. Now assume that $P \;{}_A\|_B\; Q \overset{t}{\rightsquigarrow} P_1$ and $P \;{}_A\|_B\; Q \overset{t}{\rightsquigarrow} P_2$. Then by minimality of $\rightsquigarrow$ under the rules, we may conclude that $P_1 = P' \;{}_A\|_B\; Q'$ and $P_2 = P'' \;{}_A\|_B\; Q''$, where $P \overset{t}{\rightsquigarrow} P'$ and $Q \overset{t}{\rightsquigarrow} Q'$, and $P \overset{t}{\rightsquigarrow} P''$ and $Q \overset{t}{\rightsquigarrow} Q''$. Then by the inductive hypothesis we have that $P' = P''$, and $Q' = Q''$, and hence that $P_1 = P_2$ as required.

Case $P \sqcap Q$

Assume inductively that the theorem holds for P and for Q. Now assume that $P \sqcap Q \overset{t}{\rightsquigarrow} P_1$ and $P \sqcap Q \overset{t}{\rightsquigarrow} P_2$. Then by minimality of $\rightsquigarrow$ under the closure of the rules, we may conclude that $P_1 = P' \sqcap Q'$, and that $P_2 = P'' \sqcap Q''$, and in addition, that $t = 0$, where $P \overset{t}{\rightsquigarrow} P_1$ and $Q \overset{t}{\rightsquigarrow} Q'$, and $P \overset{t}{\rightsquigarrow} P''$ and $Q \overset{t}{\rightsquigarrow} Q''$. Then by the inductive hypothesis we have that $P' = P''$, and $Q' = Q''$, and hence that $P_1 = P_2$ as required.

Case $P \setminus A$

Assume inductively that the theorem holds for P. Now assume that $P \setminus A \overset{t}{\rightsquigarrow} P_1$, and that $P \setminus A \overset{t}{\rightsquigarrow} P_2$. By minimality of $\rightsquigarrow$ under the closure of the rules, we may conclude that $P_1 = P' \setminus A$, and that $P_2 = P'' \setminus A$, and that $P \overset{t}{\rightsquigarrow} P'$, and $P \overset{t}{\rightsquigarrow} P''$, with either $t = 0$ or $A \cap init(P) = \{\}$. Then by the inductive hypothesis, we have that $P' = P''$, and so $P_1 = P_2$ as required.

□

Theorem 5.3: The set of possible initial events $init(P)$ remains constant under evolution.

$$P \overset{t}{\rightsquigarrow} P' \Rightarrow init(P) = init(P')$$

Proof

Case $WAIT\ t$

Assume $WAIT\ t \overset{u}{\rightsquigarrow} P_1$. By minimality of $\rightsquigarrow$, P_1 has the form $WAIT\ t - u$, and $u \leq t$. Then $init(WAIT\ t) = \{\} = init(P_1)$ as required.

Case $P \Box Q$

Assume true for P and for Q. Now consider $P \Box Q \overset{t}{\rightsquigarrow} P_1$. By the definition of $\rightsquigarrow$ we have that P_1 is of the form $P' \Box Q'$, where $P \overset{t}{\rightsquigarrow} P'$ and $Q \overset{t}{\rightsquigarrow} Q'$. Then $init(P) = init(P')$, and $init(Q) = init(Q')$, so $init(P \Box Q) = init(P' \Box Q')$ as required.

Case $P\,;\,Q$

Assume true for P (and for Q). Now consider $P\,;\,Q \overset{t}{\rightsquigarrow} P_1$. By the definition of $\rightsquigarrow$ we have that P_1 is of the form $P'\,;\,Q$, where $P \overset{t}{\rightsquigarrow} P'$, and either $t = 0$ or $\surd \notin init(P)$. Then $init(P) = init(P')$, and so $init(P\,;\,Q) = init(P'\,;\,Q)$ as required.

□

Theorem 5.4 Time continuity

$$P \overset{t''+t'}{\rightsquigarrow} P' \Leftrightarrow \exists\, P'' \bullet P \overset{t''}{\rightsquigarrow} P'' \wedge P'' \overset{t'}{\rightsquigarrow} P'$$

Proof

Case $P \sqcap Q$

Consider $P \sqcap Q \overset{t}{\rightsquigarrow} P_1$. Then by the definition of $\rightsquigarrow$ we have that $t = 0$. The result follows immediately.

Case $P \sqcap Q$

Assume the result for P and for Q, and consider $P \sqcap Q \overset{t}{\rightsquigarrow} P_1$ and $t'' \leq t$. Then $P_1 = P' \sqcap Q'$ for some P', Q' such that $P \overset{t}{\rightsquigarrow} P'$ and $Q \overset{t}{\rightsquigarrow} Q'$. By the inductive hypothesis, $\exists P'', Q'' \bullet P \overset{t''}{\rightsquigarrow} P'' \wedge P'' \overset{t-t''}{\rightsquigarrow} P' \wedge Q \overset{t''}{\rightsquigarrow} Q'' \wedge Q'' \overset{t-t''}{\rightsquigarrow} Q'$. Hence by the definition of $\rightsquigarrow$ we have that $P \sqcap Q \overset{t}{\rightsquigarrow} P'' \sqcap Q''$ and $P'' \sqcap Q'' \overset{t-t''}{\rightsquigarrow} P' \sqcap Q'$ as required.

Case $P \setminus A$

Assume the result for P, and consider $P \setminus A \overset{t}{\rightsquigarrow} P_1$ and $t'' \leq t$. Then P_1 has the form $P' \setminus A$, and we have either $t = 0$ or $A \cap \mathit{init}(P) = \{\}$. If $t = 0$ then the result follows immediately. If $A \cap \mathit{init}(P) = \{\}$, then we have $P \overset{t''}{\rightsquigarrow} P''$ and $P'' \overset{t-t''}{\rightsquigarrow} P'$ for some P'', and by Theorem 5.3 we have that $\mathit{init}(P'') = \mathit{init}(P)$. Therefore $A \cap \mathit{init}(P'') = \{\}$, and so $P \setminus A \overset{t''}{\rightsquigarrow} P'' \setminus A$ and $P'' \setminus A \overset{t-t''}{\rightsquigarrow} P' \setminus A$ as required.

$\Box$

Theorem 5.6: Maximal progress

$$P \xrightarrow{(t',\tau)} P' \wedge t > t' \Rightarrow \neg \exists P'' \bullet P \overset{t}{\rightsquigarrow} P''$$

Proof

Case $P \;{}_A\|_B\; Q$

Assume true for P and for Q. Now assume that $P \;{}_A\|_B\; Q \xrightarrow{(t',\tau)} P' \;{}_A\|_B\; Q'$. Then assume without loss of generality that $P \xrightarrow{(t',\tau)} P'$ and $Q \overset{t'}{\rightsquigarrow} Q'$. Now consider $t > t'$. If $P \;{}_A\|_B\; Q \overset{t}{\rightsquigarrow} P'' \;{}_A\|_B\; Q''$, then we have $P \overset{t}{\rightsquigarrow} P''$, which contradicts the assumption made of P.

Case $P \setminus A$

Assume true for P, also that $P \setminus A \xrightarrow{(t',\tau)} P' \setminus A$, and also that $t > t'$. Assume (for a contradiction) that $P \setminus A \overset{t}{\rightsquigarrow} P'' \setminus A$. Then either $P \xrightarrow{(t',\tau)} P'$ and $A \cap \mathit{init}(P) = \{\}$, or else $P \xrightarrow{(0,\mu)} P'$ for some $\mu \in A \cup \{\tau\}$. In the first case, we obtain that $P \overset{t}{\rightsquigarrow} P''$ which contradicts our assumption about P. In the second case, either $\mu = \tau$, in which case the result follows immediately; or else $\mu \in A$, in which case $A \cap \mathit{init}(P) \neq \{\}$, and so by the rule for $\rightsquigarrow$ and hiding, we obtain that $P \setminus A \overset{t}{\rightsquigarrow} P'' \setminus A \Rightarrow t = 0$, which contradicts the fact that $t > t' = 0$. In all cases a contradiction is obtained, and so the result follows.

Case $\mu X \circ P$

This has a τ action possible at time 0, and cannot evolve beyond time 0. Hence the result follows trivially.

□

Theorem 5.7: A process can always evolve up to the time of the next τ action

$$\begin{array}{rcl} P \overset{\tau}{\not\rightarrow} & \Rightarrow & \forall\, t\, \exists\, P' \bullet P \overset{t}{\rightsquigarrow} P' \\ P \xrightarrow{(t',\tau)} & \Rightarrow & \forall\, t \leq t'\, \exists\, P' \bullet P \overset{t}{\rightsquigarrow} P' \end{array}$$

Proof

Case $P \,\Box\, Q$

Assume true for P and for Q.

Consider first the case where $P \,\Box\, Q \overset{\tau}{\not\rightarrow}$. It follows that $P \overset{\tau}{\not\rightarrow}$ and $Q \overset{\tau}{\not\rightarrow}$. By the assumption for P and Q it follows that

$$\forall\, t \bullet \exists\, P', Q' \bullet P \overset{t}{\rightsquigarrow} P' \wedge Q \overset{t}{\rightsquigarrow} Q'$$

and hence that

$$\forall\, t \bullet \exists\, P_1 (= P' \,\Box\, Q') \bullet P \,\Box\, Q \overset{t}{\rightsquigarrow} P_1$$

as required.

Now consider $P \,\Box\, Q \xrightarrow{(t',\tau)} P' \,\Box\, Q'$. Without loss of generality we may assume $P \xrightarrow{(t',\tau)} P'$ and $Q \overset{t'}{\rightsquigarrow} Q'$. Then given $t \leq t'$ we have that $P \overset{t}{\rightsquigarrow} P'$ by the inductive hypothesis, and $Q \overset{t}{\rightsquigarrow} Q'$ by Theorem 5.4, and hence that $P \,\Box\, Q \overset{t}{\rightsquigarrow} P' \,\Box\, Q'$ as required.

Case $P \setminus A$

Assume true for P.

If $P \setminus A \overset{\tau}{\not\rightarrow}$ then we must have $P \overset{\tau}{\not\rightarrow}$, and also $P \overset{(0,\mu)}{\not\longrightarrow}$ for any $\mu \in A \cup \{\tau\}$, from which it follows (Theorem 5.5) that $A \cap \mathit{init}(P) = \{\}$. We therefore obtain by our hypothesis on P that $P \overset{t}{\rightsquigarrow} P'$, for any t, and hence that $P \setminus A \overset{t}{\rightsquigarrow} P' \setminus A$ as required.

If $P \setminus A \xrightarrow{(t',\tau)} P' \setminus A$ then from the transition rules for hiding we have that either $t' = 0$, which immediately yields the result, or else $t' > 0$ and $A \cap \mathit{init}(P) = \{\}$, and $P \xrightarrow{(t',\tau)} P'$. Given $t \leq t'$ we have that $P \overset{t}{\rightsquigarrow} P''$ for some P'', and hence that $P \setminus A \overset{t}{\rightsquigarrow} P'' \setminus A$ as required.

□

Theorem 5.9: Persistency

$$[P \xrightarrow{(0,a)} \wedge\, \exists\, P' \bullet P \overset{t}{\rightsquigarrow} P'] \Leftrightarrow P \xrightarrow{(t,a)}$$

Proof

Case $P \parallel\!\!\mid Q$

Assume $P \parallel\!\!\mid Q \xrightarrow{(t,a)}$. Then assume without loss of generality that $P \xrightarrow{(t,a)}$ and $Q \overset{t}{\rightsquigarrow} Q'$. By the assumption for P we have that $\exists P' \bullet P \overset{t}{\rightsquigarrow} P' \wedge P \xrightarrow{(0,a)}$. Hence $P \parallel\!\!\mid Q \xrightarrow{(0,a)}$ and $P \parallel\!\!\mid Q \overset{t}{\rightsquigarrow} P' \parallel\!\!\mid Q'$ as required.

Now assume that $P \parallel\!\!\mid Q \xrightarrow{(0,a)} \wedge \exists P' \parallel\!\!\mid Q' \bullet P \parallel\!\!\mid Q \overset{t}{\rightsquigarrow} P' \parallel\!\!\mid Q'$. Assume wlog that $P \xrightarrow{(0,a)}$. Then $P \overset{t}{\rightsquigarrow} P'$ and $Q \overset{t}{\rightsquigarrow} Q'$, so by the hypothesis for P we have that $P \xrightarrow{(t,a)}$, and hence that $P \parallel\!\!\mid Q \xrightarrow{(t,a)}$.

□

Theorem 5.10 Finite variability: if there is a sequence of processes $\{P_i\}_{i=1}^{\infty}$ and a sequence of transitions $\{(t_i, \mu_i)\}_{i=1}^{\infty}$ such that

$$\forall i \bullet P_i \xrightarrow{(t_i, a_i)} P_{i+1}$$

then $\Sigma_{i=1}^{\infty} t_i = \infty$.

Proof

We prove by structural induction on timed CSP terms P that $(P)\eta$ is unable to perform any infinite sequence of transitions in a finite time if binding η is such that $\eta(X)$ is unable to do so for any X.

The only case that is not straightforward is that of recursion. We first capture the notion of what it means for two processes to be equivalent up to time t, by introducing a form of bisimulation.

Definition A.1 Let R_t be a family of relations between processes, one for each $t \in \mathbf{R}^+$. Then the R_t are a time-bisimulation if

$$\begin{aligned}
(1)\quad & (P\ R_t\ Q \wedge P \xrightarrow{(u,\mu)} P' \wedge u < t) \\
& \Rightarrow \exists Q' \bullet Q \xrightarrow{(u,\mu)} Q' \wedge P'\ R_{t-u}\ Q' \\
(2)\quad & (P\ R_t\ Q \wedge Q \xrightarrow{(u,\mu)} Q' \wedge u < t) \\
& \Rightarrow \exists P' \bullet P \xrightarrow{(u,\mu)} P' \wedge P'\ R_{t-u}\ Q'
\end{aligned}$$

□

Each member R_t of a time-bisimulation is an equivalence relation. Define the family $\sim^t$ to be the strongest time-bisimulation. Observe that theorems 5.2, 5.3, ensure that (1) and (2) also hold when transitions are replaced by evolutions. Observe further that $P \sim^t Q \Rightarrow \mathit{init}(P) = \mathit{init}(Q)$ whenever $t > 0$. Note that if each R_t in (1) and (2) is replaced by a single R, then we obtain that R is a bisimulation relation in the sense of (Moller and Tofts 1990).

We will use the following two lemmas:

Lemma A.2 If P is t-guarded for X, η is a binding, and $Q \sim^u R$, then $(P[Q/X])\eta \sim^{u+t} (P[R/X])\eta$. $\Box$

This follows from the definition of t-guarded, by a routine induction on the structure of P.

Lemma A.3 If $P_0 \sim^t Q_0$, and there is an infinite sequence of transitions starting at P_0 whose cumulative time is less than t, then the same sequence is possible starting from Q_0. $\Box$

This lemma may be proved as follows: consider a pair of processes $P_0 \sim^t Q_0$. Assume that there is an infinite sequence of transitions $\{(t_i, \mu_i)\}_{i \in \mathbf{N}}$ and processes P_i such that $P_i \xrightarrow{(t_i,\mu_i)} P_{i+1}$ for each i, where $\Sigma t_i < t$. Then there must be a corresponding sequence of processes $\{Q_i\}_{i \in \mathbf{N}}$ such that $Q_i \xrightarrow{(t_i,\mu_i)} Q_{i+1}$ for each i, where $P_i \sim^{u_i} Q_i$ (for $u_i = (t - \Sigma_{j=1}^{i} t_j)$.)

We are now in a position to establish finite variability for the recursive case.

Case $(\mu\, X \circ P)\eta$

Since the term is well defined, we have that P is t-guarded for X, for some $t > 0$. Let η be a binding which maps term variables to processes which are unable to perform infinitely many transitions in a finite time. We assume (by our inductive hypothesis) that $(P)\eta$ is finitely variable. Observe that in this case $(\mathit{SKIP} \,;\, P)\eta$ is also finitely variable.

Assume for a contradiction that $(\mu\, X \circ P)\eta$ is not finitely variable. Then there is some time u before which $(\mu\, X \circ P)\eta$ may perform an infinite sequence of transitions.

Define the function F to mimic the unwinding of the recursive loop; the *SKIP* produces the $(0, \tau)$ corresponding to the transition marking the recursive call.

$$F(Q) \quad = \quad ((\mathit{SKIP} \,;\, P)[Q/X])\eta$$

Observe that $F((\mu\, X \circ P)\eta) \sim^{t'} (\mu\, X \circ P)\eta$, for any t'. Also, since P is t-guarded for X, so is $\mathit{SKIP} \,; P$. We first prove that $F^n(STOP)$ is finitely variable, for any n. It is trivially true for 0. If true for n, then

$$\begin{aligned} F^{n+1}(STOP) &= F(F^n(STOP)) \\ &= ((\mathit{SKIP} \,;\, P)[F^n(STOP)/X])\eta \\ &= (\mathit{SKIP} \,;\, P)\eta[F^n(STOP)/X] \end{aligned}$$

which is finitely variable by our hypothesis for P, which entails that $(\mathit{SKIP}\,; P)\eta$ is finitely variable whenever η maps term variables to finitely variable processes; in particular, this is the case for the binding is $\eta[F^n(\mathit{STOP})/X]$.

Finally, we establish that $F^n(\mathit{STOP}) \sim^{nt} ((\mu\, X \circ P)\eta)$. This is trivial for 0, since all processes are $\sim^{0}$ equivalent. If it is true for n, then by Lemma A.2 (and since $\mathit{SKIP}\,;P$ is t-guarded for X)

$$(\mathit{SKIP}\,; P[(\mu\, X \circ P)\eta/X])\eta \quad \sim^{nt+t} \quad (\mathit{SKIP}\,; P[F^n(\mathit{STOP})/X])\eta$$

or alternatively

$$F((\mu\, X \circ P)\eta) \quad \sim^{(n+1)t} \quad F^{n+1}(\mathit{STOP})$$

Also $F((\mu\, X \circ P)\eta) \sim^{t'} (\mu\, X \circ P)\eta$ for any t' so the result holds for $n+1$ by transitivity of $\sim^{t'}$. But then there is some n such that $nt \geq u$, and for such n we obtain

$$(\mu\, X \circ P)\eta \quad \sim^{u} \quad F^n(\mathit{STOP})$$

Since $F^n(\mathit{STOP})$ is finitely variable, it follows from Lemma A.3 that there can be no infinite sequence of transitions starting at $(\mu\, X \circ P)\eta$ whose total time is less than u, contradicting our earlier assumption and thus establishing the case.

$\square$

Theorem 6.4 If V is a set of variables, and η is a partial-binding that acts as the identity on V and maps variables not in V to processes, then all the free variables of $(P)\eta$ are contained in V.

Proof

Case X

If $X \in V$ then $(X)\eta = X$, and its only free variable is X, as required. If $X \notin V$, then $(X)\eta$ is a process, which has no free variables, yielding the result.

Case $\mu\, X \circ P$

The term $(\mu\, X \circ P)\eta$ is equal to $\mu\, X \circ (P)\eta_{[X \to X]}$. The set on which $\eta_{[X \to X]}$ is the identity is $V \cup \{X\}$, so by the inductive hypothesis, the free variables of $(P)\eta_{[X \to X]}$ are a subset of $V \cup \{X\}$, and therefore so are the free variables of $\mu\, X \circ (P)\eta_{[X \to X]}$. However, X is not free in $\mu\, X \circ (P)\eta_{[X \to X]}$, so its free variables are a subset of V, as required.

The other cases are trivial.

$\square$

Equivalence

Note that the axioms and semantic equations for the model $\mathcal{M}_{TF}$ are given in Appendix B

Theorem 6.7 For all timed CSP terms P and bindings η we have $\Phi((P)\eta) = \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$.

Proof Example cases

Case $STOP$

Now $STOP \not\rightarrow$, and so we have immediately from the definition that

$$\begin{aligned}\Phi((STOP)\eta) &= \{(w \setminus \tau, \aleph) \mid w = \langle\rangle\}) \\ &= \mathcal{F}_{TF}[\![STOP]\!]\overline{\eta}\end{aligned}$$

Case X

$$\begin{aligned}\Phi((X)\eta) &= \Phi(\eta(X)) \\ &= \overline{\eta}[X] \\ &= \mathcal{F}_{TF}[\![X]\!]\overline{\eta}\end{aligned}$$

Case $P \sqcap Q$

Consider $(s, \aleph) \in \Phi((P \sqcap Q)\eta)$. This is the case if and only if $(P \sqcap Q)\eta$ $\underline{\text{fail}}$ $(w \setminus \tau, \aleph)$ for some $w \setminus \tau = s$. Now either $(w, \aleph) = (\langle\rangle, \{\})$, or else $(P \sqcap Q)\eta \xrightarrow{(0,\tau)} P_1$, and P_1 $\underline{\text{fail}}$ $(w', \aleph)$, where $w = \langle(0, \tau)\rangle^\frown w'$. Now we have either $P_1 = (P)\eta$ or $P_1 = (Q)\eta$, and so $(s, \aleph) \in \Phi((P)\eta) \cup \Phi((Q)\eta)$, which by the inductive hypothesis is equivalent to $(s, \aleph) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta} \cup \mathcal{F}_{TF}[\![Q]\!]\overline{\eta} = \mathcal{F}_{TF}[\![P \sqcap Q]\!]\overline{\eta}$, which establishes the case.

Conversely, consider $(s, \aleph) \in \mathcal{F}_{TF}[\![P \sqcap Q]\!]\overline{\eta}$. Then either $(s, \aleph) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$ or $(s, \aleph) \in \mathcal{F}_{TF}[\![Q]\!]\overline{\eta}$. Assume without loss of generality that the first disjunct holds. Then $(s, \aleph) \in \Phi((P)\eta)$, and so for some $w \setminus \tau = s$ we have $(P)\eta$ $\underline{\text{fail}}$ $(w, \aleph)$. Now $(P \sqcap Q)\eta \xrightarrow{(0,\tau)} (P)\eta$, and so $(P \sqcap Q)\eta$ $\underline{\text{fail}}$ $(\langle(0, \tau)\rangle^\frown w, \aleph)$, and hence $(s, \aleph) \in \Phi((P \sqcap Q)\eta)$, as required.

Case $P \;{}_A\|_B\; Q$

We prove that each of $\Phi((P \;{}_A\|_B\; Q)\eta)$ and $\mathcal{F}_{TF}[\![P \;{}_A\|_B\; Q]\!]\overline{\eta}$ is a subset of the other.

Consider $(s, \aleph) \in \Phi((P \;{}_A\|_B\; Q)\eta)$. Then $(P \;{}_A\|_B\; Q)\eta$ $\underline{\text{fail}}$ $(w, \aleph)$ for some $w \setminus \tau = s$. Let $w = \langle(t_i, \mu_i) \mid 0 \le i \le n-1\rangle$. There is a sequence $\langle P_i \;{}_A\|_B\; Q_i\rangle$ with $P_0 = P$ and $Q_0 = Q$, such that $P_i \;{}_A\|_B\; Q_i \xrightarrow{(t_i - t_{i-1}, \mu_i)} P_{i+1} \;{}_A\|_B\; Q_{i+1}$ for $0 \le i \le n-1$. Now for each i with $0 \le i \le n-1$ we have that μ_i was performed either only by P, or only by Q, or by a synchronisation of both P and Q. We may thus derive the contribution made by each of P and Q in each of the following cases, given by the κ_i and ν_i respectively. We use $\varepsilon \notin \Sigma$ as a new symbol.

1. $\mu_i \in A \cap B$. In this case define $\kappa_i = \mu_i$ and $\nu_i = \mu_i$.

2. $\mu_i \in (A \setminus B) \vee (\mu_i = \tau \wedge P_i \xrightarrow{(t_i - t_{i-1}, \tau)} P_{i+1})$. In this case, define $\kappa_i = \mu_i$ and $\nu_i = \varepsilon$.

3. $\mu_i \in (B \setminus A) \vee (\mu_i = \tau \wedge Q_i \xrightarrow{(t_i - t_{i-1}, \tau)} Q_{i+1})$. In this case, define $\kappa_i = \varepsilon$ and $\nu_i = \mu_i$

We also define the following traces and refusal sets (taking $t_n = \infty$ for convenience), which correspond to the traces and refusal sets of P and Q respectively which combine under parallel to give $(s, \aleph)$.

$$\begin{aligned}
s_P &= \langle \kappa_i \mid \kappa_i \notin \{\tau, \varepsilon\} \rangle \\
s_Q &= \langle \nu_i \mid \nu_i \notin \{\tau, \varepsilon\} \rangle \\
\aleph_P &= \bigcup_{i=0}^{n} (\aleph \cap [t_i, t_{i+1}) \times \{a \mid P_i \overset{a}{\not\rightarrow}\}) \\
\aleph_Q &= \bigcup_{i=0}^{n} (\aleph \cap [t_i, t_{i+1}) \times \{a \mid Q_i \overset{a}{\not\rightarrow}\})
\end{aligned}$$

The following results are immediate:

$$\begin{aligned}
w \setminus \tau &\in s_P \;{}_A\|_B\; s_Q \\
\aleph &= \aleph_P \cup \aleph_Q
\end{aligned}$$

and also that $(P)\eta$ $\underline{\text{fail}}$ $(\langle (t_i, \kappa_i) \rangle \setminus \varepsilon, \aleph_P)$; hence $(s_P, \aleph_P) \in \Phi((P)\eta)$, and so $(s_P, \aleph_P) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$. By an entirely similar line of reasoning we obtain $(s_Q, \aleph_Q) \in \mathcal{F}_{TF}[\![Q]\!]\overline{\eta}$.

It therefore follows by the semantic equation for ${}_A\|_B$ that $(w \setminus \tau, \aleph) \in \mathcal{F}_{TF}[\![P \;{}_A\|_B\; Q]\!]\overline{\eta}$. Since $w \setminus \tau = s$ we have that $(s, \aleph) \in \mathcal{F}_{TF}[\![P \;{}_A\|_B\; Q]\!]$ as required.

To prove the result in the other direction, an auxiliary lemma will be useful:

Lemma A.4 Any execution of a parallel combination can be decomposed into executions of its component processes:

$$\begin{aligned}
&(P \;{}_A\|_B\; Q)\ \underline{\text{fail}}\ (w, \aleph) \Leftrightarrow \\
&\quad \exists\, w_1, w_2, \aleph_1, \aleph_2, P', P'', Q', Q'', t_1, t_2 \bullet \\
&\quad (P, P')\ \underline{\text{fail}}\ (w_1, \aleph_1) \wedge P' \overset{t_1}{\rightsquigarrow} P'' \wedge \\
&\quad (Q, Q')\ \underline{\text{fail}}\ (w_2, \aleph_2) \wedge Q' \overset{t_2}{\rightsquigarrow} Q'' \wedge \\
&\quad \mathit{end}(w_1) + t_1 = \mathit{end}(w_2) + t_2 \geq \mathit{end}(\aleph) \wedge \\
&\quad w_1 \upharpoonright A = w \upharpoonright A \wedge w_2 \upharpoonright B = w \upharpoonright B \wedge w = w \upharpoonright (A \cup B \cup \{\tau\}) \wedge \\
&\quad \forall\, t \bullet (\#(w \uparrow t \upharpoonright \tau) = \#(w_1 \uparrow t \upharpoonright \tau) + \#(w_2 \uparrow t \upharpoonright \tau))
\end{aligned}$$

□

Proof The forward direction may be proved easily by induction on the length of the execution sequence w.

Case $\#w = 0$ Then $P \;{}_A\|_B\; Q \overset{end(\aleph)}{\rightsquigarrow} P'' \;{}_A\|_B\; Q''$. Defining $\aleph_P = \aleph \upharpoonright (A \setminus init(P))$, $\aleph_Q = \aleph \upharpoonright (B \setminus init(Q))$, $t_1 = t_2 = end(\aleph)$, $w_1 = w_2 = \langle\rangle$, $P' = P$, and $Q' = Q$, we obtain the right hand side.

Case $\#w = n + 1$ Then $w = \langle(t,\mu)\rangle^\frown w'$. If $\mu \in A \cap B$, then $P \xrightarrow{(t,\mu)} P'''$ and $Q \xrightarrow{(t,\mu)} Q'''$, with $(w' - t, \aleph - t) \in \Phi'(P''' \;{}_A\|_B\; Q''')$. It is then straightforward to use the inductive hypothesis for $w' - t$ to derive the result for w. The possible cases for P performing μ independently of Q are $\mu \in A \setminus B$ and $\mu = \tau \wedge P \xrightarrow{(t,\tau)} P' \wedge Q \overset{t}{\rightsquigarrow} Q'$. The cases for Q performing μ independently of P are symmetrical to these. In all cases, the inductive step is easy to establish.

The proof from right to left may be established by a straightforward induction on the sum of the lengths of w_1 and w_2. Assume that $w_1, w_2, \aleph_1, \aleph_2, P', P'', Q', Q'', t_1, t_2$ are instantiated so that the right hand side is seen to hold.

Case $\#w_1 + \#w_2 = 0$ Then clearly $w = \langle\rangle$, $P \overset{t_1}{\rightsquigarrow} P''$ and $Q \overset{t_2}{\rightsquigarrow} Q''$, and $t_1 = t_2 \geq end(\aleph)$. Also $init(P) \cap \sigma(\aleph_P) = \{\}$, and $init(Q) \cap \sigma(\aleph_Q) = \{\}$. Hence $init(P \;{}_A\|_B\; Q) \cap \sigma(\aleph) = \{\}$, and so $P \;{}_A\|_B\; Q$ $\underline{\text{fail}}$ $(\langle\rangle, \aleph)$ as required.

Case $\#w_1 + \#w_2 = n + 1$ We will consider the case that P performs the first transition without Q. The cases where Q performs it, and where they perform it together, are entirely similar.

Assume that $w_1 = \langle(t,\mu)\rangle^\frown w_1' \wedge begin(w_2) \geq t \wedge \mu \in \{\tau\} \cup A \setminus B$. Then there are P''' and Q''' with

$$\begin{aligned} &P \xrightarrow{(t,\mu)} P''' \wedge (P''', P')\ \underline{\text{fail}}\ (w_1' - t, \aleph_1 - t) \wedge P' \overset{t_1}{\rightsquigarrow} P'' \wedge \\ &Q \overset{t}{\rightsquigarrow} Q''' \wedge (Q''', Q')\ \underline{\text{fail}}\ (w_2 - t, \aleph_2 - t) \wedge Q' \overset{t_2}{\rightsquigarrow} Q'' \end{aligned}$$

and so by the inductive hypothesis we obtain $P''' \;{}_A\|_B\; Q'''$ $\underline{\text{fail}}$ $(w', \aleph - t)$. Now $P \;{}_A\|_B\; Q \xrightarrow{(t,\mu)} P''' \;{}_A\|_B\; Q'''$, and $init(P \;{}_A\|_B\; Q) \cap \sigma(\aleph \between t) = \{\}$, so $P \;{}_A\|_B\; Q$ $\underline{\text{fail}}$ $(w', \aleph)$ as required. $\Box$

Case $P \setminus A$

We prove that each of $\Phi((P \setminus A)\eta)$ and $\mathcal{F}_{TF}[\![P \setminus A]\!]\overline{\eta}$ is a subset of the other.

Consider $(s, \aleph) \in \Phi((P \setminus A)\eta)$. Then $(P \setminus A)\eta$ $\underline{\text{fail}}$ $(w, \aleph)$ for some $w \setminus \tau = s$. Therefore $(P)\eta$ $\underline{\text{fail}}$ $(w', \aleph \cup [0, end(w', \aleph)) \times A)$ for some w' with $w'[\tau/A] = w$. Then $(w' \setminus \tau, \aleph \cup [0, end(w', \aleph)) \times A) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$, and so $(w' \setminus A \cup \{\tau\}, \aleph) \in \mathcal{F}_{TF}[\![P \setminus A]\!]\overline{\eta}$. But $w' \setminus A \cup \{\tau\} = w \setminus \tau = s$, and so $(s, \aleph) \in \mathcal{F}_{TF}[\![P \setminus A]\!]\overline{\eta}$.

Now consider $(s, \aleph) \in \mathcal{F}_{TF}[\![P \setminus A]\!]\overline{\eta}$. Then $\exists\, s' \bullet s' \setminus A = s \wedge (s', \aleph \cup [0, end(s', \aleph)) \times A) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$. Hence $(P)\eta$ $\underline{\text{fail}}$ $(w', \aleph \cup [0, end(s', \aleph)) \times A)$ for some $w' \setminus \tau = s'$. Therefore $(P \setminus A)\eta$ $\underline{\text{fail}}$ $(w'[\tau/A], \aleph)$. Hence $(s' \setminus A, \aleph) \in \Phi((P \setminus A)\eta)$ as required.

Case $f^{-1}(P)$

Consider $(s, \aleph) \in \Phi((f^{-1}(P))\eta)$. This is true if and only if $(f^{-1}(P))\eta$ $\underline{\text{fail}}$ $(w, \aleph)$ for some $w \setminus \tau = s$, which is equivalent to $(P)\eta$ $\underline{\text{fail}}$ $(f(w), f(\aleph))$, which holds if and only if $(f(s), f(\aleph)) \in \Phi((P)\eta)$ (since $f(w) \setminus \tau = f(s)$), which by our inductive hypothesis is equivalent to $(f(s), f(\aleph)) \in \mathcal{F}_{TF}[\![P]\!]\overline{\eta}$, which is equivalent to $(s, \aleph) \in \mathcal{F}_{TF}[\![f^{-1}(P)]\!]\overline{\eta}$, as required.

Case $\mu X \circ P$

$$\begin{aligned} \Phi((\mu X \circ P)\eta) &= \Phi((P[\mu X \circ P / X])\eta) \\ &= \mathcal{F}_{TF}[\![P]\!]\overline{\eta}[\Phi((\mu X \circ P)\eta)/X] \end{aligned}$$

Hence $\Phi((\mu X \circ P)\eta)$ is the *unique* fixed point of the contraction mapping on $\mathbf{P}(TF)$

$$\lambda S \bullet \mathcal{F}_{TF}[\![P]\!]\overline{\eta}[S/X]$$

and is therefore equal to $\mathcal{F}_{TF}[\![\, \mu X \circ P]\!]\overline{\eta}$. $\Box$

B The timed failures model $\mathcal{M}_{TF}$

Behaviours in the timed failures model are pairs $(s, \aleph)$, which consist of a timed trace s, and a timed refusal set $\aleph$. A timed trace is a finite sequence of timed events drawn from $\mathbf{R}^+ \times \mathbf{\Sigma}$, such that the times are in non-decreasing order. A timed refusal set is a set of timed events made up of a finite union of refusal tokens; a refusal token is the cartesian product of a finite half open interval and a set of events, drawn from the set

$$\{[b, e) \mid 0 \leq b < e < \infty\} \times \mathbf{P}(\Sigma)$$

The set of timed traces is denoted TT; the set of refusal sets is denoted $RSET$; and the set of all trace-refusal pairs is denoted TF.

B.1 The Timed Failures Model $\mathcal{M}_{TF}$

The Timed Failures Model $\mathcal{M}_{TF}$ is defined to be those elements $S \in \mathbf{P}(TF)$ which satisfy the following axioms:

1. $(\langle\rangle, \{\}) \in S$
2. $(s \frown w, \aleph) \in S \Rightarrow (s, \aleph \upharpoonright begin(w)) \in S$
3. $\forall \aleph' : RSET \bullet (s, \aleph) \in S \wedge \aleph' \subseteq \aleph \Rightarrow (s, \aleph') \in S$
4. $\forall t : [0, \infty) \bullet \exists n : \mathbf{N} \bullet \mathbf{(s, \aleph) \in S \wedge end(s) \leq t \Rightarrow \#(s) \leq n}$

5. $(s, \aleph) \in S \wedge u > 0 \Rightarrow$

$$\begin{array}{l} \exists \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s, \aleph') \in S \wedge \\ \qquad \forall (t, a) \in [0, u) \times \Sigma \bullet (t, a) \notin \aleph' \Rightarrow (s \lhd t ^\frown \langle (t, a) \rangle, \aleph' \wr t) \in S \\ \qquad\qquad \wedge \\ \qquad\qquad ((t > 0 \wedge \nexists\, \epsilon > 0 \bullet ((t - \epsilon, t) \times \{a\} \subseteq \aleph')) \Rightarrow \\ \qquad\qquad\quad (s \wr t ^\frown \langle (t, a) \rangle, \aleph' \wr t) \in S) \end{array}$$

The distance function d is defined on elements of $\mathcal{M}_{TF}$ as follows.

$$\begin{array}{rcl} S \wr t & \widehat{=} & \{(s, \aleph) \in S \mid end(s, \aleph) < t\} \\ d(S, T) & = & inf\{2^{-t} \mid S \wr t = T \wr t\} \end{array}$$

This distance function induces a complete metric on $\mathcal{M}_{TF}$.

B.2 The Semantic Function $\mathcal{F}_{\mathcal{T}}$

The set *BIN* is the set of semantic bindings, functions $\rho : \mathcal{V} \longrightarrow \mathcal{M}_{TF}$. We provide a semantic function for $TCSP_0$ terms.

The semantic function

$$\mathcal{F}_{TF} : TCSP_0 \longrightarrow BIN \longrightarrow \mathcal{M}_{TF}$$

is defined by the following set of equations. Observe that if the term P is has no free process variables, then the value of $\mathcal{F}_{TF}[\![P]\!]\rho$ is independent of the choice of ρ. In this case the constant function $\mathcal{F}_{TF}[\![P]\!]$ is often identified with the value $\mathcal{F}_{TF}[\![P]\!]\rho$ (for some arbitrary ρ).

$$\begin{array}{rcl} \mathcal{F}_{TF}[\![STOP]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid \aleph \in RSET\} \\ \mathcal{F}_{TF}[\![SKIP]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid \surd \notin \sigma(\aleph)\} \\ & & \cup \\ & & \{(\langle(t, \surd)\rangle, \aleph) \mid t \geq 0 \wedge \surd \notin \sigma(\aleph \uparrow [0, t))\} \\ \mathcal{F}_{TF}[\![WAIT\ t]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid \surd \notin \sigma(\aleph \rhd t)\} \\ & & \cup \\ & & \{(\langle(u, \surd)\rangle, \aleph) \mid u \geq t \wedge \surd \notin \sigma(\aleph \uparrow [t, u))\} \\ \mathcal{F}_{TF}[\![a \longrightarrow P]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph) \mid a \notin \sigma(\aleph)\} \\ & & \cup \\ & & \{(\langle(t, a)\rangle ^\frown s, \aleph) \mid \begin{array}[t]{l} t \geq 0 \wedge \\ a \notin \sigma(\aleph \wr t) \wedge \\ begin(s) \geq t \wedge \\ (s, \aleph) - t \in \mathcal{F}_{TF}[\![P]\!]\rho\} \end{array} \end{array}$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![P\ ;\ Q]\!]\rho & \mathrel{\widehat=} & \{(s,\aleph) \mid \surd \notin \sigma(s)\ \wedge \\
& & \qquad (s, \aleph \cup ([0, end(s,\aleph)) \times \{\surd\})) \in \mathcal{F}_{TF}[\![P]\!]\rho \\
& & \quad \vee \\
& & \quad s = s_P{}^\frown s_Q \wedge \surd \notin \sigma(s_P) \wedge \\
& & \qquad (s_Q, \aleph) - t \in \mathcal{F}_{TF}[\![Q]\!]\rho\ \wedge \\
& & \qquad (s_P{}^\frown\langle(t,\surd)\rangle, \aleph \upharpoonright t \cup ([0,t) \times \{\surd\})) \in \mathcal{F}_{TF}[\![P]\!]\rho\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![P \overset{t}{\rhd} Q]\!]\rho & \mathrel{\widehat=} & \{(s,\aleph) \mid begin(s) \le t \wedge (s,\aleph) \in \mathcal{F}_{TF}[\![P]\!]\rho\} \\
& & \cup \\
& & \{(s,\aleph) \mid begin(s) \ge t \wedge (\langle\rangle, \aleph \upharpoonright t) \in \mathcal{F}_{TF}[\![P]\!]\rho \\
& & \qquad \wedge \\
& & \qquad (s,\aleph) - t \in \mathcal{F}_{TF}[\![Q]\!]\rho\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![P \Box Q]\!]\rho & \mathrel{\widehat=} & \{(\langle\rangle,\aleph) \mid (\langle\rangle,\aleph) \in \mathcal{F}_{TF}[\![P]\!]\rho \cap \mathcal{F}_{TF}[\![Q]\!]\rho\} \\
& & \cup \\
& & \{(s,\aleph) \mid s \ne \langle\rangle \wedge (s,\aleph) \in \mathcal{F}_{TF}[\![P]\!]\rho \cup \mathcal{F}_{TF}[\![Q]\!]\rho \\
& & \qquad \wedge \\
& & \qquad (\langle\rangle, \aleph \upharpoonright begin(s)) \in \mathcal{F}_{TF}[\![P]\!]\rho \cap \mathcal{F}_{TF}[\![Q]\!]\rho\}
\end{array}
$$

$$
\mathcal{F}_{TF}[\![P \sqcap Q]\!]\rho \mathrel{\widehat=} \mathcal{F}_{TF}[\![P]\!]\rho \cup \mathcal{F}_{TF}[\![Q]\!]\rho
$$

$$
\mathcal{F}_{TF}[\![\textstyle\bigsqcap_{i\in I} P_i]\!]\rho \mathrel{\widehat=} \bigcup_{i\in I} \mathcal{F}_{TF}[\![P_i]\!]\rho
$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![a : A \longrightarrow P_a]\!]\rho & \mathrel{\widehat=} & \{(\langle\rangle,\aleph) \mid A \cap \sigma(\aleph) = \{\}\} \\
& & \cup \\
& & \{(\langle(t,a)\rangle^\frown s, \aleph) \mid\ a \in A \wedge t \ge 0\ \wedge \\
& & \qquad \sigma(\aleph \upharpoonright t) \cap A = \{\}\ \wedge \\
& & \qquad begin(s) \ge t\ \wedge \\
& & \qquad (s,\aleph) - t \in \mathcal{F}_{TF}[\![P_a]\!]\rho\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![P\ {}_A\|_B\ Q]\!]\rho & \mathrel{\widehat=} & \{(s, \aleph_P \cup \aleph_Q \cup \aleph_R) \mid \exists s_P, s_Q \bullet \\
& & \qquad \sigma(\aleph_P) \subseteq A \wedge \sigma(\aleph_Q) \subseteq B\ \wedge \\
& & \qquad \sigma(\aleph_R) \subseteq \Sigma - (A \cup B) \wedge s \in (s_P\ {}_A\|_B\ s_Q)\ \wedge \\
& & \qquad (s_P, \aleph_P) \in \mathcal{F}_{TF}[\![P]\!]\rho \wedge (s_Q, \aleph_Q) \in \mathcal{F}_{TF}[\![Q]\!]\rho\ \}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{F}_{TF}[\![P \parallel Q]\!]\rho & \mathrel{\widehat=} & \{(s,\aleph) \mid \exists s_P, s_Q \bullet\ s \in s_P \parallel s_Q\ \wedge \\
& & \qquad (s_P, \aleph) \in \mathcal{F}_{TF}[\![P]\!]\rho\ \wedge \\
& & \qquad (s_Q, \aleph) \in \mathcal{F}_{TF}[\![Q]\!]\rho\}
\end{array}
$$

$$
\mathcal{F}_{TF}[\![P \setminus A]\!]\rho \mathrel{\widehat=} \{(s \setminus A, \aleph) \mid (s, \aleph \cup ([0, end(s,\aleph) \times A) \in \mathcal{F}_{TF}[\![P]\!]\rho\}
$$

$$
\mathcal{F}_{TF}[\![f(P)]\!]\rho \mathrel{\widehat=} \{(f(s), \aleph) \mid (s, f^{-1}(\aleph)) \in \mathcal{F}_{TF}[\![P]\!]\rho\}
$$

$$
\mathcal{F}_{TF}[\![f^{-1}(P)]\!]\rho \mathrel{\widehat=} \{(s,\aleph) \mid (f(s), f(\aleph)) \in \mathcal{F}_{TF}[\![P]\!]\rho\}
$$

$$
\mathcal{F}_{TF}[\![X]\!]\rho \mathrel{\widehat=} \rho[\![X]\!]
$$

$$\mathcal{F}_{TF}[\![\mu\, X \circ P]\!]\rho \;\widehat{=}\; \text{the unique fixed point of the contraction mapping } \lambda\, S \bullet \mathcal{F}_{TF}[\![P]\!]\rho[S/X] \text{ on the complete metric space } \mathcal{M}_{TF}$$

where the auxiliary functions on timed traces are defined as follows:

$$\begin{aligned} s_P\ {}_A\|_B\ s_Q &\;\widehat{=}\; \{s \in TT \mid s \upharpoonright A = s_P \wedge s \upharpoonright B = s_Q \wedge s \upharpoonright (A \cup B) = s\} \\ s_P \ |\!|\!|\ s_Q &\;\widehat{=}\; \{s : TT \mid \forall\, t : \mathbf{R}^+ \bullet \forall\, \mathbf{a} : \mathbf{\Sigma} \bullet \\ &\qquad\qquad s \uparrow t \upharpoonright \{a\} = s_P \uparrow t \upharpoonright \{a\}^\frown s_Q \uparrow t \upharpoonright \{a\}\} \end{aligned}$$

In order to guarantee the well-definedness of taking a prefix choice or nondeterministic choice over a set of processes, we require that the set is *uniformly bounded* to ensure that the result meets Axiom (4) above; a set of terms $\{P_i \mid i \in I\}$ is *uniformly bounded* if there is a function $n : \mathbf{R}^+ \longrightarrow \mathbf{N}$ such that

$$\forall\, i \in I, t \in \mathbf{R}^+ \bullet (\mathbf{s}, \aleph) \in \mathcal{F}_{\mathbf{TF}}[\![\mathbf{P_i}]\!]\rho_{\mathbf{0}} \wedge \mathbf{end}(\mathbf{s}) \leq \mathbf{t} \Rightarrow \#\mathbf{s} \leq \mathbf{n}(\mathbf{t})$$

where ρ_0 is the constant binding that maps every term variable to $\mathcal{F}_{TF}[\![STOP]\!]$.