CORRECTNESS AND COMMUNICATION IN REAL-TIME SYSTEMS

by

Steve Schneider

Technical Monograph PRG-84
ISBN 0-902928-63-5

March 1990

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

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

Electronic mail: `sas@uk.ac.oxford.prg` (JANET)

Correctness and Communication in Real-Time Systems

Steven A. Schneider
Balliol College
Oxford

Snbmitted for the degree of Doctor of Philosophy, Michaelmas 1989

Abstract

This thesis builds upon the mathematical theory for real-time distributed computing developed by Reed and Roscoe. Time-critical process constructors for modelling timeonts, interrupts, and communication constructs, are defined in terms of the primitive operators of Timed Communicating Sequential Processes (*TCSP*). The work on communication involves the modelling of channels, inputs, outputs, chaining, and a cbaracterisation and analysis of buffers. These tools are applied to the specification, construction, and verification of communication protocols. The methods are generalised to apply to networks.

Real-time systems are inherently complex, and this is reflected in the complexity of the verification process. This thesis presents three useful verification methods. Tbe first is a complete compositional proof system for behavioural specifications on *TCSP*. The second involves the definition of generic specifications on processes, capturing those properties of component processes which combine readily when constructing large networks. Methods of constructing and of identifying processes meeting such specifications are examined, and laws are formulated concerning tbeir interaction.

The third approach exploits the mappings between different models within Reed's hierarchy. A notion of timewise refinement is presented which allows simple processes to be refined by the introduction of timing considerations. Properties which are preserved by timewise refinement are important, since there already exist well-established techniques for proving that such properties hold of processes in models lower in the hierarchy.

Acknowledgments

I would like to thank my supervisor, Mike Reed, for his encouragement, advice, and guidance during the course of this thesis. Thanks are also due to Bill Roscoe for his invaluable comments and suggestions concerning this work. I am grateful for the comments provided by Jim Davies, whose concurrent approach to the subject has resulted in many fruitful discussions and collaborative results. This work has also benefitted from conversations with and advice from Michael Goldsmith, Dave Jackson, Stephen Blamey, Steve Brookes, Alan Jeffrey, Karen Paliwoda, Geraint Jones, and many other colleagues in the PRG, and on the ESPRIT BRA-SPEC project.

This thesis is dedicated to Elizabeth, for sharing the strain of writing up. Thanks are also due to Mum, Dad, Chris, and Joy, for their love and support.

I am grateful to SERC for financial support during the course of this research.

Contents

1 Introduction

The theory of Communicating Sequential Processes (*CSP*) [Hoa85] has matured into a complete methodology for the analysis of concurrent systems. It provides a complete framework for systems design, from the capture of liveness and safety specifications, through the development of *CSP* programs, to the verification of systems. It supports program refinement, and provides the theoretical foundation for the occam languages. General methods concerning many important aspects of concurrency (for example, deadlock-freedom) have been developed.

The inclusion of real-time considerations in the theory of *CSP* was first attempted in [Jon82]. However, technical difficulties made the treatment difficult to apply. The introduction of more successful real-time models ([RR86, RR87, Ree88]) for *CSP* is relatively recent. Although such models enable the rigorous analysis of processes whose description involves timing considerations, the development of general techniques for such analysis is still in its infancy. Reed provides a theoretical basis for the specification and verification of timed processes in [Ree88], but any application depends upon manipulations of the axioms of the semantic model TM_{FS}, and this makes proofs extremely arduous. We need to develop general methods for reasoning about time-critical and time-dependent behaviour, in the way that the high level theory for *CSP* rests upon the semantic models for the language.

Any effective model for real-time distributed computing should enable the mathematical definition of temporal concepts such as timeouts, interrupts and priority, and should support the definition of new process constructors (for example, a timeout operator). An adequate treatment of communication is also required in order to analyse processes whose characterisation involves the use of communication channels, such as buffers, protocols, and pipes.

The motivation for providing a formal semantics of a computer language is to allow "programmers to make rigorous statements about the behaviour of the programs they write" [Sto77]. Our primary concern is to use the timed models to express time-dependent specifications, and to prove that these are satisfied by candidate processes. There are a number of fruitful approaches we can take. In this thesis, we examine three possibilities: the use of the compositional nature of the semantics to break down proof obligations to manageable proportions; the creation of a library of common specifications, sucb as 'the process will respond within t seconds', and developing techniques for recognising and constructing processes meeting them; the exploitation of the hierarchy of models defined in [Ree88] so that verifications in simple models remain valid further up the hierarchy. We will see that each approach insulates us from much of the detail present in thesemantic models.

The thesis is organised as follows. In chapter 2, we introduce some new notation for reasoning about timed behaviour. The following chapter introduces new process constructors, including operators for time-critical constructs. In chapter 4, we present a complete proof system for the timed failures model (TM_F) of Timed CSP (*TCSP*), and consider how it can be extended to the timed failures stability model. Chapter 5 introduces a series of families of specifications, and examines the relationships between them. In chapter 6, we define two forms of timewise refinement, and consider their use in verification. Chapter 7 defines several concepts for communication, and analyses the behaviour of processes that involve their application. Chapter 8 illustrates the methods and constructs of the previous chapters with examples. The final chapter contains a comparison of our results with existing work in the literature, and a discussion of future research directions.

Our aims in this thesis are to ease the high level specification of real-time systems. to facilitate the construction of *TCSP* programs, and to provide techniques for verification. In particular, we consider these techniques in the context of new constructs for both time-critical behaviour and point-to-point communication. In this way, we lay the foundation for a powerful specification and development methodology for real-time concurrent systems.

2 Notation

We use the notation of [Ree88], and introduce some new definitions. A glossary of the notation introduced here is provided at the end of this dissertation.

Recall from [Ree88] the universal delay constant δ, which represents the time taken for a process to make a recursive call, or to recover from the performance of an event. We assume in this dissertation that the duration δ is significantly smaller than one time unit (no greater than *0.1* time units).

Timed Traces

Timed traces are finite sequences of (*time,event*) pairs where the times associated with the events appear in non-decreasing order. Times are drawn from $[0, \infty)$, and events from the universal set of events Σ. The set of all possible timed traces is denoted $T\Sigma^{*}_{\leq}$. We write $s_1 \frown s_2$ to represent the concatenation of traces s_1 and s_2, and $\#s$ to represent the length of s. As in [Hoa85], we define the relation **in** as follows:

$$s_1 \ \mathbf{in}\ s_2 \ \mathrel{\widehat{=}}\ \exists\, u, v \bullet u \frown s_1 \frown v = s_2$$

Tbis relation holds whenever the first trace is a contiguous subsequence of the second.

The *first* and *last* operators are defined upon non-empty traces, returning the first and last events in a trace, respectively:

$$\begin{aligned} first(\langle (t,a)\rangle \frown s) &\mathrel{\widehat{=}} a \\ last(s \frown \langle (t,a)\rangle) &\mathrel{\widehat{=}} a \end{aligned}$$

To ease the subsequent mathematics we allow $first(\langle\rangle) = last(\langle\rangle) = e$ for some object $e \notin \Sigma$ (suggested by J. Davies). The *begin* and *end* operators are defined for all traces:

$$\begin{aligned} begin(\langle\rangle) &\mathrel{\widehat{=}} \infty \\ begin(\langle (t,a)\rangle \frown s) &\mathrel{\widehat{=}} t \\ end(\langle\rangle) &\mathrel{\widehat{=}} 0 \\ end(s \frown \langle (t,a)\rangle &\mathrel{\widehat{=}} t \end{aligned}$$

Again, the values chosen for the empty trace are the most convenient for the subsequent mathematics.

The $@n$ operator returns the nth timed event of a trace. It is defined as follows:

$$\begin{aligned}
\langle\rangle @ n &\mathrel{\widehat{=}} (\infty, e)\\
(\langle(t,a)\rangle^\frown s) @ 1 &\mathrel{\widehat{=}} (t,a)\\
(\langle(t,a)\rangle^\frown s) @ (n+1) &\mathrel{\widehat{=}} s @ n
\end{aligned}$$

We define the *during*, *before*, and *after* operators on timed traces. The first returns the subsequence of the trace with times drawn from set I. The others return the parts of the trace before and after the specified time.

$$\begin{array}{rcll}
\langle\rangle \uparrow I &\mathrel{\widehat{=}}& \langle\rangle & \\
(\langle(t,a)\rangle^\frown s) \uparrow I &\mathrel{\widehat{=}}& \langle(t,a)\rangle^\frown(s \uparrow I) & \text{if } t \in I\\
&& (s \uparrow I) & \text{otherwise}\\
\langle\rangle \upharpoonright t &\mathrel{\widehat{=}}& \langle\rangle & \\
(\langle(t',a)\rangle^\frown s) \upharpoonright t &\mathrel{\widehat{=}}& \langle(t',a)\rangle^\frown(s \upharpoonright t) & \text{if } t' \leqslant t\\
&& \langle\rangle & \text{otherwise}\\
s \upharpoonleft t &\mathrel{\widehat{=}}& s \uparrow (t, \infty) &
\end{array}$$

where I is a set of time values. In the case that $I = \{t\}$ for some time t, we may omit the set brackets. The *before* operator, $\upharpoonright$, is also used to denote the restriction of a trace to events drawn from a given set of events. If the second argument of the operator is a set, then:

$$\begin{array}{rcll}
\langle\rangle \upharpoonright A &\mathrel{\widehat{=}}& \langle\rangle & \\
(\langle(t,a)\rangle^\frown s) \upharpoonright A &\mathrel{\widehat{=}}& \langle(t,a)\rangle^\frown(s \upharpoonright A) & \text{if } a \in A\\
(\langle(t,a)\rangle^\frown s) \upharpoonright A &\mathrel{\widehat{=}}& s \upharpoonright A & \text{otherwise}\\
(\langle(t,\hat{a})\rangle^\frown s) \upharpoonright A &\mathrel{\widehat{=}}& \langle(t,\hat{a})\rangle^\frown(s \upharpoonright A) & \text{if } a \in A\\
(\langle(t,\hat{a})\rangle^\frown s) \upharpoonright A &\mathrel{\widehat{=}}& s \upharpoonright A & \text{otberwise}
\end{array}$$

If timed refusals are not being considered, the events in a timed trace may be labelled with hats, to indicate that they have occurred at the instant they became available; the operator *hstrip* strips the hats from a timed trace:

$$\begin{aligned}
\mathit{hstrip}(\langle\rangle) &\mathrel{\widehat{=}} \langle\rangle\\
\mathit{hstrip}(\langle(t,a)\rangle^\frown s) &\mathrel{\widehat{=}} \langle(t,a)\rangle^\frown \mathit{hstrip}(s)\\
\mathit{hstrip}(\langle(t,\hat{a})\rangle^\frown s) &\mathrel{\widehat{=}} \langle(t,a)\rangle^\frown \mathit{hstrip}(s)
\end{aligned}$$

Following Reed, we write $\check{s}$ for $\mathit{hstrip}(s)$.

The operator *tstrip* strips the timing information from a trace:

$$\begin{aligned}
\mathit{tstrip}(\langle\rangle) &\mathrel{\widehat{=}} \langle\rangle\\
\mathit{tstrip}(\langle(t,a)\rangle^\frown s) &\mathrel{\widehat{=}} \langle a\rangle^\frown \mathit{tstrip}(s)
\end{aligned}$$

We use *thstrip* to denote the composition of these two functions.

We define an operator σ on traces, which yields the set of events present in the trace:

$$\sigma(s) \mathrel{\widehat{=}} \{a \in \Sigma \mid \exists\, t \bullet \langle(t,a)\rangle \text{ in } \mathit{hstrip}(s)\}$$

Note that we discard the 'hat' information when considering which events are present in a trace.

We define a temporal shift operator:

$$\begin{array}{rcll} \langle\rangle \dot{-} t & \mathrel{\widehat{=}} & \langle\rangle & \\ (\langle(t_1,a)\rangle^\frown s) \dot{-} t & \mathrel{\widehat{=}} & \langle(t_1 - t, a)\rangle^\frown(s \dot{-} t) & \text{if } t_1 \geqslant t \\ (\langle(t_1,a)\rangle^\frown s) \dot{-} t & \mathrel{\widehat{=}} & s \dot{-} t & \text{otherwise} \end{array}$$

and a count operator, $\downarrow$, which returns the number of occurrences of events from a given set:

$$s \downarrow A \mathrel{\widehat{=}} \#(s \upharpoonright A)$$

In the case that $A = \{a\}$ for some event a, we omit the brackets.

The following functions are used in conjunction with the corresponding *TCSP* operators. We define a simple hiding operator on traces, with the effect of removing hidden events:

$$s \setminus A \mathrel{\widehat{=}} s \upharpoonright (\Sigma - A)$$

and an equivalence relation $\cong$ on timed traces as follows: $u \cong v$ if and only if u is a permutation of v. As both are timed traces, only events occurring at the same time may be interchanged. Formally, $u \cong v$ precisely when there is a bijective function $f : \{i \in \mathbb{N} \mid 1 \leqslant i \leqslant \#u\} \to \{i \in \mathbb{N} \mid 1 \leqslant i \leqslant \#u\}$ such that $\forall\, i \in \{i \in \mathbb{N} \mid 1 \leqslant i \leqslant \#u\} \bullet u@i = v@f(i)$.

Finally, we define parallel operators on traces, corresponding to the effect of parallel composition in Timed CSP. These are used in the semantic equations.

$$\begin{array}{rcl} u \;{}_X\|_Y\; v & \mathrel{\widehat{=}} & \{s \mid s \upharpoonright X = u \wedge s \upharpoonright Y = v \wedge s \upharpoonright (X \cup Y) = s\} \\ u \;|||\; v & \mathrel{\widehat{=}} & \{s \mid \forall\, t \bullet s \uparrow t \cong (u \uparrow t)^\frown(v \uparrow t)\} \end{array}$$

We write $\mathit{Tmerge}(u,v)$ to denote the set of all traces in $T\widehat{\Sigma}^*_{\leqslant}$ obtained by interleaving u and v. When $\tilde{u} = \tilde{v}$ we define the trace $s = u \vee v$ to be such that $\tilde{s} = \tilde{v}$, and the nth element of s is hatted precisely when the nth element of u or v is.

Timed Refusals

Recall the definitions from [Ree88]:

$$\begin{array}{rcl} TINT & \widehat{=} & \{[l,r) \mid l,r \in \mathbf{R} \wedge \mathit{0} \leqslant l < r < \infty\} \\ RTOK & \widehat{=} & \{I \times X \mid I \in TINT \wedge X \in \mathbf{P}(\Sigma)\} \\ RSET & \widehat{=} & \{\bigcup Z \mid Z \in p(RTOK)\} \end{array}$$

Refusal tokens are drawn from $RTOK$. Each refusal token is the cross product of a half open interval of times, and a set of events, and is therefore considered as a set of (time,event) pairs. Refusals $\aleph$ are drawn from $RSET$, and therefore consist of finite unions of these refusal tokens. Hence a refusal set is also a set of (time,event) pairs.

A number of the operators of the previous section have a similar action when applied to timed refusal sets. If we make the definition

$$I(\aleph) \;\widehat{=}\; \{t \mid \exists\, a \in \Sigma \bullet (t,a) \in \aleph\}$$

then we can define *begin* and *end* on refusals, to be the infimum and supremum respectively, of the times at which events are refused:

$$\begin{array}{rcll} begin(\aleph) & \widehat{=} & \infty & \text{if } \aleph = \emptyset \\ begin(\aleph) & \widehat{=} & inf(I(\aleph)) & \text{otherwise} \\ end(\aleph) & \widehat{=} & \mathit{0} & \text{if } \aleph = \emptyset \\ end(\aleph) & \widehat{=} & sup(I(\aleph)) & \text{otherwise} \end{array}$$

The *before*, *after*, and *during* operators can be defined on refusals:

$$\begin{array}{rcl} \aleph \upharpoonright t & \widehat{=} & \aleph \cap ([\mathit{0},t) \times \Sigma) \\ \aleph \upharpoonleft t & \widehat{=} & \aleph \cap ([t,\infty) \times \Sigma) \\ \aleph \uparrow [t_1,t_2) & \widehat{=} & \aleph \cap ([t_1,t_2) \times \Sigma) \end{array}$$

Recalling that Σ denotes the set of all events, we see that these restrict a refusal set to events that may be refused before, during, and after the specified times.

We overload the $\upharpoonright$ symbol to denote set restriction:

$$\aleph \upharpoonright A \;\widehat{=}\; \aleph \cap ([\mathit{0},\infty) \times A)$$

with hiding defined in terms of restriction:

$$\aleph \setminus A \;\widehat{=}\; \aleph \upharpoonright (\Sigma - A)$$

We define a temporal shift operator on refusals:

$$\aleph \dot{-} t \;\;\widehat{=}\;\; \{(t_1 - t, a) \mid (t_1, a) \in \aleph \wedge t_1 \geqslant t\}$$

We define an alphabet operator σ:

$$\sigma(\aleph) \;\;\widehat{=}\;\; \{a \in \Sigma \mid \exists\, t \bullet (t, a) \in \aleph\}$$

All of the operators on refusal sets may be extended to infinite refusal sets (which are discussed in chapter 3)

Failures

For convenience, we extend some of the above definitions to individual timed (*trace,refusal*) pairs, which are termed *failures*.

$$\begin{aligned}
begin(s,\aleph) \;&\widehat{=}\; min(\{begin(s), begin(\aleph)\}) \\
end(s,\aleph) \;&\widehat{=}\; max(\{end(s), end(\aleph)\}) \\
(s,\aleph) \uparrow I \;&\widehat{=}\; (s \uparrow I, \aleph \uparrow I) \\
(s,\aleph) \upharpoonright t \;&\widehat{=}\; (s \upharpoonright t, \aleph \upharpoonright t) \\
(s,\aleph) \upharpoonleft t \;&\widehat{=}\; (s \upharpoonleft t, \aleph \upharpoonleft t) \\
\sigma(s,\aleph) \;&\widehat{=}\; \sigma(s) \cup \sigma(\aleph) \\
(s,\aleph) \dot{-} t \;&\widehat{=}\; (s \dot{-} t, \aleph \dot{-} t) \\
(s,\aleph) \upharpoonright A \;&\widehat{=}\; (s \upharpoonright A, \aleph \upharpoonright A) \\
(s,\aleph) \setminus A \;&\widehat{=}\; (s \setminus A, \aleph \setminus A)
\end{aligned}$$

We define a predicate $\widehat{A}$ on failures, to indicate that the set A is forced over the entire behaviour, and hence that all occurrences of events from A in the trace happen as soon as possible. Such a failure is termed A-*active*.

$$\widehat{A}(s,\aleph) \;\;\widehat{=}\;\; [0, end(s,\aleph)) \times A \subseteq \aleph$$

This allows an alternative formulation of the semantic equation for hiding in TM_F:

$$\mathcal{F}_T[\![P \setminus A]\!] \;\;=\;\; \{(s \setminus A, \aleph - \aleph') \mid \widehat{A}(s,\aleph) \wedge (s,\aleph) \in \mathcal{F}_T[\![P]\!] \wedge \sigma(\aleph') \subseteq A\}$$

Processes

The functions *traces*, *fail* and *stab* are defined on sets of *(trace,stability,refusal)* triples as in [Ree88]:

$$\begin{aligned}
traces(P) &\mathrel{\widehat{=}} \{s \mid (s,\alpha,\aleph) \in P\}\\
fail(P) &\mathrel{\widehat{=}} \{(s,\aleph) \mid (s,\alpha,\aleph) \in P\}\\
stab(P) &\mathrel{\widehat{=}} \{(s,\alpha) \mid (s,\alpha,\aleph) \in P\}\\
SUP(P) &\mathrel{\widehat{=}} \{(s,\alpha,\aleph) \mid (s,\aleph) \in fail(P) \wedge \alpha = sup\{\beta \mid (s,\beta,\aleph) \in P\}\}\\
CL_{\cong}(P) &\mathrel{\widehat{=}} \{(s,\alpha,\aleph) \mid \exists (w,\alpha,\aleph) \in P \bullet w \cong s\}
\end{aligned}$$

These functions are extended to apply to *TCSP* processes, by applying them to the semantics (in TM_{FS}) of their arguments.

We extend the alphabet operator to *TCSP* processes:

$$\sigma(P) \mathrel{\widehat{=}} \bigcup\{\sigma(s) \mid s \in traces(P)\}$$

and observe that this differs from the *alphabet* concept nsed in other versions of *CSP* (e.g. [Hoa85]). The operator defined here only allows events that a process may perform to be included in its alphabet, whereas the alphabet concept in [Hoa85] also allows events that the process cannot perform to appear in its alphabet.

We use $\mathcal{T}$, $\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}$ to denote the semantic mappings from *CSP* to M_T, M_S, M_F, and M_{FS} respectively. We use $\mathcal{T}_T$, $\mathcal{S}_T$, $\mathcal{F}_T$, $\mathcal{E}^*_T$, and $\mathcal{E}_T$ to denote the semantic mappings from *TCSP* to TM_T, TM_S, TM_F, TM^*_{FS}, and TM_{FS} respectively (see appendix B).

We use the $\equiv$ relation between *TCSP* processes when they have the same semantics in the model TM_{FS} (given in [Ree88] and reproduced in appendix B.2).

$$P \equiv Q \mathrel{\widehat{=}} \mathcal{E}_T[\![P]\!] = \mathcal{E}_T[\![Q]\!]$$

If two processes are equivalent in this sense, then they will also have the same semantics in TM_F. Hence the laws given later for TM_{FS} also hold for TM_F.

The set of the possible initial events of a process P is denoted $inits(P)$. It is defined by

$$inits(P) \mathrel{\widehat{=}} \{a \mid \exists\, t \bullet \langle (t,a) \rangle \in traces(P)\}$$

As in untimed *CSP*, we may label processes by applying an alphabet transformation to them.

$$i : P \mathrel{\widehat{=}} f_i(P)$$

where $f_i(a) = i.a$ for all events $a \in \Sigma \setminus \{\checkmark\}$; but termination is still signalled by $\checkmark$ so we have $f_i(\checkmark) = \checkmark$. The labels may be removed by applying f_i^{-1} to the process. We define $rem_i(P) = f_i^{-1}(P)$.

Finally, we will use an indexed parallel operator, with finite indexing set, as an abbreviation for a sequence of binary parallel operations. It is defined inductively on the indexing set:

$$\begin{aligned} \mathop{\Big\|}_{\{X_i \mid 1 \leqslant i \leqslant 2\}} P_i &\mathrel{\widehat{=}} P_1 \ {}_{X_1}\|_{X_2} \ P_2 \\ \mathop{\Big\|}_{\{X_i \mid 1 \leqslant i \leqslant n+1\}} P_i &\mathrel{\widehat{=}} \Big(\mathop{\Big\|}_{\{X_i \mid 1 \leqslant i \leqslant n\}} P_i \Big) \ {}_{(\bigcup_{1 \leqslant i \leqslant n} X_i)}\|_{X_{n+1}} \ P_{n+1} \end{aligned}$$

when $n \geqslant 2$. This indexed parallel operator is entirely similar to the untimed version, and so we have the standard result that the order in which the (P_i, X_i) pairs are labelled does not affect the semantics of the parallel composition.

3 Additions to Timed CSP

The semantic models presented in [RR86, RR87, Ree88] (reproduced in Appendix B) provide a secure foundation for the modelling of real-time concurrency. In this chapter we build upon that foundation by defining new syntactic process constructors in terms of the basic ones, thus guaranteeing their well-definedness and continuity. These operators include both timeout and interrupt, which allow the explicit introduction of such hehaviour into a process description. We also propose a defining equation for arbitrary non-deterministic choice, and see that it is not always well defined. We discuss a timed *after* operator, and the notion of infinite behaviours, which will be used to succinctly capture particular desirable aspects of behaviour in chapter 5, and to define strong timewise refinement in chapter 6.

3.1 Delayed Sequential Composition

The semantics given for the sequential composition operator $P\,;\,Q$ allows instantaneous passing of control from P to Q, at the moment that P successfully terminates. This may introduce unwanted non-determinism, since both P and Q may perform events at the instant of P's successful termination. If an event is possible for both processes then the choice between them will be non-deterministic. In many cases the complexity introduced by this behaviour is not necessary for correctness of processes, and serves only to make verification more difficult. We shall see when we consider the proof system that it is easier to analyse a sequential operator that enforces a delay between successful termination of its first operand and the passing of control to its second. We sball also see, in Chapter 6, that it is advantageous to use the delayed sequential composition operator in producing timewise refinements of untimed CSP processes.

We define the new sequential composition as follows:

$$P \fatsemi Q \;\; \widehat{=} \;\; P\,;\,WAIT\,\delta\,;\,Q$$

It follows that $\fatsemi$ is well defined and continuous, since it is defined in terms of the basic *TCSP* operators. We can derive a semantic equatiou for $P \fatsemi Q$ in the evaluation domain TM_{FS} from the equations for the sequential operator ; and $WAIT\,\delta$. This may be simplified to eliminate the closure operator $CL_{\trianglelefteq}$, whose presence in the equation for $P\,;\,Q$ is made necessary by the instantaneous passing of control.

Theorem 3.1.1 *For any P and Q, the semantie equations for $P \fatsemi Q$ may be written as follows:*

$$
\begin{aligned}
\mathcal{E}_T[\![P \fatsemi Q]\!] \;=\;& SUP(\{(s,\alpha,\aleph) \mid \checkmark \notin \sigma(s) \wedge \forall I \in TINT \bullet \\
&\qquad\qquad\qquad (s,\alpha,\aleph \cup (I \times \{\checkmark\})) \in \mathcal{E}_T[\![P]\!]\} \\
&\quad \cup \\
&\quad \{(s,\alpha,\aleph) \mid \exists t \bullet \checkmark \notin \sigma(s \upharpoonright t) \\
&\qquad \wedge ((s \upharpoonright t)^\frown\langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup ([0,t) \times \{\checkmark\})) \in fail(\mathcal{E}_T[\![P]\!]) \\
&\qquad \wedge (s \mathbin{\dot{-}} (t+\delta), \alpha - (t+\delta), \aleph \mathbin{\dot{-}} (t+\delta)) \in \mathcal{E}_T[\![Q]\!]\})
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}^*_T[\![P \fatsemi Q]\!] \;=\;& SUP(\{(s,\alpha,X) \mid \checkmark \notin \sigma(s) \wedge (s,\alpha,X \cup \{\checkmark\}) \in \mathcal{E}^*_T[\![P]\!]\} \\
&\quad \cup \\
&\quad \{(s,\alpha,X) \mid \exists t \bullet (s \upharpoonright t)^\frown\langle(t,\hat{\checkmark})\rangle \in traees(\mathcal{E}^*_T[\![P]\!]) \\
&\qquad\qquad \wedge \checkmark \notin \sigma(s \upharpoonright t) \wedge s \uparrow (t, t+\delta) = \langle\rangle \\
&\qquad\qquad \wedge (s \mathbin{\dot{-}} (t+\delta), \alpha - (t+\delta), X) \in \mathcal{E}^*_T[\![Q]\!]\}
\end{aligned}
$$

Observe that $SKIP \fatsemi P \equiv WAIT\,\delta\,; P$, and that $WAIT\,t \fatsemi P \equiv WAIT\,(t+\delta)\,; P$. The $\fatsemi$ operator cannot be used with the $WAIT$ construct to introduce delays of length less than δ.

This new operator obeys many of the laws which hold of the original sequential composition operator:

$$
\begin{aligned}
(a \rightarrow P) \fatsemi Q &\equiv a \rightarrow (P \fatsemi Q) \qquad (a \neq \checkmark) \\
(P \fatsemi Q) \fatsemi R &\equiv P \fatsemi (Q \fatsemi R) \\
(WAIT\,t_1 \;|||\; WAIT\,t_2) \fatsemi P &\equiv WAIT\,min\{t_1, t_2\} \fatsemi P
\end{aligned}
$$

However, other equivalences are no longer valid because of the delay in transfer of control:

$$
\begin{aligned}
WAIT\,t_1 \fatsemi WAIT\,t_2 &\not\equiv WAIT\,(t_1 + t_2) \\
SKIP \fatsemi P &\not\equiv P
\end{aligned}
$$

In place of the first of these we have

$$
WAIT\,t_1 \fatsemi WAIT\,t_2 \;\equiv\; WAIT\,(t_1 + t_2 + \delta)
$$

3.2 Indexed Choice

3.2.1 Arbitrary non-deterministic choice

We define an indexed non-deterministic choice operator, while recognising that it is does not always have a well-defined semantics, and does not provide for an effective treatment of unbounded non-determinism which does not manifest itself in a finite time. However, it is useful in many situations, such as the modelling of a process which may terminate at any time chosen non-deterministically from an interval.

The addition to the syntax of *TCSP* of an infinite non-deterministic choice operator involves an extension of each of the semantic mappings (see appendix B) from *TCSP* to the various timed models:

Definition 3.2.1 *We extend the semantic mappings to the timed models as follows:*

$$\begin{aligned}
\mathcal{E}_T[\![\bigsqcap_{i\in I} P_i]\!] &\mathrel{\widehat{=}} SUP(\bigcup_{i\in I}\mathcal{E}_T[\![P_i]\!]) \\
\mathcal{E}_T^*[\![\bigsqcap_{i\in I} P_i]\!] &\mathrel{\widehat{=}} SUP(\bigcup_{i\in I}\mathcal{E}_T^*[\![P_i]\!]) \\
\mathcal{F}_T[\![\bigsqcap_{i\in I} P_i]\!] &\mathrel{\widehat{=}} \bigcup_{i\in I}\mathcal{F}_T[\![P_i]\!] \\
\mathcal{S}_T[\![\bigsqcap_{i\in I} P_i]\!] &\mathrel{\widehat{=}} SUP(\bigcup_{i\in I}\mathcal{S}_T[\![P_i]\!]) \\
\mathcal{T}_T[\![\bigsqcap_{i\in I} P_i]\!] &\mathrel{\widehat{=}} \bigcup_{i\in I}\mathcal{T}_T[\![P_i]\!]
\end{aligned}$$

However, the mappings are not in general well defined: the set $SUP(\bigcup_{i\in I}\mathcal{E}_T[\![P_i]\!])$ satisfies all tbe axioms of the model TM_{FS} (reproduced on page 194) with the possible exception of axiom 5, the bounded speed axiom (also axiom 5 in TM_F, axiom 6 in the other models). Recall that this axiom states that

$$\forall t \in [0,\infty), \exists n \in \mathbb{N}, \forall s \in traces(S) \bullet end(s) \leqslant t \Rightarrow \#s \leqslant n$$

for any element S of TM_{FS}. Now consider

$$\begin{aligned}
P_0 &\mathrel{\widehat{=}} STOP \\
P_{n+1} &\mathrel{\widehat{=}} P_n \;|||\; a \rightarrow STOP \\
P &\mathrel{\widehat{=}} \bigsqcap_{n\in\mathbb{N}} P_n
\end{aligned}$$

For every n, process P_n may perform n a's at time 0. Hence there is no bound on the number of a's that P, the non-deterministic choice of all the P_n, may perform at time 0; so axiom 5 does not hold of our proposed semantics for P.

If we wish to use an infinite non-deterministic choice operator, we must take care that this problem does not arise, by ensuring that the set of processes over which the choice is made has a function which places a bound upon the speed of all the processes in that set.

Definition 3.2.2 *A set* $\{P_i \mid i \in I\}$ *is* uniformly bounded in TM_{FS} *if*

$$\begin{aligned} &\exists\, n : [0,\infty) \to \mathbb{N}, \forall\, i \in I, t \in [0,\infty) \bullet \\ &\qquad s \in traces(\mathcal{E}_T[\![P_i]\!]) \wedge end(s) \leqslant t \Rightarrow \#s \leqslant n(t) \end{aligned}$$

This definition extends to the other timed models in the obvious way.

The function n provides a bound for the speed of each of the processes separately, and thus upon the speed of the non-deterministic composition, and will ensure that $\bigcup_{i\in I} \mathcal{E}_T[\![P_i]\!]$ satisfies the bounded speed axiom. Observe that any finite set of processes is uniformly hounded in each timed model.

If a set of processes is *not* uniformly bounded, then the bounded speed axiom will not hold of $\bigcup_{i\in I} \mathcal{E}_T[\![P_i]\!]$. Thus we bave a necessary and sufficient condition for $\mathcal{E}_T[\![\bigsqcap_{i\in I} P_i]\!]$ to be well-defined:

Lemma 3.2.3 $\mathcal{E}_T[\![\bigsqcap_{i\in I} P_i]\!]$ *is well-defined if and only if the set*

$$\{P_i \mid i \in I\}$$

is uniformly bonnded in TM_{FS}.

This result extends immediately to the other models.

A useful result we obtain from this analysis is that we are able to model a delay wbich is non-deterministic over some interval. For all t, the process $WAIT\ t$ has its speed bounded by $\lambda\, t \bullet 1$, since it may perform at most one event. Hence, for any set $T \subseteq [0,\infty)$, we have that the set of processes $\{WAIT\ t \mid t \in T\}$ is uniformly bounded, and so $\bigsqcap_{t\in T} WAIT\ t$ will have a well defined semantics; we will abbreviate it $WAIT\ T$

Corollary 3.2.4 *The semantics of the process WAIT T is always well defined, where*

$$WAIT\ T \;\widehat{=}\; \bigsqcap_{t\in T} WAIT\ t$$

3.2.2 Indexed prefix choice

As we shall see in chapter 7, indexed prefix choice is required for the definition of inputting a message onto a channel. Such a construct cannot be modelled with binary deterministic choice if the number of possible messages for the channel is infinite.

Definition 3.2.5 *We extend the semantic mappings of the timed models as follows:*

$$\begin{aligned}
\mathcal{E}_T[\![a : A \rightarrow P_a]\!] \;\widehat{=}\;\; & \{(\langle\rangle, 0, \aleph) \mid A \cap \sigma(\aleph) = \emptyset\} \\
& \cup\{((\langle(t, a)\rangle^\frown(s + (t + \delta)), \alpha + t + \delta, \aleph) \mid \\
& \qquad a \in A \wedge t \geqslant 0 \wedge A \cap \sigma(\aleph \restriction t) = \emptyset \\
& \qquad \wedge (s, \alpha, \aleph \dot{-} (t + \delta)) \in \mathcal{E}_T[\![P(a)]\!]\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}^*_T[\![a : A \rightarrow P(a)]\!] \;\widehat{=}\;\; & \{(\langle\rangle, 0, X) \mid A \cap X = \emptyset\} \\
& \cup\{((\langle(0, \hat{a})\rangle^\frown(s + \delta), \alpha + \delta, X) \mid \\
& \qquad a \in A \wedge (s, \alpha, X) \in \mathcal{E}^*_T[\![P(a)]\!]\} \\
& \cup\{((\langle(t, a)\rangle^\frown(s + t + \delta), \alpha + t + \delta, X) \mid \\
& \qquad a \in A \wedge t \geqslant 0 \wedge (s, \alpha, X) \in \mathcal{E}^*_T[\![P(a)]\!]\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}_T[\![a : A \rightarrow P_a]\!] \;\widehat{=}\;\; & \{(\langle\rangle, \aleph) \mid A \cap \sigma(\aleph) = \emptyset\} \\
& \cup\{((\langle(t, a)\rangle^\frown(s + (t + \delta)), \aleph) \mid \\
& \qquad a \in A \wedge t \geqslant 0 \wedge A \cap \sigma(\aleph \restriction t) = \emptyset \\
& \qquad \wedge (s, \aleph \dot{-} (t + \delta)) \in \mathcal{F}_T[\![P(a)]\!]\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{S}_T[\![a : A \rightarrow P(a)]\!] \;\widehat{=}\;\; & \{(\langle\rangle, 0)\} \\
& \cup\{(\langle(0, \hat{a})\rangle^\frown(s + \delta), \alpha + \delta) \mid a \in A \wedge (s, \alpha) \in \mathcal{S}_T[\![P(a)]\!]\} \\
& \cup\{((\langle(t, a)\rangle^\frown(s + t + \delta), \alpha + t + \delta) \mid \\
& \qquad a \in A \wedge t \geqslant 0 \wedge (s, \alpha) \in \mathcal{S}_T[\![P(a)]\!]\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}_T[\![a : A \rightarrow P(a)]\!] \;\widehat{=}\;\; & \{\langle\rangle\} \\
& \cup\{\langle(0, \hat{a})\rangle^\frown(s + \delta) \mid a \in A \wedge s \in \mathcal{T}_T[\![P(a)]\!]\} \\
& \cup\{\langle(t, a)\rangle^\frown(s + t + \delta) \mid a \in A \wedge t \geqslant 0 \wedge s \in \mathcal{T}_T[\![P(a)]\!]\}
\end{aligned}$$

Recall that in the models without timing information, the presence of a hat on an event in the trace indicates that it happened at the moment it became available.

It should be noted that the same clash with axiom 5 arises with the treatment of indexed prefix choice. Recalling the processes P_n defined above, it can be seen that the semantics of $n : \mathbb{N} \rightarrow P_n$ will not meet that axiom, for exactly the same reason as in the case of the infinite non-determinism operator. To define infinite prefix choice, the same condition is necessary and sufficient:

Lemma 3.2.6 *The semantics of $a : A \rightarrow P(a)$ is well-defined if and only if the set of processes $\{P(a) \mid a \in A\}$ is uniformly bounded.*

3.3 An alternative syntax for TCSP

As we shall see, the instantaneous sequential composition operator is awkward to use both in the proof system and in timewise refinement; the closure operator used in the defining equation gives rise to unnecessary complications in the associated proof rule, and in conditions for preservation of timewise refinements. By using the delayed sequential composition operator we can bypass these problems. However, if the first argument is a simple delay, then the complications do not arise. Hence we retain the delay operation $WAIT\ t\, ; P$.

We will use the syntax given below as our language $TCSP$. We have added the two indexed choice constructs, and have replaced instantaneous sequential composition with delayed sequential composition. We have retained the instantaneous sequential composition operator when it is used with a $WAIT\ t$ command. With the exception of these alterations, the syntax is identical to that defined in [Ree88]. Our syntax for $TCSP$ is given as follows:

$$
\begin{aligned}
P \;::=\; & STOP \mid \bot \mid SKIP \mid WAIT\ t \mid \\
& a \rightarrow P \mid a : A \rightarrow P(a) \mid P \sqcap P \mid \bigsqcap_i P_i \mid P \,\Box\, P \mid \\
& P \parallel P \mid P\ {}_A\|_B\ P \mid P \,|||\, P \mid \\
& WAIT\ t\,; P \mid P \fatsemi P \mid P \setminus A \mid \\
& f^{-1}(P) \mid f(P) \mid \mu X \bullet F(X)
\end{aligned}
$$

If we wish to give a more explicit semantic treatment of recursion, then we nse a syntax with process variables, as follows:

$$\begin{aligned} P \ ::= \ & STOP \mid \perp \mid SKIP \mid WAIT\,t \mid \\ & a \rightarrow P \mid a : A \rightarrow P(a) \mid P \sqcap P \mid \bigsqcap_i P_i \mid P \Box P \mid \\ & P \parallel P \mid P\ {}_A\|_B\ P \mid P \mid\mid\mid P \mid \\ & WAIT\,t\,; P \mid P\,;P \mid P \setminus A \mid \\ & f^{-1}(P) \mid f(P) \mid X \mid \mu\, X \bullet P \end{aligned}$$

where X is of type *var*, the set of process variable names. This approach is more laborious, since the semantic equations require environments, which are only used explicitly in the definition of recursion.

The syutax above gives rise to *TCSP terms.* The processes *STOP*, *SKIP*, $\perp$ and *WAIT t* have no free process variables; X has as its set of free process variables the set $\{X\}$, the set of free process variables of $\mu\, X \bullet P$ is that of P with X removed; and the set of free variables of any other compound process is the union of the sets of free variables of the component processes. Any *TCSP* term which has no free process variables (in the sense of [Ros82]) is a *TCSP process.* The semantics of a *TCSP* process is independent of the environment in which it is evaluated, so the semantic equations give rise to exactly the same semantics as that provided by the equations given in [Ree88].

We will use the syutax with variables in chapter 6, where an explicit treatment of recursion is required; with the exceptiou of that chapter, we will use the syntax without variables throughout the thesis, although we are aware that we could provide a more explicit (though more laborious) justification of our results if required.

3.4 Derived Operators

We make implicit use of the following lemma in the construction of these operators:

Lemma 3.4.1 $\sigma(P) \cap X = \langle\rangle \Rightarrow P \equiv P \setminus X$

This lemma states that the hiding of an impossible set of events from a process will not alter the behaviour of the process.

Our motivation for defining the following operators by syntactic equivalence is that we automatically obtain the result that the operators are non-expanding. In practice, when reasoning about processes built with these operators, it will usually be easier to use the derived semantic equations.

3.4.1 Timeout

One important aspect of time-critical behaviour is the *timeout*, where control is passed from one process to another if the first performs no external actions in a given period of time. This behaviour can be modelled in Timed *CSP*.

Definition 3.4.2 *The timeout operator* $\overset{t}{\triangleright}$ *is defined as follows:*

$$P \overset{t}{\triangleright} Q \;\;\hat{=}\;\; rem_i((i : P \,\Box\, WAIT\ t\, ;\, trig \to i : Q) \setminus \{trig\})$$

If $i : P$ performs no visible actions by time t, then the event $trig$ is made available by $WAIT\ t\, ;\, trig \to i : Q$. The abstraction operator forces this event to occur as soon as it becomes available, resolving the choice against process $i : P$; control passes to process $(i : Q) \setminus trig$, which is equivalent to $i : Q$, since $trig \notin \sigma(i : Q)$. If $i : P$ does perform a visible action by time t, then the choice is resolved in favour of $(i : P) \setminus trig$, equivalent to P, and the other side of the choice is no longer a possibility. Observe that $i : P$ may not resolve the choice with a hidden action, since it cannot perform event $trig$.

If $trig \notin \sigma(P) \cup \sigma(Q)$ then we have the following equivalence:

$$P \overset{t}{\triangleright} Q \;\;\equiv\;\; (P \,\Box\, WAIT\ t\, ;\, trig \to Q) \setminus \{trig\}$$

Further, if P is unable to terminate successfully, $\checkmark \notin \sigma(P)$, then we have

$$P \overset{t}{\triangleright} Q \;\;\equiv\;\; (P \,\Box\, WAIT\ t)\, ;\, Q$$

Example

The process $(a \to c \to STOP) \overset{5}{\triangleright} b \to STOP$ is initially prepared to engage in event a. If a is performed within 5 time units, then (after a delay of δ) the process behaves like $c \to STOP$. If the a does not occur within 5 time units, then the offer is withdrawn and the process behaves like $b \to STOP$.

Theorem 3.4.3 *The semantics of $P \overset{t}{\rhd} Q$ in TM_F simplifies to the following equation:*

$$\begin{aligned}
\mathcal{F}_T[\![P \overset{t}{\rhd} Q]\!] \;=\; & \{(s,\aleph) \mid begin(s) \leqslant t \wedge (s,\aleph) \in \mathcal{F}_T[\![P]\!] \\
& \vee \\
& ((\langle\rangle, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!] \wedge begin(s) \geqslant t + \delta \wedge \\
& (s \dot{-} (t+\delta), \aleph \dot{-} (t+\delta)) \in \mathcal{F}_T[\![Q]\!]\}
\end{aligned}$$

The timeout operator obeys the following laws:

$$\begin{array}{rcll}
P \overset{t_1}{\rhd} (Q \overset{t_2}{\rhd} R) & \equiv & (P \overset{t_1}{\rhd} Q) \overset{t_1+t_2+\delta}{\rhd} R & \\
(P \sqcap Q) \overset{t}{\rhd} R & \equiv & (P \overset{t}{\rhd} R) \sqcap (Q \overset{t}{\rhd} R) & \\
P \overset{t}{\rhd} (Q \sqcap R) & \equiv & (P \overset{t}{\rhd} Q) \sqcap (P \overset{t}{\rhd} R) & \\
(P \mathbin{\Box} Q) \overset{t}{\rhd} R & \equiv & (P \overset{t}{\rhd} R) \mathbin{\Box} (Q \overset{t}{\rhd} R) & \\
P \overset{t}{\rhd} (Q \mathbin{\Box} R) & \equiv & (P \overset{t}{\rhd} Q) \mathbin{\Box} (P \overset{t}{\rhd} R) & \\
(P \parallel Q) \overset{t}{\rhd} (R \parallel S) & \equiv & (P \overset{t}{\rhd} R) \parallel (Q \overset{t}{\rhd} S) & \\
(P \overset{t}{\rhd} Q) \parallel (WAIT\; t \fatsemi R) & \equiv & WAIT\; t \fatsemi (Q \parallel R) & \\
(WAIT\; t_1 \fatsemi P) \overset{t_2+\delta}{\rhd} Q & \equiv & WAIT\; t_1 \fatsemi (P \overset{t_2-t_1}{\rhd} Q) & \text{if } t_2 \geqslant t_1 \\
 & & WAIT\; (t_2+\delta) \fatsemi Q & \text{if } t_2 < t_1 \\
(WAIT\; t_1 \,;\, P) \overset{t_2}{\rhd} Q & \equiv & WAIT\; t_1 : (P \overset{t_2-t_1}{\rhd} Q) & \text{if } t_2 \geqslant t_1 \\
 & & WAIT\; (t_2+\delta) \,;\, Q & \text{if } t_2 < t_1 \\
((a \rightarrow P) \overset{t}{\rhd} Q) \setminus A & \equiv & WAIT\; \delta \,;\, (P \setminus A) & \text{if } a \in A, t > 0
\end{array}$$

3.4.2 Tireout

Another important aspect of timed behaviour is the *tireout*, where a process is allowed to run for a particular length of time, after which control is passed to a second process, providing the first has not terminated.

In order to define the tireout operator, we must first define a subsidiary operator $HALT_t$, which behaves like its argument P up until time t, but must then refuse to perform anything else (except that it may terminate successfully if this is possible for P). It is defined in the followiug fashion: giveu au argument P, axiom 5 for TM_F yields that

$$\exists\, n \bullet \forall\, s \in traces(P) \bullet end(s) \leqslant t \Rightarrow \#s \leqslant n$$

Then

$$HALT_t(P) \mathrel{\widehat{=}} P\ {}_{\Sigma}\|_{\Sigma'} (((\mathop{|||}_{i=1}^{n}(a : \Sigma' \to STOP)) \mathbin{|||} WAIT\ t) ; STOP)$$

where $\Sigma' = \Sigma \setminus \{\checkmark\}$. The semantics of $HALT_t(P)$ are therefore given by

$$\begin{array}{rcl} \mathcal{F}_T[\![HALT_t(P)]\!] & = & \{(s, \aleph) \mid end(s \setminus \checkmark) \leqslant t \wedge (s \upharpoonright t, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!] \\ & & \qquad \wedge (s, (\aleph \upharpoonright t) \cup (\aleph \cap [t, \infty) \times \{\checkmark\})) \in \mathcal{F}_T[\![P]\!] \end{array}$$

Definition 3.4.4 *The tireout operator, $\underset{t}{\lightning}$, is defined as follows:*

$$P \underset{t}{\lightning} Q \quad \widehat{=} \quad rem_i\left(\begin{array}{l} \left(\begin{array}{l} i : HALT_t(P) \\ ||| \\ WAIT\ t ; a \to SKIP \end{array} \right) ; \left(\begin{array}{l} SKIP \\ \Box \\ b \to i : Q \end{array} \right) \\ \\ {}_{\Sigma}\|_{\{a,b,\checkmark\}} \\ \\ \left(\begin{array}{l} SKIP \\ \Box \\ a \to b \to SKIP \end{array} \right) \end{array} \right) \setminus \{a, b\})$$

Remarks by A.W. Roscoe were helpful in formulating the above definition.

This construction begins with control at $i : HALT_t(P)$, and terminates successfully if $i : HALT_t(P)$ does so before time t. If $i : HALT_t(P)$ does not terminate successfully by time t, then hidden event a becomes available and so is forced to occur. After a delay of δ, termination of the construct

$$(((i : HALT_t(P)) \mathbin{|||} (WAIT\ t ; a \to SKIP))$$

becomes possible, so it is forced by the sequential composition operator: control is removed from $i : HALT_t(P)$. After another delay of δ the process passes control to $i : Q$. The construction works because $(i : HALT_t(P)) \setminus \{a, b\} \equiv i : HALT_t(P)$ and $(i : Q) \setminus \{a, b\} \equiv i : Q$.

Example

The process

$$(\mu X \bullet a \to X) \underset{10}{\lightning} \mu X \bullet b \to X$$

will recursively engage in event a for 10 time units, after which it will behave like $\mu X \bullet b \to X$, regardless of the number of as performed.

Theorem 3.4.5 *The semantic equation for the tireout operator reduces to the following:*

$$\begin{aligned}
\mathcal{F}_T[\![P \underset{t}{\lightning} Q]\!] \;=\; & \{(s,\aleph) \mid (s \upharpoonright t, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \in \mathcal{F}_T[\![P]\!] \wedge \checkmark \notin \sigma(s \upharpoonright t) \wedge \\
& \qquad s \uparrow (t, t+2\delta) = \langle\rangle \wedge (s \dot{-} (t+2\delta), \aleph \dot{-} (t+2\delta)) \in \mathcal{F}_T[\![Q]\!]\} \\
& \cup \\
& \{(s,\aleph) \mid (s^\frown\langle(t',\checkmark)\rangle, \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\} \in \mathcal{F}_T[\![P]\!] \wedge t' \leqslant t \wedge \\
& \qquad \checkmark \notin \sigma(s) \wedge \checkmark \notin \sigma(\aleph \upharpoonleft t')\} \\
& \cup \\
& \{(s^\frown\langle(t'',\checkmark)\rangle, \aleph) \mid (s^\frown\langle(t',\checkmark)\rangle, \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\}) \in \mathcal{F}_T[\![P]\!] \\
& \quad \wedge t' \leqslant t \wedge \checkmark \notin \sigma(s) \wedge t'' \geqslant t' \wedge \checkmark \notin \sigma(\aleph \uparrow [t',t''))\}
\end{aligned}$$

If it is not possible for P to terminate successfully before (or at) time t then a simpler construction is possible.

$$P \underset{t}{\lightning} Q \;\equiv\; (P \,|||\, \mathit{WAIT}\ t) \,;\, \mathit{SKIP} \,;\, Q$$

This construction fails if P is able to terminate before time t, since on its successful termination control will pass to Q, whereas we require that the whole process $P \underset{t}{\lightning} Q$ terminates successfully if P does so before control is removed. In the case where P cannot terminate successfully, only the first component of the union in the semantic definition given above is non-empty, and so the semantic equation is reduced to

$$\begin{aligned}
\mathcal{F}_T[\![P \underset{t}{\lightning} Q]\!] \;=\; & \{(s,\aleph) \mid (s \upharpoonright t, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!] \wedge s \uparrow (t, t+2\delta) = \langle\rangle \\
& \wedge (s \dot{-} (t+2\delta), \aleph \dot{-} (t+2\delta)) \in \mathcal{F}_T[\![Q]\!]\}
\end{aligned}$$

The tireout operator obeys the following laws:

$$\begin{aligned}
P \underset{t_1}{\lightning} (Q \underset{t_2}{\lightning} R) \;&\equiv\; (P \underset{t_1}{\lightning} Q) \underset{t_1+t_2+2\delta}{\lightning} R \\
(P \sqcap Q) \underset{t}{\lightning} R \;&\equiv\; (P \underset{t}{\lightning} R) \sqcap (Q \underset{t}{\lightning} R) \\
P \underset{t}{\lightning} (Q \sqcap R) \;&\equiv\; (P \underset{t}{\lightning} Q) \sqcap (P \underset{t}{\lightning} R) \\
(P \,\Box\, Q) \underset{t}{\lightning} R \;&\equiv\; (P \underset{t}{\lightning} R) \,\Box\, (Q \underset{t}{\lightning} R) \\
P \underset{t}{\lightning} (Q \,\Box\, R) \;&\equiv\; (P \underset{t}{\lightning} Q) \,\Box\, (P \underset{t}{\lightning} R)
\end{aligned}$$

$$
\begin{array}{rcll}
(P \parallel Q) \underset{t}{\lightning} (R \parallel S) & \equiv & (P \underset{t}{\lightning} R) \parallel (Q \underset{t}{\lightning} S) & \\
(P \underset{t}{\lightning} Q) \parallel (WAIT\,(t+\delta)\,; R) & \equiv & WAIT\, t+\delta\,; (Q \parallel R) & \\
(WAIT\, t_1\,; P) \underset{t_2+\delta}{\lightning} Q & \equiv & WAIT\, t_1\,; (P \underset{t_2-t_1}{\lightning} Q) & \text{if } t_2 \geqslant t_1 \\
 & & WAIT\,(t_2 + 2\delta)\,; Q & \text{if } t_2 < t_1 \\
 & & & \\
(WAIT\, t_1\,; P) \underset{t_2}{\lightning} Q & \equiv & WAIT\, t_1\,; (P \underset{t_2-t_1}{\lightning} Q) & \text{if } t_2 \geqslant t_1 \\
 & & WAIT\,(t_2 + 2\delta)\,; Q & \text{if } t_2 < t_1 \\
 & & & \\
((a \rightarrow P) \underset{t+\delta}{\lightning} Q) \setminus X & \equiv & WAIT\, \delta\,; (P \underset{t}{\lightning} Q) \setminus X & \text{if } a \in X \\
(P \overset{t_1}{\rhd} Q) \underset{t_1+t_2+\delta}{\lightning} R & \equiv & (P \underset{t_1+t_2+\delta}{\lightning} R) \overset{t_1}{\rhd} (Q \underset{t_2}{\lightning} R) & \\
(P \underset{t_1}{\lightning} Q) \underset{t_1+t_2+2\delta}{\lightning} R & \equiv & P \underset{t_1}{\lightning} (Q \underset{t_2}{\lightning} R) &
\end{array}
$$

3.4.3 Interrupts

The tireout operator passes control from one process to another at a predetermined time. There is also a need to model the situation where a special interrupt event may cause the passing of control from one process to another, known as the interrupt handler. The interrupt described here is a simple interrupt, in the sense that control cannot be passed back to the first process. It is a simple adaptation of the definition of the tireout operator: the essential alteration is that it is the occurrence of the interrupting event i, rather than the termination of a delay process, that triggers the availability of the ✓ event which removes control from P. Observe that i is always available while P is running.

We will model a process that behaves like P except that iuterrupt event i is always available before P terminates; if event i occurs, then control is removed from P and passed to the interrupt handler Q. This will be denoted $P \underset{i}{\nabla} Q$.

Definition 3.4.6 *We define the interrupt operator as follows:*

$$P \mathop{\nabla}_{i} Q \;\;\widehat{=}\;\; rem_j\left(\begin{pmatrix} \begin{pmatrix} j : P \\ ||| \\ j.i \rightarrow SKIP \end{pmatrix} ; \begin{pmatrix} SKIP \\ \Box \\ b \rightarrow j : Q \end{pmatrix} \\ \\ {}_{\Sigma}\|_{\{j.i, b, \checkmark\}} \\ \\ \begin{pmatrix} SKIP \\ \Box \\ j.i \rightarrow b \rightarrow SKIP \end{pmatrix} \end{pmatrix} \setminus \{b\}\right)$$

Observe that this will not necessarily have the desired behaviour if $i \in \sigma(P)$, since the performance of event i in that case may be due to P as opposed to the trigger process. It is therefore the responsibility of the programmer to ensure that interrupt events are distinct from the alphabet of the interruptible process.

As in the case of the tireout operator, the interrupt operator has a simpler formulation if process P cannot terminate:

$$P \mathop{\nabla}_{i} Q \;\;\equiv\;\; (P \;|||\; i \rightarrow SKIP) \,;\, Q$$

We may generalise the interrupt operator to model the case where there are a number of different interrupt events, and the behaviour of the process following an interrupt is dependent on the identity of the event causing the interrupt. We take $P \mathop{\nabla}_{i \in I} Q(i)$ to be the process which has each element of the set I available as an interrupt event while it behaves as P, and if an interrupt event i occurs, then control is passed from P to $Q(i)$. The definition is extended as follows:

Definition 3.4.7 *The generalised interrupt operator is defined as follows:*

$$P \mathop{\nabla}_{i \in I} Q \;\;\widehat{=}\;\; rem_j\left(\begin{pmatrix} \begin{pmatrix} j : P \\ ||| \\ j.i : j_I \rightarrow SKIP \end{pmatrix} ; \begin{pmatrix} SKIP \\ \Box \\ b_i : b_I \rightarrow j : Q(i) \end{pmatrix} \\ \\ {}_{\Sigma}\|_{I \cup b_I \cup \{\checkmark\}} \\ \\ \begin{pmatrix} SKIP \\ \Box \\ j.i : j_I \rightarrow b_i \rightarrow SKIP \end{pmatrix} \end{pmatrix} \setminus b_I\right)$$

where $b_I = \{b_i \mid i \in I\}$, *and* $j_I = \{j.i \mid i \in I\}$

As before, we require tbat $\sigma(P) \cap I = \emptyset$ in order to ensure that the operator yields the desired behaviour.

The construct allows interrupts to he prioritised. In the process

$$((P \mathop{\nabla}_{i \in I} Q(i)) \mathop{\nabla}_{j \in J} R(j))$$

the interrupts drawn from set J have a higher priority than those drawn from set I, in that they can interrupt the interrupt handler of a lower priority interrupt.

3.5 The After Operator

The after operator on untimed processes is defined as follows:

Definition 3.5.1 *If $s \in traces(P)$, then*

$$\mathcal{F}[\![P/s]\!] \ \ \widehat{=}\ \ \{(w, X) \mid (s^\frown w, X) \in \mathcal{F}[\![P]\!]\}$$

As we shall see, the introduction of timing information requires a more subtle treatment of the after operator.

Definition 3.5.2 *We define the* after *operator / on processes as follows: if $(s, \aleph) \in fail(\mathcal{E}_T[\![P]\!])$ and $end(s, \aleph) \leqslant t$ then*

$$\begin{aligned} \mathcal{E}_T[\![P/((s,\aleph),t)]\!] \ \ \widehat{=}\ \ & \{(u, \alpha \dot{-} t, \aleph') \mid \\ & \qquad (s^\frown(u+t), \alpha, \aleph \cup (\aleph' + t)) \in \mathcal{E}_T[\![P]\!]\} \end{aligned}$$

Otherwise $P/((s,\aleph),t)$ is undefined.

This operator is very different to the *CSP* version presented in [Ros82] and [Hoa85]. The only information an observer can obtain from an untimed process while it is executing is the trace of eveuts that have occurred, so the untimed after operator only considers the behaviour of a process after a *trace*. In timed *CSP*, we can also observe which events were refused during the performance of the trace, and this additional information may enable us to predict more accurately the future behaviour of the process. For example, consider the process

$$\begin{aligned} P \ \ \widehat{=}\ \ & a \to ((b \to STOP) \overset{5}{\rhd} b \to c \to STOP) \\ & \sqcap \\ & a \to STOP \,\Box\, c \to STOP \end{aligned}$$

If we only concern ourselves only with traces, then we cannot know, after the observation of a, whether or not event b will become available. However, the knowledge that event c was refused before the a was performed allows us to deduce that the non-deterministic choice was resolved in favour of the first process, since the second cannot refuse c before performing a. Hence the addition of relevant refusal information will allow us to deduce that b will certainly become available after the a is performed.

Even the (*trace*, *refusal*) pair corresponding to what we may have observed does not give us all the information we could use to predict possible future behaviours, since implicit in every observation of a failure $(s, \aleph)$ is a time t which corresponds to the time up to which the process has been under observation when $(s, \aleph)$ has been observed. It provides information about what has *not* been observed: no events other than those in s have been performed, and only those events in $\aleph$ have been refused, up to time t.

The time t associated with the failure $(s, \aleph)$ allows us to predict the future behaviour of the process by enabling us to deduce how much internal progress has taken place. In the process P above, we are able to deduce that event b will be available after observing the failure $(\langle(1, a)\rangle, [0, 1+\delta) \times \{c\})$, but we cannot know whether or not the timeout has occurred, and hence that c will become available after b has been performed, unless we can keep track of the internal progress of the process. The additional knowledge that nothing else has occurred (or been refused) up until time 6 allows us to infer that a c will become available after a b has been performed. This is apparent from the following equivalence:

$$P/((\langle(1, a)\rangle, [0, 1+\delta) \times \{c\}), 6) \;\equiv\; b \to c \to STOP$$

We overload the after operator in the following way:

$$\begin{aligned} P/(s, \aleph) &\mathrel{\widehat=} P/((s, \aleph), end(s, \aleph)) \\ P/s &\mathrel{\widehat=} P/((s, \emptyset), end(s)) \\ P/(s, t) &\mathrel{\widehat=} P/((s, \emptyset), t) \\ P/t &\mathrel{\widehat=} P/((\langle\rangle, \emptyset), t) \end{aligned}$$

Some of the laws given for the after operator in [Hoa85] extend naturally to include time, and new laws regarding timing behaviour are introduced:

$$\begin{aligned} P/\langle\rangle &\equiv P \\ (P/s)/u &\equiv P/s^\frown(u + end(s)) \\ (P/t_1)/t_2 &\equiv P/(t_1 + t_2) \\ (WAIT\, t\, ; P)/t &\equiv P \end{aligned}$$

$$\begin{aligned}(WAIT\,t\,;P)/t+\delta &\equiv P\\ (P/(s,\aleph))/t &\equiv P/((s,\aleph),end(s,\aleph)+t)\end{aligned}$$

3.6 Infinite behaviours

There are many cases where we would wish to consider the infinite behaviour of processes. A classic case is iu the treatment of one form of unbounded non-determinism, where two processes may be distinguished only by their infinite behaviours: for instance, we cannot distinguish in finite time between a process which can wait an arbitrary length of time and then offer event a, and a process which could also always refuse to offer a. The problem of unbounded non-determinism in untimed *CSP* has been successfully treated in [Ros88b], where infinite traces are added to the standard failures-divergence model.

Infinite behaviours are also necessary for the framing of temporal logic statements such as 'always Φ'. Indeed, part of the definition of a timed buffer, presented iu [Sch88], is an attempt to capture a statement of this kind: that if the buffer is empty, then it will eventually offer to input; and if the buffer is non-empty, then output will eventually be offered.

The semantic models presented in [Ree88] deal only with finite behaviours, in the sense that each possible behaviour of a process can be considered as the result of observing the process for a finite time: $(s,\aleph) \in \mathcal{F}_T[\![P]\!]$ tells us that if we watch P up until time $end(s,\aleph)$, tben $(s,\aleph)$ is a possible observation. This notion may be extended to infinite behaviours of P: observations that may be made if P is watched for *all* time. The infinitary nature of these observations derives from the fact that they are possible complete histories of the process, and so the observations themselves need not contain traces of infinite length nor unbounded refusals: it is perfectly possible to see only a finite number of events performed, and only a finite number of refusals, when a process is observed for all time. The discussion of the after operator made the point that a failure of a process may be considered as a possible behaviour of that process up to any time after the end of the failure. We can therefore consider it as a possible behaviour of the process observed for all time. Hence any failure of P may also be considered as an infinite failure of P.

In this thesis, we retain tbe philosophy of the timed models presented in [Ree88], that processes are completely characterised by their finite behaviours. The set of infinite behaviours of a process will therefore be completely defined by its set of finite behaviours. Hence we are not solving the problem of unbounded non-determinism mentioned above, but we are making it possible to frame some statements about processes (such as the statement that a process is a buffer) in terms of infinite behaviours. Although these statements could be translated into statements concerning the finite behaviours of the process under consideration, they will be more than convenient notational shorthand, since such statements anticipate a treatment of infinite behaviours analogous to the treatment for untimed *CSP* mentioned earlier; when such a treatment is available, we intend that our definitions and predicates are still concerned with the infinite behaviours of a process.

Any treatment of infinite failures must have that the restriction of an infinite failure to a finite time must be a failure of P. For the purposes of this thesis, we will take *any* infinite failure which has this property to be an infinite failure of P.

Definition 3.6.1 *Define the infinite traces $T\Sigma^i_\leqslant$, and infinite refusal sets $IRSET$ by*

$$T\Sigma^\omega_\leqslant \;\widehat{=}\; \{s \in ([0,\infty) \times \Sigma)^\omega \mid \forall t \in [0,\infty) \bullet s \upharpoonright t \in T\Sigma^*_\leqslant\}$$

where the definition of the before operator $\upharpoonright$ is extended in the obvious way to allow infinite traces as arguments.

$$\begin{aligned} T\Sigma^i_\leqslant &\;\widehat{=}\; T\Sigma^*_\leqslant \cup T\Sigma^\omega_\leqslant \\ IRSET &\;\widehat{=}\; \{\bigcup Z \mid Z \in \mathbf{P}(RTOK) \wedge \forall t \bullet \bigcup Z \upharpoonright t \in RSET\} \end{aligned}$$

We may then define infinite behaviours of a set of failures.

Definition 3.6.2 *If P is a set of failures, then*

$$I(P) \;\widehat{=}\; \{(s,\aleph) \mid \forall t \in [0,\infty) \bullet (s,\aleph) \upharpoonright t \in P\}$$

This gives rise to a semantic mapping $\mathcal{I}_T$, which denotes the infinite behaviours of a process.

Definition 3.6.3 *We define $\mathcal{I}_T : TCSP \rightarrow T\Sigma^i_\leqslant \times IRSET$ as follows:*

$$\mathcal{I}_T[\![P]\!] \;\widehat{=}\; \{(s,\aleph) \mid \forall t \in [0,\infty) \bullet (s,\aleph) \upharpoonright t \in \mathcal{F}_T[\![P]\!]\}$$

A particularly useful subset of a set of infinite failures is those pairs whose traces have finite length. These allow considerations of infinite refusals following a trace, which will be used when we come to define strong timewise refinement. For convenience, we will call these the semi-infinite failures.

Definition 3.6.4 *If P is a set of failures, then*

$$SI(P) \;\widehat{=}\; \{(s,\aleph) \mid \forall t \in [0,\infty) \bullet (s,\aleph) \upharpoonright t \in P \wedge \#s \textit{ is finite}\}$$

Again, this extends to a semantic mapping $\mathcal{SI}_T$:

Definition 3.6.5 *The semantic mapping $\mathcal{SI}_T : TCSP \rightarrow T\Sigma^{\checkmark}_{\leqslant} \times IRSET$ is defined as follows:*

$$\mathcal{SI}_T[P] \;\widehat{=}\; \{(s,\aleph) \mid (s,\aleph) \in \mathcal{I}_T[\![P]\!] \wedge \#s \textit{ is finite}\}$$

We obtain the equations below for infinite behaviours, by considering them as limits of finite approximations. We conjecture in section 9.3 that these equations would also hold in a model which could adequately handle all aspects of infinite behaviour.

$$\mathcal{I}_T[\![STOP]\!] \;=\; \{(\langle\rangle,\aleph) \mid \aleph \in IRSET\}$$

$$\mathcal{I}_T[\![\bot]\!] \;=\; \{(\langle\rangle,\aleph) \mid \aleph \in IRSET\}$$

$$\begin{aligned}\mathcal{I}_T[\![SKIP]\!] \;=\;& \{(\langle\rangle,\aleph) \mid \checkmark \notin \sigma(\aleph)\}\\ &\cup\{((\langle()t,\checkmark),\aleph) \mid \checkmark \notin \sigma(\aleph \upharpoonright t)\}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_T[\![WAIT\ t]\!] \;=\;& \{(\langle\rangle,\aleph) \mid \checkmark \notin \sigma(\aleph)\}\\ &\cup\{((\langle()t',\checkmark),\aleph) \mid \checkmark \notin \sigma(\aleph \uparrow [t,t'))\}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_T[\![a \rightarrow P]\!] \;=\;& \{(\langle\rangle,\aleph) \mid a \notin \sigma(\aleph)\}\\ &\{(\langle(t,a)\rangle^\frown s,\aleph) \mid a \notin \sigma(\aleph \upharpoonright t) \wedge\\ &\qquad (s-(t+\delta),\aleph \dot{-} (t+\delta)) \in \mathcal{I}_T[\![P]\!]\}\end{aligned}$$

$$\begin{aligned}\mathcal{I}_T[\![a : A \rightarrow P(a)]\!] \;=\;& \{(\langle\rangle,\aleph) \mid A \cap \sigma(\aleph) = \emptyset\\ &\cup\\ &\{(\langle(t,a)\rangle^\frown s,\aleph) \mid A \cap \sigma(\aleph \upharpoonright t) = \emptyset\\ &\quad \wedge (s-(t+\delta),\aleph \dot{-} (t+\delta)) \in \mathcal{I}_T[\![P(a)]\!]\}\end{aligned}$$

$$\mathcal{I}_T[\![P \sqcap Q]\!] \;=\; \mathcal{I}_T[\![P]\!] \cup \mathcal{I}_T[\![Q]\!]$$

$$\begin{aligned}
\mathcal{I}_T[\![P \mathbin{\Box} Q]\!] &= \{(s,\aleph) \mid (\langle\rangle, \aleph \upharpoonright begin(s)) \in \mathcal{I}_T[\![P]\!] \cap \mathcal{I}_T[\![Q]\!] \\
&\qquad \wedge (s.\aleph) \in \mathcal{I}_T[\![P]\!] \cup \mathcal{I}_T[\![Q]\!]\}
\end{aligned}$$

$$\mathcal{I}_T[\![P \parallel Q]\!] = \{(s, \aleph_1 \cup \aleph_2) \mid (s, \aleph_1) \in \mathcal{I}_T[\![P]\!] \wedge (s, \aleph_2) \in \mathcal{I}_T[\![Q]\!]\}$$

$$\begin{aligned}
\mathcal{I}_T[\![P\ {}_X\!\parallel_Y Q]\!] &= \{(s, \aleph_1 \cup \aleph_2) \mid (\aleph_1 \cup \aleph_2) \upharpoonright (X \cup Y) = \aleph_1 \upharpoonright X \cup \aleph_2 \upharpoonright Y \\
&\qquad \wedge (s \upharpoonright X, \aleph_1 \upharpoonright X) \in \mathcal{I}_T[\![P]\!] \\
&\qquad \wedge (s \upharpoonright Y, \aleph_1 \upharpoonright Y) \in \mathcal{I}_T[\![Q]\!] \\
&\qquad \wedge s \upharpoonright X \cup Y = s\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{I}_T[\![P \mathbin{|||} Q]\!] &= \{(s,\aleph) \mid \exists\, s_1, s_2 \bullet s \in s_1 \mathbin{|||} s_2 \\
&\qquad \wedge (s_1, \aleph) \in \mathcal{I}_T[\![P]\!] \wedge (s_2, \aleph) \in \mathcal{I}_T[\![Q]\!]
\end{aligned}$$

$$\mathcal{I}_T[\![WAIT\ t\,; P]\!] = \{(s + t, \aleph) \mid (s, \aleph \dot{-} t) \in \mathcal{I}_T[\![P]\!]\}$$

$$\begin{aligned}
\mathcal{I}_T[\![P \,;\, Q]\!] &= \{(s,\aleph) \mid (s, \aleph \cup [0,\infty) \times \{\checkmark\}) \in \mathcal{I}_T[\![P]\!] \wedge \checkmark \notin \sigma(s)\} \\
&\quad \cup \\
&\quad \{(s^\frown(w + (t+\delta)), \aleph) \mid (w, \aleph \dot{-} (t+\delta)) \in \mathcal{I}_T[\![P]\!] \\
&\qquad \wedge \checkmark \notin \sigma(s) \wedge (s^\frown\langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \in \mathcal{I}_T[\![P]\!]\}
\end{aligned}$$

$$\mathcal{I}_T[\![f^{-1}(P)]\!] = \{(s,\aleph) \mid (f(s), f(\aleph)) \in \mathcal{I}_T[\![P]\!]\}$$

Observe that the equations for hiding, infinite nondeterministic choice, alphabet transformation, and recursion have been omitted. We will see in section 9.3 that we would expect these to be different when all aspects of infinite behaviour can be adequately modelled.

The following theorem allows the derivation of semi-infinite behaviours from (finite) stable behaviours.

Theorem 3.6.6 *If* $(s, \alpha, \aleph) \in \mathcal{E}_T[\![P]\!]$ *then* $(s, [\alpha, \infty) \times \sigma(\aleph \upharpoonleft \alpha)) \in \mathcal{SI}_T[\![P]\!]$.

Proof This follows immediately from axiom 12 of TM_{FS}. □

4 Factorising Proofs

In this chapter we present the first of our verification metbods. We see that the semantic equations for TM_F naturally give rise to a compositional proof system for a particular class of specification, *behavioural* ones. The advantages of compositionality in a proof system are well-known: it supports the modular development of systems, and permits factorisation of the verification task.

Behavioural specifications capture a wide range of predicates on processes, including many of the specifications defined in Chapter 5. The proof system is therefore applicable to verifications that processes have these properties. In Chapter 8 we will see detailed examples of the use of behavioural specifications in the capture of system requirements, and of the application of the proof system.

4.1 Behavioural Specifications

Reed [Ree88] defines a *specification* on TM_{FS} to be a mapping from the complete metric space TM_{FS} to the set $\{TRUE, FALSE\}$. We think of S as a predicate on TM_{FS} processes, such that S holds of P if and only if $S(P) = TRUE$. Specifications on the other semantic models may be similarly characterised as mappings from those models to $\{TRUE, FALSE\}$.

We define a *behavioural specification* on TM_{FS} to be a predicate on the set of triples $(s, \alpha, \aleph)$ underlying TM_{FS}, $T\Sigma^*_\leqslant \times [0,\infty] \times RSET$. Similarly, a behavioural specification on TM_F is a predicate on $T\Sigma^*_\leqslant \times RSET$; a behavioural specification on TM_S is a predicate on $T\hat{\Sigma}^*_\leqslant \times [0,\infty]$; and a behavioural specification on TM_T is a predicate on $T\hat{\Sigma}^*_\leqslant$. We use the convention that the argument to a behavioural specification identifies the model employed, so for example $S(s,\alpha,\aleph)$ is a behavioural specification on TM_{FS}. Ohserve that, in this thesis, we use a many-sorted first order predicate calculus with equality as our assertion language.

We define the satisfaction relation **sat** between a $TCSP$ process P and a behavioural specification S to hold if and only if the specification holds of every element of the semantics of P in the model identified by S:

Definition 4.1.1

$$\begin{aligned}
P \ \mathbf{sat}\ S(s,\alpha,\aleph) &\mathrel{\widehat{=}} \forall (s,\alpha,\aleph) \in \mathcal{E}_T[\![P]\!] \bullet S(s,\alpha,\aleph)\\
P \ \mathbf{sat}\ S(s,\alpha,X) &\mathrel{\widehat{=}} \forall (s,\alpha,X) \in \mathcal{E}^*_T[\![P]\!] \bullet S(s,\alpha,X)\\
P \ \mathbf{sat}\ S(s,\aleph) &\mathrel{\widehat{=}} \forall (s,\aleph) \in \mathcal{F}_T[\![P]\!] \bullet S(s,\aleph)\\
P \ \mathbf{sat}\ S(s,\alpha) &\mathrel{\widehat{=}} \forall (s,\alpha) \in \mathcal{S}_T[\![P]\!] \bullet S(s,\alpha)
\end{aligned}$$

$$P \textbf{ sat } S(s) \;\; \widehat{=} \;\; \forall s \in \mathcal{T}_T[\![P]\!] \bullet S(s)$$

In each model, every behavioural specification has a corresponding specification on processes. In TM_{FS} the specification $S(s, \alpha, \aleph)$ corresponds to the predicate U_S on processes, where $U_S(P) = T \Leftrightarrow \forall (s, \alpha, \aleph) \in \mathcal{E}_T[\![P]\!] \bullet S(s, \alpha, \aleph)$. In Reed's notation we obtain $Pred_{U_S} = S$. However, not every specification on processes has a corresponding behavioural specification: for example, the specification $(\langle(1, a)\rangle, 2, \emptyset) \in \mathcal{E}_T[\![P]\!]$ on process P cannot be written with a behavioural specification and the **sat** relation.

The nature of behavioural specifications gives rise to a set of inference rules which allow us to deduce properties of the behaviours of composite processes from properties of the behaviours of the syntactic subcomponents. The rules can be derived directly from the clauses of the semantic equation, and so there is a law for each syntactic construct. The set of rules for TM_F allows us to reduce the proof obligation on *any* composite *TCSP* process to proof obligations on its component processes. The cases where stability values are included in the model are not so straightforward, so we will first present the rules for behavioural specifications on TM_F.

4.2 The proof system for TM_F

The rules for the basic processes are as follows:

Rule *STOP*

$$\frac{(s = \langle\rangle) \Rightarrow S(s, \aleph)}{STOP \textbf{ sat } S(s, \aleph)}$$

Rule $\perp$

$$\frac{(s = \langle\rangle) \Rightarrow S(s, \aleph)}{\perp \textbf{ sat } S(s, \aleph)}$$

Rule $SKIP$

$$\frac{\left.\begin{array}{l}(s = \langle\rangle \ \wedge \checkmark \notin \sigma(\aleph)) \\ \vee \\ (s = \langle(t, \checkmark)\rangle \wedge \checkmark \notin \sigma(\aleph \restriction t) \wedge t \geqslant 0)\end{array}\right\} \Rightarrow S(s,\aleph)}{SKIP \ \textbf{sat} \ S(s,\aleph)}$$

Rule $WAIT\ t$

$$\frac{\left.\begin{array}{l}s = \langle\rangle \ \wedge \checkmark \notin \sigma(\aleph \uparrow t) \\ \vee \\ s = \langle(t', \checkmark)\rangle \wedge t' \geqslant t \wedge \checkmark \notin \sigma(\aleph \uparrow [t, t'))\end{array}\right\} \Rightarrow S(s,\aleph)}{WAIT\ t \ \textbf{sat} \ S(s,\aleph)}$$

The rules for the more complex operators are written to enable reduction of proof obligations. The form of tbe conclusion of each rule matches the form of a proof requirement on a composite process: in order to prove it, we need only find behavioural specifications S_1 and S_2, or in some cases just S_1, such that the premises of the inference rule hold. We will have then reduced an obligation on a composite process to requirements on its syntactic subcomponents and a logical proposition.

Rule $a \rightarrow P$

$$\frac{\begin{array}{l}P \ \textbf{sat} \ S_1(s,\aleph) \\ s = \langle\rangle \wedge a \notin \sigma(\aleph) \Rightarrow S(s,\aleph) \\ \left.\begin{array}{l}s = \langle(t, a)\rangle^\frown s' \wedge a \notin \sigma(\aleph \restriction t) \wedge begin(s') \geqslant t + \delta \\ \wedge \ S_1(s' \dot{-} (t+\delta), (\aleph \dot{-} (t+\delta)))\end{array}\right\} \Rightarrow S(s,\aleph)\end{array}}{(a \rightarrow P) \ \textbf{sat} \ S(s,\aleph)}$$

Rule $a : A \rightarrow P_a$

$$\frac{\begin{array}{l}\forall a \in A \bullet P_a \ \textbf{sat} \ S_a(s,\aleph) \\ s = \langle\rangle \wedge \sigma(\aleph) \cap A = \emptyset \Rightarrow S(s,\aleph) \\ \left.\begin{array}{l}s = \langle(t, a)\rangle^\frown s') \wedge begin(s') \geqslant t + \delta \\ \wedge \ S_a(s' \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))\end{array}\right\} \Rightarrow S(s,\aleph)\end{array}}{a : A \rightarrow P_a \ \textbf{sat} \ S(s,\aleph)}$$

Rule $P_1 \sqcap P_2$

$$\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ S_1(s,\aleph) \vee S_2(s,\aleph) \Rightarrow S(s,\aleph) \\ \hline P_1 \sqcap P_2 \ \mathbf{sat} \ S(s,\aleph) \end{array}$$

Rule $\bigsqcap_{i \in I} P_i$

$$\begin{array}{l} \forall\, i \in I \bullet P_i \ \mathbf{sat} \ S_i(s,\aleph) \\ \forall\, i \in I \bullet (S_i(s,\aleph) \Rightarrow S(s,\aleph)) \\ \hline \bigsqcap_{i \in I} P_i \ \mathbf{sat} \ S(s,\aleph) \end{array}$$

Rule $P_1 \,\Box\, P_2$

$$\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ \left.\begin{array}{l} (S_1(s,\aleph) \vee S_2(s,\aleph) \\ \wedge\, S_1(\langle\rangle, \aleph \restriction begin(s)) \wedge S_2(\langle\rangle, \aleph \restriction begin(s)) \end{array}\right\} \Rightarrow S(s,\aleph) \\ \hline P_1 \,\Box\, P_2 \ \mathbf{sat} \ S(s,\aleph) \end{array}$$

Rule $P_1 \parallel P_2$

$$\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ S_1(s,\aleph_1) \wedge S_2(s,\aleph_2) \Rightarrow S(s,\aleph_1 \cup \aleph_2) \\ \hline P_1 \parallel P_2 \ \mathbf{sat} \ S(s,\aleph) \end{array}$$

Rule $P_1 \ {}_X\!\parallel_Y P_2$

$$\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ \left.\begin{array}{l} \sigma(s_1,\aleph_1) \subseteq X \wedge \sigma(s_2,\aleph_2) \subseteq Y \\ \wedge\, \sigma(\aleph_3) \subseteq \Sigma - (X \cup Y) \\ \wedge\, S_1(s_1,\aleph_1) \wedge S_2(s_2,\aleph_2) \wedge s_3 \in s_1 \ {}_X\!\parallel_Y s_2) \end{array}\right\} \Rightarrow S(s_3, \aleph_1 \cup \aleph_2 \cup \aleph_3) \\ \hline P_1 \ {}_X\!\parallel_Y P_2 \ \mathbf{sat} \ S(s,\aleph) \end{array}$$

Rule $P_1 \mid\mid\mid P_2$

$$\frac{\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ (s \in \mathit{Tmerge}(s_1,s_2) \wedge S_1(s_1,\aleph) \wedge S_2(s_2,\aleph)) \Rightarrow S(s,\aleph) \end{array}}{P_1 \mid\mid\mid P_2 \ \mathbf{sat} \ S(s,\aleph)}$$

Rule $\mathit{WAIT}\ t\,;P$

$$\frac{\begin{array}{l} P \ \mathbf{sat} \ S_1(s,\aleph) \\ (S_1(s \mathbin{\dot{-}} t, \aleph \mathbin{\dot{-}} t) \wedge \mathit{begin}(s) \geqslant t) \Rightarrow S(s,\aleph) \end{array}}{\mathit{WAIT}\ t\,; P \ \mathbf{sat} \ S(s,\aleph)}$$

Rule $P_1 \fatsemi P_2$

$$\frac{\begin{array}{l} P_1 \ \mathbf{sat} \ S_1(s,\aleph) \\ P_2 \ \mathbf{sat} \ S_2(s,\aleph) \\ (\checkmark \notin \sigma(s) \wedge \forall I \in \mathit{TINT} \bullet S_1(s, \aleph \cup (I \times \{\checkmark\}))) \Rightarrow S(s,\aleph) \\ \left.\begin{array}{l} (S_1(s \upharpoonright t \frown \langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \wedge \\ s \uparrow (t, t+\delta) = \langle\rangle \wedge S_2(s \mathbin{\dot{-}} (t+\delta), \aleph \mathbin{\dot{-}} (t+\delta)) \end{array}\right\} \Rightarrow S(s,\aleph) \end{array}}{P_1 \fatsemi P_2 \ \mathbf{sat} \ S(s,\aleph)}$$

Rule $P \setminus A$

$$\frac{\begin{array}{l} P \ \mathbf{sat} \ S_1(s,\aleph) \\ (S_1(s,\aleph) \wedge \mathbf{act}(A)(s,\aleph) \wedge \sigma(\aleph') \subseteq A) \Rightarrow S(s \setminus A, \aleph - \aleph') \end{array}}{P \setminus A \ \mathbf{sat} \ S(s,\aleph)}$$

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

$$\frac{\begin{array}{l} P \ \mathbf{sat} \ S_1(s,\aleph) \\ S_1(f(s), f(\aleph)) \Rightarrow S(s,\aleph) \end{array}}{f^{-1}(P) \ \mathbf{sat} \ S(s,\aleph)}$$

Rule $f(P)$

$$\begin{array}{l} P \ \mathbf{sat}\ S_1(s,\aleph) \\ S_1(s,f^{-1}(\aleph)) \Rightarrow S(f(s),\aleph) \\ \hline f(P)\ \mathbf{sat}\ S(s,\aleph) \end{array}$$

Rule $\mu X \bullet F(X)$

$$\begin{array}{l} \forall X \bullet X \ \mathbf{sat}\ S_1(s,\aleph) \Rightarrow F(\mathit{WAIT}\,\delta\,; X)\ \mathbf{sat}\ S_1(s,\aleph) \\ S_1(s,\aleph) \Rightarrow S(s,\aleph) \\ \hline \mu X \bullet F(X)\ \mathbf{sat}\ S(s,\aleph) \end{array}$$

Observe in the last rule that the variable X in the first antecedent ranges over *all* sets of failures, not only those sets that happen to be elements of TM_F. This means that we cannot assume that the axioms for TM_F hold of X. If we wish to use this assumption in establishing that $F(\mathit{WAIT}\,\delta\,; X)$ preserves S_1, then we must also establish that S_1 is satisfiable:

Rule $\mu X \bullet F(X)$ alternative version

$$\begin{array}{l} \forall X : TM_F \bullet (X \ \mathbf{sat}\ S_1(s,\aleph) \Rightarrow \\ \qquad\qquad\qquad F(\mathit{WAIT}\,\delta\,; X)\ \mathbf{sat}\ S_1(s,\aleph)) \\ S_1(s,\aleph) \Rightarrow S(s,\aleph) \\ \hline \mu X \bullet F(X)\ \mathbf{sat}\ S(s,\aleph) \end{array} \quad [\, \exists P : TM_F \bullet P \ \mathbf{sat}\ S_1(s,\aleph)\,]$$

We provide the rule for instantaneous sequential composition. We also provide rules (which may be derived from the basic rules) for some of the operators defined in terms of the basic *TCSP* operators.

Rule $P_1\,;P_2$

$$\begin{array}{l} P_1 \ \mathbf{sat}\ S_1(s,\aleph) \\ P_2 \ \mathbf{sat}\ S_2(s,\aleph) \\ \checkmark \notin \sigma(s) \wedge \forall I \in TINT \bullet S_1(s,\aleph \cup (I \times \{\checkmark\})) \Rightarrow S(s,\aleph) \\ \left.\begin{array}{l} s \cong s_1 {}^\frown (s_2 + t) \wedge \checkmark \notin \sigma(s_1) \wedge end(\aleph_1) \leqslant t \wedge \\ S_1(s_1 {}^\frown \langle (t,\checkmark)\rangle, \aleph_1 \cup ([0,t) \times \{\checkmark\})) \wedge S_2(s_2,\aleph_2) \end{array}\right\} \Rightarrow S(s,\aleph_1 \cup (\aleph_2 + t)) \\ \hline (P_1\,;P_2)\ \mathbf{sat}\ S(s,\aleph) \end{array}$$

Rule $P_1 \overset{t}{\triangleright} P_2$

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ \ S_1(s,\aleph) \\
P_2 \ \mathbf{sat}\ \ S_2(s,\aleph) \\
\mathit{begin}(s) \leqslant t \wedge S_1(s,\aleph) \Rightarrow S(s,\aleph) \\
(\mathit{begin}(s) \geqslant t+\delta \wedge S_1(\langle\rangle, \aleph \upharpoonright t) \wedge S_2(s \mathbin{\dot{-}} (t+\delta), \aleph \mathbin{\dot{-}} (t+\delta))) \Rightarrow S(s,\aleph) \\
\hline
P_1 \overset{t}{\triangleright} P_2 \ \mathbf{sat}\ S(s,\aleph)
\end{array}$$

Rule $P_1 \underset{t}{↯} P_2$

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ \ S_1(s,\aleph) \\
P_2 \ \mathbf{sat}\ \ S_2(s,\aleph) \\
\left.\begin{array}{l} S_1(s \upharpoonright t, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \wedge \checkmark \notin \sigma(s \upharpoonright t) \wedge \\ s \uparrow (t, t+2\delta) = \langle\rangle \wedge S_2(s \mathbin{\dot{-}} (t+2\delta), \aleph \mathbin{\dot{-}} (t+2\delta)) \end{array}\right\} \Rightarrow S(s,\aleph) \\
\left.\begin{array}{l} S_1(s^\frown\langle(t',\checkmark)\rangle, \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\} \wedge t' \leqslant t \wedge \\ \checkmark \notin \sigma(s) \wedge \checkmark \notin \sigma(\aleph \uparrow t')\} \end{array}\right\} \Rightarrow S(s,\aleph) \\
\left.\begin{array}{l} s = s'^\frown\langle(t'',\checkmark)\rangle \wedge S_1(s'^\frown\langle(t',\checkmark)\rangle), \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\}) \\ \wedge\, t' \leqslant t \wedge t' \leqslant t'' \wedge \checkmark \notin \sigma(s) \wedge \checkmark \notin \sigma(\aleph \uparrow [t',t'')) \end{array}\right\} \Rightarrow S(s,\aleph) \\
\hline
P_1 \underset{t}{↯} P_2 \ \mathbf{sat}\ S(s,\aleph)
\end{array}$$

The rule for tireout simplifies considerably if the first process is unable to terminate. In this case, we have $\checkmark \notin \sigma(P_1)$ (which is expressible as a behavioural specification).

Rule $P_1 \underset{t}{↯} P_2$ special case version

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ \checkmark \notin \sigma(s) \\
P_1 \ \mathbf{sat}\ \ S_1(s,\aleph) \\
P_2 \ \mathbf{sat}\ \ S_2(s,\aleph) \\
\left.\begin{array}{l} S_1(s \upharpoonright t, \aleph \upharpoonright t) \wedge s \uparrow (t, t+2\delta) = \langle\rangle \\ \wedge\, S_2(s \mathbin{\dot{-}} (t+2\delta), \aleph \mathbin{\dot{-}} (t+2\delta)) \end{array}\right\} \Rightarrow S(s,\aleph) \\
\hline
P_1 \underset{t}{↯} P_2 \ \mathbf{sat}\ S(s,\aleph)
\end{array}$$

4.3 Soundness

The basic processes

The semantics of each of the processes $STOP$, $\bot$, $SKIP$ and $WAIT\,t$ may be written in the form

$$\mathcal{F}_T[\![P]\!] \quad = \quad \{b \mid T(b)\}$$

for the appropriate predicate T on behaviours, (*trace,refusal*) pairs b. The corresponding law is of the form

Rule

$$\frac{T(b) \Rightarrow S(b)}{P \ \mathbf{sat}\ S(b)}$$

The soundness of the law in each case follows from the fact that T holds of each behaviour in P: $\forall\, b \in \mathcal{F}_T[\![P]\!] \bullet T(b)$. Since we have $T(b) \Rightarrow S(b)$ as a premiss, we conclude that $\forall\, b \in \mathcal{F}_T[\![P]\!] \bullet S(b)$, which may be written as P **sat** $S(b)$.

One place operators

The semantics of each of the one-place operators on processes, $a \longrightarrow P$, $WAIT\,t;P$, $P \setminus A$, $f(P)$, and $f^{-1}(P)$ may be written in the form

$$\mathcal{F}_T[\![\bigodot_1 P]\!] \quad = \quad \{b \mid T'(b)\} \cup \{C(b) \mid f(b) \in \mathcal{F}_T[\![P]\!] \wedge T(b)\}$$

and the corresponding inference rule is of the form

$$\frac{\begin{array}{l} P \ \mathbf{sat}\ S_1(b) \\ T'(b) \Rightarrow S(b) \\ (S_1(f(b)) \wedge T(b)) \Rightarrow S(C(b)) \end{array}}{\bigodot_1 P \ \mathbf{sat}\ S(b)}$$

In the cases where $\forall\, b \bullet T'(b) = F$ (i.e. all those except $a \rightarrow P$), the antecedent $T'(b) \Rightarrow S(b)$ is vacuous, and has been dropped from the rule. The soundness proof of this rule runs as follows:

$$\begin{array}{lrcl}
& \mathcal{F}_T[\![\odot_1 P]\!] & = & \{b \mid T'(b)\} \cup \{C(b) \mid f(b) \in \mathcal{F}_T[\![P]\!] \wedge T(b)\} \\
& P & \mathsf{sat} & S_1(b) \\
& T'(b) & \Rightarrow & S(b) \\
& (S_1(f(b)) \wedge T(b)) & \Rightarrow & S(C(b)) \\
\vdash & b' \in \mathcal{F}_T[\![\odot_1 P]\!] & \Rightarrow & b' \in \{b \mid T'(b)\} \\
& & & \vee \\
& & & b' \in \{C(b) \mid f(b) \in \mathcal{F}_T[\![P]\!] \wedge T(b)\} \\
\vdash & b' \in \mathcal{F}_T[\![\odot_1 P]\!] & \Rightarrow & T'(b') \vee \exists\, b \bullet (b' = C(b) \wedge f(b) \in \mathcal{F}_T[\![P]\!] \wedge T(b)) \\
\vdash & b' \in \mathcal{F}_T[\![\odot_1 P]\!] & \Rightarrow & S(b') \vee \exists\, b \bullet (b' = C(b) \wedge S_1(f(b)) \wedge T(b)) \\
\vdash & b' \in \mathcal{F}_T[\![\odot_1 P]\!] & \Rightarrow & S(b') \vee \exists\, b \bullet (b' = C(b) \wedge S(C(b))) \\
\vdash & \forall\, b' \in \mathcal{F}_T[\![\odot_1 P]\!] & \bullet & S(b') \vee \exists\, b \bullet S(b') \\
\vdash & \odot_1 P & \mathsf{sat} & S(b)
\end{array}$$

Two place operators

All the semantic definitions for TM_F for the two place $TCSP$ operators $P_1 \sqcap P_2$, $P_1 \Box P_2$, $P_1 \parallel P_2$, $P_1 \;{}_X\|_Y\; P_2$, $P_1 \mathbin{|||} P_2$, $P_1 \mathbin{\boldsymbol{;}} P_2$, and also those for the two place operators $P_1 \,;\, P_2$, $P_1 \overset{t}{\rhd} P_2$, $P_1 \underset{t}{\lightning} P_2$, may be written in the form

$$\mathcal{F}_T[\![P_1 \odot_2 P_2]\!] \;=\; \{C(b) \mid f_1(b_1) \in \mathcal{F}_T[\![P_1]\!] \wedge f_2(b_2) \in \mathcal{F}_T[\![P_2]\!] \wedge R(b, b_1, b_2)\}$$

The corresponding rule for operator $\odot_2$ is:

$$\frac{\begin{array}{l} P_1 \ \mathbf{sat}\ S_1(b) \\ P_2 \ \mathbf{sat}\ S_2(b) \\ (S_1(b_1) \wedge S_2(b_2) \wedge R(b, b_1, b_2)) \Rightarrow S(C(b)) \end{array}}{P_1 \odot_2 P_2 \ \mathsf{sat}\ S(b)}$$

and the soundness proof of this rule runs as follows:

$$\begin{array}{rrcl}
 & \mathcal{F}_T\llbracket P_1 \odot_2 P_2 \rrbracket & = & \{C(b) \mid f_1(b_1) \in \mathcal{F}_T\llbracket P_1 \rrbracket \wedge f_2(b_2) \in \mathcal{F}_T\llbracket P_2 \rrbracket \\
 & & & \quad \wedge\ R(b, b_1, b_2)\} \\
 & P_1 & \textbf{sat} & S_1(b) \\
 & P_2 & \textbf{sat} & S_2(b) \\
 & (S_1(b_1) \wedge S_2(b_2) & & \\
 & \wedge\ R(b, b_1, b_2)) & \Rightarrow & S(C(b)) \\
\vdash & b' \in \mathcal{F}_T\llbracket P_1 \odot_2 P_2 \rrbracket & \Rightarrow & \exists\, b, b_1, b_2 \bullet (b' = C(b) \wedge f_1(b_1) \in \mathcal{F}_T\llbracket P_1 \rrbracket \wedge \\
 & & & \qquad \in \mathcal{F}_T\llbracket P_2 \rrbracket\ \wedge R(b, b_1, b_2)) \\
\vdash & b' \in \mathcal{F}_T\llbracket P_1 \odot_2 P_2 \rrbracket & \Rightarrow & \exists\, b, b_1, b_2 \bullet (b' = C(b) \wedge S_1(b_1) \wedge S_2(b_2) \\
 & & & \qquad \wedge\ R(b, b_1, b_2)) \\
\vdash & \forall\, b' \in \mathcal{F}_T\llbracket P_1 \odot_2 P_2 \rrbracket & \bullet & \exists\, b, b_1, b_2 \bullet (b' = C(b) \wedge S(C(b))) \\
\vdash & \forall\, b' \in \mathcal{F}_T\llbracket P_1 \odot_2 P_2 \rrbracket & \bullet & S(b') \\
\vdash & P_1 \odot_2 P_2 & \textbf{sat} & S(b)
\end{array}$$

Hence the rule for each two place operator is sound.

Indexed operators

The semantics for the two arbitrary choice operators $\bigsqcap_{a \in A} P_a$ and $a : A \rightarrow P_a$ may both he written in the form

$$\mathcal{F}_T\llbracket \bigodot_{a \in A} P_a \rrbracket \;=\; \{b \mid T'(b)\} \cup \{C(b) \mid \exists\, a \in A \bullet f(b) \in \mathcal{F}_T\llbracket P_a \rrbracket \wedge T(b)\}$$

and the corresponding proof rule is given by

$$\begin{array}{l}
\forall\, a \in A \bullet P_a \ \textbf{sat}\ S_a(b) \\
T'(b) \Rightarrow S(b) \\
\forall\, a \in A \bullet (S_a(f(b)) \wedge T(b) \Rightarrow S(C(b))) \\
\hline
\odot_{a \in A} P_a \ \textbf{sat}\ S(b)
\end{array}$$

In the case of the nondeterministic choice we have $\forall\, b \bullet T'(b) = F$, and so $T'(b) \Rightarrow S(b)$ is vacuous, and has therefore been dropped from the premisses of that rule. The soundness proof of each rule runs as follows:

$$\begin{array}{lrcl}
 & \mathcal{F}_T \llbracket \bigodot_{a \in A} P_a \rrbracket & = & \{b \mid T'(b)\} \cup \\
 & & & \{C(b) \mid \exists\, a \in A \bullet f(b) \in \mathcal{F}_T\llbracket P_a \rrbracket \wedge T(b)\} \\
 & \forall\, a \in A & \bullet & P_a \ \mathbf{sat}\ S_a(b) \\
 & T'(b) & \Rightarrow & S(b) \\
 & \forall\, a \in A & \bullet & S_a(b) \wedge T(b) \Rightarrow S(C(b)) \\
\vdash & b' \in \mathcal{F}_T \llbracket \bigodot_{a \in A} P_a \rrbracket & \Rightarrow & b' = C(b) \wedge f(b) \in \mathcal{F}_T\llbracket P_a \rrbracket \wedge T(b) \\
\vdash & b' \in \mathcal{F}_T \llbracket \bigodot_{a \in A} P_a \rrbracket & \Rightarrow & S(b') \vee \exists\, a, b \bullet b' = C(b) \wedge \\
 & & & \qquad S_a(f(b)) \in \mathcal{F}_T\llbracket P_a \rrbracket \wedge T(b) \\
\vdash & b' \in \mathcal{F}_T \llbracket \bigodot_{a \in A} P_a \rrbracket & \Rightarrow & S(b') \vee \exists\, a, b \bullet b' = C(b) \wedge S(C(b)) \\
\vdash & \forall\, b' \in \mathcal{F}_T \llbracket \bigodot_{a \in A} P_a \rrbracket & \bullet & S(b') \\
\vdash & \bigodot_{a \in A} P_a & \mathbf{sat} & S(b)
\end{array}$$

Hence the rule for each of the indexed operators is sound.

Recursion

The recursion induction theorem for Timed CSP states that if $\hat{C} : TM_F \rightarrow TM_F$ is a contraction mapping, and S is a continuous satisfiable specification such that $S(P) \Rightarrow S(\hat{C}(P))$, then $S(\mu\, X \bullet F(X))$. The rule for recursion is a particular instance of the recursion induction theorem, where we restrict our attention to behavioural specifications.

We first observe that all behavioural specifications are continuous or closed in TM_F. To prove this, we will use theorem 9.6.3 from [Ree88] This tells us that a specification S on TM_F is closed if

$$\forall\, P \in TM_F \bullet (\neg S(P) \rightarrow (\exists\, t \in [0, \infty) \bullet Q(t) = P(t) \Rightarrow \neg S(Q))$$

The proof that behavioural specifications are closed then runs as follows:

$$\begin{array}{ll}
 & \neg(P \ \mathbf{sat}\ S(s, \aleph)) \\
\vdash & \exists (s, \aleph) \in P \bullet \neg S(s, \aleph) \\
\vdash & P(end(s, \aleph) + 1) = Q(end(s, \aleph) + 1) \Rightarrow (s, \aleph) \in Q \\
\vdash & \neg(Q \ \mathbf{sat}\ S(s, \aleph)) \\
\Box &
\end{array}$$

The second recursion rule given above follows automatically.

In order to drop satisfiability of S from the premisses of the recursion induction, to yield the first recursion rule given above, we extend the syntax of $TCSP$ and the model TM_F. Define $\overline{TF}$ to be the subsets of $T\Sigma_{\leq}^{*} \times RSET$. The metric on $\overline{TF}$ is defined exactly the same way as the metric on TM_F, given in appendix B. Then $\overline{TF}$ is an extension of TM_F. Define $TCSP^{+} = TCSP \mid X_E$, where E is drawn from the set $\overline{TF}$. We extend the function $\mathcal{F}_T$ to the function $\overline{\mathcal{F}}_T$ as follows:

$$\overline{\mathcal{F}}_T[\![X_E]\!] \;\widehat{=}\; E$$

The clauses for $\overline{\mathcal{F}}_T$ on the other $TCSP^{+}$ combinators are entirely similar to the defining clauses for $\mathcal{F}_T$ on $TCSP$, with every occurrence of $\mathcal{F}_T[\![P]\!]$ replaced by $\overline{\mathcal{F}}_T[\![P]\!]$. We may therefore conclude that the inference rules for the $TCSP$ operators in TM_F are also sound for $\overline{TF}$, and any behavioural specification on $\overline{TF}$ will be closed in the metric.

Now any function F composed without any $TCSP^{+}$ operators of the form X_E corresponds to a mapping C on $\overline{TF}$. The restriction of C to TM_F is the mapping on TM_F corresponding to F considered as a $TCSP$ function. Hence if C is a contraction mapping on $\overline{TF}$ and it maps processes in TM_F to processes in TM_F, then its restriction to TM_F will be a contraction mapping on TM_F, and so the fixed point of C will be in TM_F (since TM_F is a complete metric space). Now any function F composed only of $TCSP$ operators will map TM_F into itself, so the fixed point in $\overline{TF}$ of the contraction mapping $\hat{C}$ corresponding to $WAIT\,\delta\,; F$ will be in TM_F.

By the recursion induction theorem for $\overline{TF}$, in order to demonstrate that $\mu\, X \bullet F(X)$ **sat** $S(s,\aleph)$, we need only show that the specification " **sat** $S(s,\aleph)$" is continuous, satisfiable, and that X **sat** $S(s,\aleph) \Rightarrow WAIT\,\delta; F(X)$ **sat** $S(s,\aleph)$. We have already shown that any behavioural specification gives rise to a closed specification on $\overline{TF}$. It is also trivially true that $X_{\emptyset}$ **sat** $S(s,\aleph)$, since $\overline{\mathcal{F}}_T[\![X_{\emptyset}]\!] = \emptyset$. Hence any behavioural specification on $\overline{TF}$ is automatically satisfiable. This allows us to discharge automatically the side conditions for application of the recursion induction theorem, and leaves us only with the principle proof obligation that $S(s,\aleph)$ is preserved by recursive calls *on arbitrary sets of failures* — we cannot assume in any proof that the argument to the function F will be a process. The soundness of the other inference rules does not rest on the assumption that the arguments to the operators will be processes, and so those rules are also sound for the clauses of $\overline{\mathcal{F}}_T$. Hence the inference rules may be used to establish that our candidate specification S is preserved by recursive calls

4.4 Completeness

We have shown that the proof system is *sound*, in the sense that if we can find behavioural specifications for which all the premisses of an inference rule are simultaneously **true**, tben the conclusion will also be true. We will shortly show that the proof system is *complete* with respect to the semantics, in that if the conclusion of an inference rule is true of the semantics of a process, then there are behavioural specifications which enable all the premisses of that rule to hold simultaneously. Moreover, we provide a systematic method of producing such behavioural specifications.

Strongest Specification

The behavioural specifications satisfied by a given process P form a complete lattice under the order

$$S(s,\aleph) \sqsubseteq T(s,\aleph) \ \ \widehat{=}\ \ \forall (s,\aleph) \bullet T(s,\aleph) \Rightarrow S(s,\aleph)$$

We may therefore identify the strongest specification that holds of a given process; it will be logically equivalent to the top element of the lattice. We denote the *strongest specification* of process P by $\mathcal{SS}[\![P]\!]$. It is clear that

$$\mathcal{SS}[\![P]\!](s,\aleph) \ \Leftrightarrow\ (s,\aleph) \in \mathcal{F}_T[\![P]\!]$$

Consider first the case of a one-place process constructor. If it is the case that $\odot_I\ P$ **sat** $S(b)$, then consider the inference rule for $\odot_I\ P$ with $\mathcal{SS}[\![P]\!]$ substituted for S_I. The rule becomes

$$\begin{array}{l} P\ \mathbf{sat}\ \mathcal{SS}[\![P]\!] \\ T'(b) \Rightarrow S(b) \\ (\mathcal{SS}[\![P]\!](f(b)) \wedge T(b)) \Rightarrow S(C(b)) \\ \hline \odot_I\ P\ \mathbf{sat}\ S(b) \end{array}$$

Tbe first premiss must hold, and the second and third premisses together are equivalent to the assertion

$$b \in \mathcal{F}_T[\![\odot_I P]\!] \ \Rightarrow\ S(b)$$

But if $\odot_I\ P$ **sat** $S(b)$, then the second and third premisses of the rule hold. Hence if it is the case that $\odot_I\ P$ **sat** $S(b)$ then we are able to find a behavioural specification on P which enables an application of the law.

The same treatment applies to all the other cases (except recursion), since in each case the inference rule is derived from the corresponding clause for the definition of $\mathcal{F}_T$. Moreover, we may use the inference rules to break down the strongest specification of a compound process into the strongest specification of its component processes. The rules for the basic processes $STOP$, $\perp$, $SKIP$ and $WAIT\ t$ yield the strongest specifications as follows:

$$\begin{aligned}
\mathcal{SS}[\![STOP]\!](s,\aleph) &\Leftrightarrow s = \langle\rangle \\
\mathcal{SS}[\![\perp]\!](s,\aleph) &\Leftrightarrow s = \langle\rangle \\
\mathcal{SS}[\![SKIP]\!](s,\aleph) &\Leftrightarrow (s = \langle\rangle \wedge \checkmark \notin \sigma(\aleph)) \\
&\qquad \vee \\
&\qquad (s = \langle(t,\checkmark)\rangle \wedge \checkmark \notin \sigma(\aleph \upharpoonright t) \wedge t \geqslant 0) \\
\mathcal{SS}[\![WAIT\ t]\!](s,\aleph) &\Leftrightarrow s = \langle\rangle \wedge \checkmark \notin \sigma(\aleph \upharpoonleft t) \\
&\qquad \vee \\
&\qquad s = \langle(t',\checkmark)\rangle \wedge t' \geqslant t \wedge \checkmark \notin \sigma(\aleph \uparrow [t,t'))
\end{aligned}$$

As we would expect, the strongest specifications for compound processes are given in terms of the strongest specifications of their component processes:

$$\begin{aligned}
\mathcal{SS}[\![a \rightarrow P]\!](s,\aleph) &\Leftrightarrow s = \langle\rangle \wedge a \notin \sigma(\aleph) \\
&\qquad \vee \\
&\qquad s = \langle(t,a)\rangle^\frown s' \wedge a \notin \sigma(\aleph \upharpoonright t) \wedge begin(s') \geqslant t + \delta \\
&\qquad \wedge\, \mathcal{SS}[\![P]\!](s \dot{-} (t+\delta), (\aleph \dot{-} (t+\delta))) \\
\mathcal{SS}[\![P_1 \,\Box\, P_2]\!](s,\aleph) &\Leftrightarrow (\mathcal{SS}[\![P_1]\!](s,\aleph) \vee \mathcal{SS}[\![P_2]\!](s,\aleph) \\
&\qquad \wedge \\
&\qquad \mathcal{SS}[\![P_1]\!](\langle\rangle, \aleph \upharpoonright begin(s)) \wedge \mathcal{SS}[\![P_2]\!](\langle\rangle, \aleph \upharpoonright begin(s)) \\
\mathcal{SS}[\![a : A \rightarrow P_a]\!](s,\aleph) &\Leftrightarrow s = \langle\rangle \wedge \sigma(\aleph) \cap A = \emptyset \\
&\qquad \vee \\
&\qquad \exists\, a \in A \bullet s = \langle(t,a)\rangle^\frown s') \wedge begin(s') \geqslant t + \delta \wedge \\
&\qquad\qquad \mathcal{SS}[\![P_a]\!](s' \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))
\end{aligned}$$

$$\mathcal{SS}\llbracket P_1 \sqcap P_2 \rrbracket(s,\aleph) \Leftrightarrow \mathcal{SS}\llbracket P_1 \rrbracket(s,\aleph) \vee \mathcal{SS}\llbracket P_2 \rrbracket(s,\aleph)$$

$$\mathcal{SS}\llbracket \bigsqcap_{i\in I} P_i \rrbracket(s,\aleph) \Leftrightarrow \exists\, i \in I \bullet \mathcal{SS}\llbracket P_i \rrbracket(s,\aleph)$$

$$\begin{aligned}\mathcal{SS}\llbracket P_1 \parallel P_2 \rrbracket(s,\aleph) \Leftrightarrow\ & \exists\,\aleph_1,\aleph_2 \bullet \mathcal{SS}\llbracket P_1 \rrbracket(s,\aleph_1) \wedge \mathcal{SS}\llbracket P_2 \rrbracket(s,\aleph_2)\\ & \qquad \wedge \aleph = \aleph_1 \cup \aleph_2\end{aligned}$$

$$\begin{aligned}\mathcal{SS}\llbracket P_1 \;{}_X\!\parallel_Y P_2 \rrbracket(s,\aleph) \Leftrightarrow\ & \exists\,\aleph_1,\aleph_2 \bullet \mathcal{SS}\llbracket P_1 \rrbracket(s \upharpoonright X, \aleph_1 \upharpoonright X)\\ & \wedge \mathcal{SS}\llbracket P_2 \rrbracket(s \upharpoonright Y, \aleph_2 \upharpoonright Y)\\ & \wedge \aleph \upharpoonright (X \cup Y) = \aleph_1 \upharpoonright X \cup \aleph_2 \upharpoonright Y \wedge s = s \upharpoonright (X \cup Y)\end{aligned}$$

$$\begin{aligned}\mathcal{SS}\llbracket P_1 \mid\mid\mid P_2 \rrbracket(s,\aleph) \Leftrightarrow\ & \exists\, s_1, s_2 \bullet (s \in \mathit{Tmerge}(s_1,s_2) \wedge\\ & \mathcal{SS}\llbracket P_1 \rrbracket(s_1,\aleph) \wedge \mathcal{SS}\llbracket P_2 \rrbracket(s_2,\aleph))\end{aligned}$$

$$\begin{aligned}\mathcal{SS}\llbracket P_1 \,;\, P_2 \rrbracket(s,\aleph) \Leftrightarrow\ & \checkmark \notin \sigma(s) \wedge \forall\, I \in \mathit{TINT} \bullet \mathcal{SS}\llbracket P_1 \rrbracket(s, \aleph \cup (I \times \{\checkmark\}))\\ & \vee\\ & \mathcal{SS}\llbracket P_1 \rrbracket(s \upharpoonright t ^\frown \langle (t,\checkmark) \rangle, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \wedge\\ & s \uparrow (t, t+\delta) = \langle\rangle \wedge \mathcal{SS}\llbracket P_2 \rrbracket(s \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))\end{aligned}$$

$$\mathcal{SS}\llbracket \mathit{WAIT}\ t \,;\, P \rrbracket(s,\aleph) \Leftrightarrow \mathcal{SS}\llbracket P \rrbracket(s \dot{-} t, \aleph \dot{-} t) \wedge \mathit{begin}(s) \geqslant t$$

$$\mathcal{SS}\llbracket P \setminus A \rrbracket(s,\aleph) \Leftrightarrow \exists\, s_1 \bullet \mathcal{SS}\llbracket P \rrbracket(s_1, \aleph \cup [0, \mathit{end}(s_1)) \times A) \wedge s_1 \setminus A = s$$

$$\mathcal{SS}\llbracket f^{-1}(P) \rrbracket(s,\aleph) \Leftrightarrow \mathcal{SS}\llbracket P \rrbracket(f(s), f(\aleph))$$

$$\mathcal{SS}\llbracket f(P) \rrbracket(s,\aleph) \Leftrightarrow \mathcal{SS}\llbracket P \rrbracket(s, f^{-1}(\aleph))$$

$$\begin{aligned}\mathcal{SS}\llbracket P_1 \,;\, P_2 \rrbracket(s,\aleph) \Leftrightarrow\ & \checkmark \notin \sigma(s) \wedge \forall\, I \in \mathit{TINT} \bullet \mathcal{SS}\llbracket P_1 \rrbracket(s, \aleph \cup (I \times \{\checkmark\}))\\ & \vee\\ & \exists\, s_1, s_2 \bullet s \cong s_1 {}^\frown (s_2 + t) \wedge \checkmark \notin \sigma(s_1) \wedge \mathit{end}(\aleph_1) \leqslant t\\ & \wedge \mathcal{SS}\llbracket P_1 \rrbracket(s_1 {}^\frown \langle (t,\checkmark) \rangle, \aleph_1 \cup ([0,t) \times \{\checkmark\}))\\ & \wedge \mathcal{SS}\llbracket P_2 \rrbracket(s_2, \aleph_2)\end{aligned}$$

$$
\begin{aligned}
\mathcal{SS}[\![P_1 \overset{t}{\rhd} P_2]\!](s,\aleph) \;\Leftrightarrow\;& begin(s) \leqslant t \wedge \mathcal{SS}[\![P_1]\!](s,\aleph)\\
&\vee\\
&begin(s) \geqslant t + \delta \wedge \mathcal{SS}[\![P_1]\!](\langle\rangle, \aleph \upharpoonright t)\\
&\wedge \mathcal{SS}[\![P_2]\!](s \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{SS}[\![P_1 \underset{t}{\lightning} P_2]\!](s,\aleph) \;\Leftrightarrow\;& \mathcal{SS}[\![P_1]\!](s \upharpoonright t, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \wedge \checkmark \notin \sigma(s \upharpoonright t) \wedge\\
&s \uparrow (t, t+2\delta) = \langle\rangle \wedge \mathcal{SS}[\![P_2]\!](s \dot{-} (t+2\delta), \aleph \dot{-} (t+2\delta))\\
&\vee\\
&\mathcal{SS}[\![P_1]\!](s ^\frown \langle(t',\checkmark)\rangle, \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\} \wedge t' \leqslant t \wedge\\
&\checkmark \notin \sigma(s) \wedge \checkmark \notin \sigma(\aleph \uparrow t')\}\\
&\vee\\
&s = s' {}^\frown \langle(t'',\checkmark)\rangle \wedge \exists\, t' \bullet \checkmark \notin \sigma(s) \wedge\\
&\mathcal{SS}[\![P_1]\!](s' {}^\frown \langle(t',\checkmark)\rangle), \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\})\\
&\wedge \checkmark \notin \sigma(\aleph \uparrow [t',t'')) \wedge t' \leqslant t \wedge t' \leqslant t''
\end{aligned}
$$

Recursion

Recall the inference rule for recursion:

$$
\frac{\begin{array}{l}\forall X \bullet X \textbf{ sat } S_I(s,\aleph) \Rightarrow F(\mathit{WAIT}\,\delta\,;X) \textbf{ sat } S_I(s,\aleph)\\ S_I(s,\aleph) \Rightarrow S(s,\aleph)\end{array}}{\mu X \bullet F(X) \textbf{ sat } S(s,\aleph)}
$$

If $S_I(s,\aleph) \Leftrightarrow \mathcal{SS}[\![\mu X \bullet F(X)]\!](s,\aleph)$, then recalling that $S_I(s,\aleph) \Leftrightarrow (s,\aleph) \in \mathcal{F}_T[\![\mu X \bullet F(X)]\!]$, we observe that the first premiss holds:

$$
\begin{aligned}
& Y \textbf{ sat } S_I(s,\aleph)\\
\Rightarrow\;& \mathcal{F}_T[\![Y]\!] \subseteq \mathcal{F}_T[\![\mu X \bullet F(X)]\!]\\
\Rightarrow\;& \mathcal{F}_T[\![F(\mathit{WAIT}\,\delta\,;Y)]\!] \subseteq \mathcal{F}_T[\![F(\mathit{WAIT}\,\delta\,;\mu X \bullet F(X))]\!] \quad \text{(monotonicity)}\\
\Rightarrow\;& \mathcal{F}_T[\![F(\mathit{WAIT}\,\delta\,;X)]\!] \subseteq \mathcal{F}_T[\![\mu X \bullet F(X)]\!]\\
\Rightarrow\;& F(\mathit{WAIT}\,\delta\,;X) \textbf{ sat } S_I(s,\aleph)
\end{aligned}
$$

If $\mu X \bullet F(X)$ **sat** $S(s,\aleph)$ then $S_I(s,\aleph) \Rightarrow S(s,\aleph)$, and so for any behavioural specification satisfied by $\mu X \bullet F(X)$ there is a predicate S_I which enables the application of the law. Hence the entire proof system is complete with respect to the semantics of $\mathcal{F}_T$.

In order to prove that a recursively defined process satisfies a behavioural specification S, we need only show that the strongest specification of that recursive process implies S. We therefore need a form of the strongest specification that will facilitate this.

We know that $\mathcal{F}_T[\![\mu X \bullet F(X)]\!]$ is the fixed point of the constructive function corresponding to $\hat{F}(X) = F(\mathit{WAIT}\, \delta \,; X)$, and that $\hat{F}$ is δ-constructive, in that

$$X \restriction t = Y \restriction t \Rightarrow \hat{F}(X) \restriction (t+\delta) = \hat{F}(Y) \restriction (t+\delta)$$

Since we also have that

$$\forall X, Y \in \overline{TF} \bullet X \restriction 0 = Y \restriction 0$$

we are able to conclude that

$$\forall X, Y \in \overline{TF} \bullet \hat{F}^n(X) \restriction n\delta = \hat{F}^n(Y) \restriction n\delta$$

In particular, instantiating Y with $\mu X \bullet F(X)$ we conclude that

$$\forall X \in \overline{TF} \bullet \hat{F}^n(X) \restriction n\delta = \mu X \bullet F(X) \restriction n\delta$$

and so, if we instantiate X with STOP (for definiteness), we obtain

$$\mathit{end}(s,\aleph) < n\delta \Rightarrow ((s,\aleph) \in \mathcal{F}_T[\![\hat{F}^n(\mathit{STOP})]\!] \Leftrightarrow (s,\aleph) \in \mathcal{F}_T[\![\mu X \bullet F(X)]\!])$$

This yields a useful form of the strongest specification for a recursively defined process in terms of the strongest specification of STOP and the operators used in constructing the recursive function.

$$\mathcal{SS}[\![\mu X \bullet F(X)]\!](s,\aleph) \;\Leftrightarrow\; \forall n \bullet \mathit{end}(s,\aleph) < n\delta \Rightarrow \mathcal{SS}[\![\hat{F}^n(\mathit{STOP})]\!](s,\aleph)$$

Since $\mathcal{SS}[\![\hat{F}^n(\mathit{STOP})]\!](s,\aleph)$ holds either for all $n > \mathit{end}(s,\aleph)/\delta$ or for no such n, the strongest specification may alternatively be written:

$$\mathcal{SS}[\![\mu X \bullet F(X)]\!](s,\aleph) \;\Leftrightarrow\; \mathcal{SS}[\![\hat{F}^{\lfloor \mathit{end}(s,\aleph)/\delta \rfloor + 1}(\mathit{STOP})]\!](s,\aleph)$$

($\lfloor r \rfloor$ denotes the greatest integer no larger than r)

Limits of Applicability

We have supplied a great deal of machinery for proving correctness results concerning behavioural specifications. However, as we saw earlier not every specification on processes can be written as a behavioural specification. Demonstrating that a particular specification *can* be written as a behavioural specification is straightforward: we need only exhibit the specification in the right form.

We require conditions on specifications that tell us when they *cannot* be written as hehavioural specifications, that a search for an eqnivalent specification of that form would not succeed. We provide some snfficient conditions on specifications that guarantee that they cannot be written as behavioural specifications.

Theorem 4.4.1 *If a specification on TM_F is not closed in TM_F then it cannot be written as a behavioural specification.*

This is equivalent to our earlier result that all behavioural specifications on TM_F are closed. This result will also be valid for the instabilities model TM_{FI} introduced in the next section, but it does not hold for TM_{FS}. For example, the behavioural specification defined by $S(s, \alpha, \aleph) \mathrel{\widehat{=}} \alpha < \infty$ is not closed.

Theorem 4.4.2 *If $\exists\, P, Q \in TM_F \bullet S(P \sqcap Q) \wedge \neg S(P)$ then S cannot be written as a behavioural specification.*

Proof Assume for a contradiction that there is a behavioural specification $T(s, \aleph)$ such that

$$S(P) \;\Leftrightarrow\; P \textbf{ sat } T(s, \aleph)$$

Then we have $\mathcal{F}_T[\![P]\!] \subseteq \mathcal{F}_T[\![P \sqcap Q]\!]$, and so

$$P \sqcap Q \textbf{ sat } T(s, \aleph) \;\Rightarrow\; P \textbf{ sat } T(s, \aleph)$$

yielding a contradiction. □

Theorem 4.4.3 *If $S(P)$, $S(Q)$, but $\neg S(P \sqcap Q)$, then S cannot be written as a behavioural specification.*

4.5 A treatment of stability

The completeness of the proof system described above rests upon the following facts:

- given a combination of possible behaviours from subprocesses, we can determine whether or not they give rise to a behaviour of the process without examining the entire semantic set of each subprocess;
- if a given combination of possible behaviours of subprocesses does give rise to a behaviour of the process, then that behaviour is completely determined by the given behaviours.

The semantics is in some sense *directly* compositional. This is reflected in the logical premises of the proof rules: the antecedents hold precisely when the failures under consideration give rise to a failure in the composite process.

When we move to models which contain stability information, this is no longer the case. For example, consider the semantic equation to TM_{FS} for the interleaving operator:

$$\begin{array}{rcl} \mathcal{E}_T[\![P_1 \mid\mid\mid P_2]\!] & = & SUP(I(P_1, P_2)) \\ I(P_1, P_2) & \triangleq & \{(s, \alpha, \aleph) \mid s \in Tmerge(s_1, s_2), \alpha = max\{\alpha_1, \alpha_2\} \\ & & \quad \wedge (s_1, \alpha_1, \aleph) \in \mathcal{E}_T[\![P_1]\!] \\ & & \quad \wedge (s_2, \alpha_2, \aleph) \in \mathcal{E}_T[\![P_2]\!] \end{array}$$

If we define

$$\begin{array}{rcl} P_1 & \triangleq & a \rightarrow a \rightarrow STOP \\ P_2 & \triangleq & a \rightarrow WAIT\ 1\ ; STOP \end{array}$$

then we have

$$\begin{array}{rcl} (\langle(1, a)\rangle, 1 + \delta, \emptyset) & \in & \mathcal{E}_T[\![P_1]\!] \\ (\langle(0, a)\rangle, 1 + \delta, \emptyset) & \in & \mathcal{E}_T[\![P_2]\!] \end{array}$$

and so

$$(\langle(0, a), (1, a)\rangle, 1 + \delta, \emptyset) \ \in I(P_1, P_2)$$

but it does not contribute to a behaviour of $P_1 \mid\mid\mid P_2$ because its stability value is superseded by that of

$$(\langle(0, a), (1, a)\rangle, 2 + \delta, \emptyset) \ \in I(P_1, P_2)$$

Consideration of the behaviour $(\langle(0, a), (1, a)\rangle, 1 + \delta, \emptyset)$ in isolation will not enable us to determine whether or not it contributes to a behaviour of $P_1 \mid\mid\mid P_2$. In calculating the stability value associated with the failure $(\langle(0, a), (1, a)\rangle, \emptyset)$ the entire set $I(P_1, P_2)$ must be considered. This means tbat a behavioural proof rule for iuterleaviug will not be complete. The rule is as follows:

$$\begin{array}{l}
P_1 \ \mathbf{sat} \ S_1(s, \alpha, \aleph) \\
P_2 \ \mathbf{sat} \ S_2(s, \alpha, \aleph) \\
\left.\begin{array}{l} s \in Tmerge(s_1, s_2) \wedge \\ S_1(s_1, \alpha_1, \aleph) \wedge S_2(s_2, \alpha_2, \aleph) \end{array}\right\} \Rightarrow S(s, max\{\alpha_1, \alpha_2\}, \aleph) \\
\hline
P_1 \ ||| \ P_2 \ \mathbf{sat} \ S(s, \alpha, \aleph)
\end{array}$$

The rule is sound: any bebaviour of $P_1 \ ||| \ P_2$ will be the interleaving of two behaviours from the components, with a stability value corresponding to the maximum of the two component stability values. However, the rule is not complete. Consider the application of the rule to the two processes P_1 and P_2 given earlier, to the proof obligation S, where

$$S(s, \alpha, \aleph) \quad \Leftrightarrow \quad s = \langle (0, a), (1, a) \rangle \Rightarrow \alpha = 2 + \delta$$

a behavioural specification which holds of $P_1 \ ||| \ P_2$. If we are able to find behavioural specifications S_1 and S_2 which hold of P_1 and P_2 respectively, then we must have $S_1(\langle (1, a) \rangle, 1 + \delta, \emptyset)$ and $S_2(\langle (0, a) \rangle, 1 + \delta, \emptyset)$, and so $S(\langle (0, a), (1, a) \rangle, 1 + \delta, \emptyset)$. But this is not the case. Hence, the rule cannot be used in establishing this proof obligation.

The SUP operator presents a further obstacle: consider the set of processes $\{P_n\}$ defined by $P_n = n \rightarrow WAIT\,(1 - \frac{1}{n})$, then the process $P = (n : \mathbb{N} \rightarrow P_n) \backslash \mathbb{N}$ has the behaviour $(\langle\rangle, 1, \emptyset)$, even though there is no behaviour of $n : \mathbb{N} \rightarrow P_n$ from which it can have come. The stability value 1 really does arise from the entire set of stability values of the behaviours that have given rise to the failure $(\langle\rangle, \emptyset)$; no single component bebaviour can give rise to that stability value. Hence it is not possible to formulate a rule for behavioural specifications of the hiding operator.

These problems can be overcome by altering our treatment of stability, employing the notion of 'instability' values introduced by Blamey in [Bla89]. In this approach, each failure $(s, \aleph)$ is associated with a *set* of instability values, each corresponding to a time at which the process might still be unstable, having performed s and refused $\aleph$ up to that time. The iustability value $end(s)$ is always included as a trivial instabihty. Hence, if process P in TM_{FS} has a behaviour $(s, \alpha, \aleph)$, then the semantics of the process described in terms of instability values will contain all elements of the set

$$\{(s, \gamma, \aleph) \mid end(s) \leqslant \gamma < \alpha \vee end(s) = \gamma = \alpha\}$$

We will denote by TM_{FI} the model which describes processes in terms of triples $(s, \gamma, \aleph)$ which represent (*trace,instability,refusal*) information. Blamey demonstrates that there is a natural isomorphism between TM_{FS} and TM_{FI}. If S is an element of TM_{FS}, then the corresponding element of TM_{FI} is given by

$$\hat{S} \;=\; \{(s,\gamma,\aleph) \mid \exists\,\alpha \bullet (s,\alpha,\aleph) \in S \wedge (end(s) \leqslant \gamma < \alpha \vee end(s) = \gamma)\}$$

Conversely, if P is an element of TM_{FI}, then the corresponding element of TM_{FS} is given by

$$\check{P} \;=\; \{(s,\alpha,\aleph) \mid \exists\,\gamma \bullet (s,\gamma,\aleph) \in P \wedge \alpha = sup\{\gamma \mid (s,\gamma,\aleph) \in P\}\}$$

The semantic equation $\mathcal{E}_{TI} : TCSP \rightarrow TM_{FI}$ enjoys the property that allowed the construction of a complete proof system for TM_F: that it follows purely from the nature of the component behaviours themselves whether or not they give rise to a composite behaviour in the compound process. We are thus able to formulate a complete (and sound) inference rule for each syntactic operator, so we arrive at a proof system for behavioural specifications on TM_{FI}. Blamey also shows that behavioural specifications are closed in TM_{FI} (on which the obvious metric is defined), so the side condition for the recursion induction principle requiring closure is automatically discharged.

The information represented by $(s,\gamma,\aleph) \in \mathcal{E}_{TI}[\![P]\!]$ is that the process may perform s and refuse $\aleph$, and that internal activity may still be occurring after the performance of s and refusal of $\aleph \upharpoonright \gamma$. The expressive power of behavioural specifications on TM_{FI} is therefore different from those on TM_{FS}, since the triples in each case represent different aspects of stability behaviour. Indeed, any behavioural specification on TM_{FI} corresponds to a behavioural specification on TM_{FS}: we have P sat $S(s,\gamma,\aleph)$ if and only if $\check{P}$ sat $T_S(s,\alpha,\aleph)$, where T_S is defined by

$$T_S(s,\alpha,\aleph) \;\widehat{=}\; S(s,end(s),\aleph) \wedge \forall\,\gamma \in [end(s),\alpha) \bullet S(s,\gamma,\aleph)$$

The converse is not true. For example, there is no behavioural specification on TM_{FI} that corresponds to the behavioural specification on TM_{FS}:

$$S(s,\alpha,\aleph) \;\widehat{=}\; \alpha = end(s) + 3$$

since this can only be deduced from the whole set of instabilities. As was the case for TM_F, we can only apply the proof system to a particular kind of specification on processes: in this case, we can only treat a certain kind of behavioural specification on TM_{FS} — those that can be written as behavioural specifications on TM_{FI}.

4.6 The proof system for TM_{FI}

Rule *STOP*

$$\frac{(s = \langle\rangle \wedge \gamma = 0) \Rightarrow S(s,\gamma,\aleph)}{STOP \ \textbf{sat} \ S(s,\gamma,\aleph)}$$

Rule $\perp$

$$\frac{(s = \langle\rangle) \Rightarrow S(s,\gamma,\aleph)}{\perp \ \textbf{sat} \ S(s,\gamma,\aleph)}$$

Rule *SKIP*

$$\frac{\begin{array}{l}(s = \langle\rangle \wedge \gamma = 0 \wedge \checkmark \notin \sigma(\aleph)) \Rightarrow S(s,\gamma,\aleph)\\ (s = \langle(\gamma,\checkmark)\rangle \wedge \checkmark \notin \sigma(\aleph \upharpoonright \gamma)) \Rightarrow S(s,\gamma,\aleph)\end{array}}{SKIP \ \textbf{sat} \ S(s,\gamma,\aleph)}$$

Rule *WAIT t*

$$\frac{\begin{array}{l}(s = \langle\rangle \wedge (\gamma = 0 \vee \gamma < t) \wedge \checkmark \notin \sigma(\aleph \upharpoonleft t) \Rightarrow S(s,\gamma,\aleph)\\ s = \langle(\gamma,\checkmark)\rangle \wedge \gamma \geqslant t \wedge \checkmark \notin \sigma(\aleph \uparrow [t,\gamma)) \Rightarrow S(s,\gamma,\aleph)\end{array}}{WAIT\ t \ \textbf{sat} \ S(s,\gamma,\aleph)}$$

The following rules apply to compound processes. When a process variable is present. it is more convenient to match proof obligations to consequents: the form in which the rules are presented makes this possible.

Rule $a \to P$

$$\frac{\begin{array}{l}P \ \textbf{sat} \ T(s,\gamma,\aleph)\\ s = \langle\rangle \wedge \gamma = 0 \wedge a \notin \sigma(\aleph) \Rightarrow S(s,\gamma,\aleph)\\ \left.\begin{array}{l}s = \langle(t,a)\rangle^\frown(s \upharpoonleft (t+\delta)) \wedge \gamma \geqslant end(s) \wedge a \notin \sigma(\aleph \upharpoonright t) \wedge\\ T(s' \mathbin{\dot{-}} (t+\delta), \gamma \mathbin{\dot{-}} (t+\delta), \aleph \mathbin{\dot{-}} (t+\delta))\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\end{array}}{(a \to P) \ \textbf{sat} \ S(s,\gamma,\aleph)}$$

Rule $a : A \rightarrow P_a$

$$\begin{array}{l}
\forall a \in A \bullet P_a \textbf{ sat } S_a(s, \gamma, \aleph) \\
(s = \langle\rangle \wedge \gamma = 0 \wedge A \cap \sigma(\aleph) = \emptyset) \Rightarrow S(s, \gamma, \aleph) \\
\left.\begin{array}{l}
(s = \langle(t\,, a)\rangle^\frown(s \uparrow (t + \delta))) \wedge \gamma \geqslant end(s) \wedge A \cap \alpha(\aleph \upharpoonright t) \wedge \\
S_a(s \mathbin{\dot{-}} (t + \delta), \gamma \mathbin{\dot{-}} (t + \delta), (\aleph \mathbin{\dot{-}} (t + \delta)))
\end{array}\right\} \Rightarrow S(s, \gamma, \aleph) \\
\hline
a : A \rightarrow P_a \textbf{ sat } S(s, \gamma, \aleph)
\end{array}$$

Rule $P_1 \sqcap P_2$

$$\begin{array}{l}
P_1 \textbf{ sat } S_1(s, \gamma, \aleph) \\
P_2 \textbf{ sat } S_2(s, \gamma, \aleph) \\
(S_1(s, \gamma, \aleph) \vee S_2(s, \gamma, \aleph)) \Rightarrow S(s, \gamma, \aleph) \\
\hline
P_1 \sqcap P_2 \textbf{ sat } S(s, \gamma, \aleph)
\end{array}$$

Rule $\bigsqcap_i P_i$

$$\begin{array}{l}
\forall i \in I \bullet P_i \textbf{ sat } S_i(s, \gamma, \aleph) \\
\forall i \in I \bullet (S_i(s, \gamma, \aleph) \Rightarrow S(s, \gamma, \aleph)) \\
\hline
\bigsqcap_{i \in I} P_i \textbf{ sat } S(s, \gamma, \aleph)
\end{array}$$

Rule $P_1 \Box P_2$

$$\begin{array}{l}
P_1 \textbf{ sat } S_1(s, \gamma, \aleph) \\
P_2 \textbf{ sat } S_2(s, \gamma, \aleph) \\
\left.\begin{array}{l}
(S_1(s, \gamma, \aleph) \vee S_2(s, \gamma, \aleph) \wedge \\
S_1(\langle\rangle, 0\,, \aleph \upharpoonright begin(s)) \wedge S_2(\langle\rangle, 0, \aleph \upharpoonright begin(s))
\end{array}\right\} \Rightarrow S(s, \gamma, \aleph) \\
\hline
P_1 \Box P_2 \textbf{ sat } S(s, \gamma, \aleph)
\end{array}$$

Rule $P_1 \parallel P_2$

$$\begin{array}{l}
P_1 \textbf{ sat } S_1(s, \gamma, \aleph) \\
P_2 \textbf{ sat } S_2(s, \gamma, \aleph) \\
\left.\begin{array}{l}
S_1(s, end(s), \aleph_1) \wedge S_2(s, end(s), \aleph_2 \wedge \\
(S_1(s, \gamma, \aleph_1) \vee S_2(s, \gamma, \aleph_2)
\end{array}\right\} \Rightarrow S(s, \gamma, \aleph_1 \cup \aleph_2) \\
\hline
P_1 \parallel P_2 \textbf{ sat } S(s, \gamma, \aleph)
\end{array}$$

Rule $P_1 \;{}_X\|_Y\; P_2$

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2 \ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
\left.\begin{array}{l}
\sigma(s) \subseteq X \cup Y \wedge \aleph \restriction X \cup Y = \aleph_1 \restriction X \cup \aleph_2 \restriction Y \ \wedge\\
S_1(s \restriction X, end(s \restriction X), \aleph_1) \wedge S_2(s \restriction Y, end(s \restriction Y), \aleph_2) \ \wedge\\
(S_1(s \restriction X, \gamma, \aleph_1) \vee S_2(s \restriction Y, \gamma, \aleph_2))
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
P_1 \;{}_X\|_Y\; P_2 \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P_1 \mathbin{|||} P_2$

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2 \ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
\left.\begin{array}{l}
(s \in \mathit{Tmerge}(s_1, s_2) \wedge S_1(s_1, end(s_1), \aleph) \wedge S_2(s_2, end(s_2), \aleph)) \ \wedge\\
(S_1(s_1, \gamma, \aleph) \vee S_2(s_2, \gamma, \aleph))
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
P_1 \mathbin{|||} P_2 \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $\mathit{WAIT}\ t\ ; P$

$$\begin{array}{l}
P \ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
(S_1(s \dot{-} t, \gamma \dot{-} t, \aleph \dot{-} t) \wedge begin(s) \geqslant t \wedge \gamma \geqslant t) \Rightarrow S(s,\gamma,\aleph)\\
\hline
\mathit{WAIT}\ t\ ; P \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P_1 \, ; P_2$

$$\begin{array}{l}
P_1 \ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2 \ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
\checkmark \notin \sigma(s) \wedge \forall \beta \bullet S_1(s, \gamma, \aleph \cup [0,\beta) \times \{\checkmark\}) \Rightarrow S(s,\gamma,\aleph)\\
\left.\begin{array}{l}
\exists t \bullet \checkmark \notin \sigma(s \restriction t) \wedge S_1(s \restriction t ^\frown \langle (t,\checkmark) \rangle, t, \aleph \cup [0,t) \times \{\checkmark\})\\
\wedge S_2(s \dot{-} (t+\delta), \gamma \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))\\
\wedge\, s \uparrow (t, t+\delta) = \langle\rangle \wedge \gamma \geqslant end(s)
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
P_1 \, ; P_2 \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P \setminus A$

$$\begin{array}{l}
P \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \\
\left.\begin{array}{l}
end(s) \leqslant \gamma \wedge \exists\, s_1 \bullet s_1 \setminus A \wedge \\
((\exists\, \beta > \gamma \bullet S_1(s_1, \beta, \aleph \cup ([0, max\{\beta, end(\aleph)\}) \times A)) \vee \\
(end(s) = \alpha \wedge S_1(s_1, \gamma, \aleph \cup ([0, max\{\alpha, end(\aleph)\} \times A)))
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph) \\
\hline
P \setminus A \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

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

$$\begin{array}{l}
P \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \\
S_1(f(s),\gamma,f(\aleph)) \Rightarrow S(s,\gamma,\aleph) \\
\hline
f^{-1}(P) \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $f(P)$

$$\begin{array}{l}
P \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \\
S_1(s,\gamma,f^{-1}(\aleph)) \Rightarrow S(f(s),\gamma,\aleph) \\
\hline
f(P) \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $\mu\, X \bullet F(X)$

$$\begin{array}{l}
\forall\, X \bullet X \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \Rightarrow F(WAIT\,\delta\,; X) \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \\
S_1(s,\gamma,\aleph) \Rightarrow S(s,\gamma,\aleph) \\
\hline
\mu\, X \bullet F(X) \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Observe that, similar to the corresponding rule in the proof system for TM_F, the process variable X in the first premiss ranges over all sets of instability values, not only those in TM_{FI}. Again, we provide an alternative rule:

Rule $\mu\, X \bullet F(X)$ alternative version

$$\begin{array}{l}
\forall\, X : TM_{FI} \bullet (X \ \mathbf{sat}\ S_1(s,\gamma,\aleph) \Rightarrow \\
\qquad\qquad\qquad F(WAIT\,\delta\,; X) \ \mathbf{sat}\ S_1(s,\gamma,\aleph)) \\
S_1(s,\gamma,\aleph) \Rightarrow S(s,\gamma,\aleph) \\
\hline
\mu\, X \bullet F(X) \ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array} \quad [\, \exists\, P : TM_{FI} \bullet P \ \mathbf{sat}\ S_1(s,\gamma,\aleph)\,]$$

We also provide some additional rules:

Rule $P_1 ; P_2$

$$\begin{array}{l}
P_1\ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2\ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
(\checkmark \notin \sigma(s) \wedge \forall \beta \bullet S_1(s,\gamma,\aleph \cup ([0,\beta) \times \{\checkmark\})) \Rightarrow S(s,\gamma,\aleph)\\
\left.\begin{array}{l}
s \cong s_1 \frown \langle s_2 \rangle \wedge \checkmark \notin \sigma(s_1) \wedge end(s_1) \leqslant \beta \leqslant begin(s_2) \wedge\\
S_1(s_1 \frown \langle(\beta,\checkmark)\rangle, \beta, \aleph \upharpoonright \beta \cup ([0,t) \times \{\checkmark\})) \wedge\\
\gamma \geqslant end(s) \wedge S_2(s_2 \dot{-} \beta, \gamma \dot{-} \beta, \aleph_2)
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
(P_1 ; P_2)\ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P_1 \overset{t}{\triangleright} P_2$

$$\begin{array}{l}
P_1\ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2\ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
(S_1(s,\gamma,\aleph) \wedge begin(s) \leqslant t \Rightarrow S(s,\gamma,\aleph)\\
\left.\begin{array}{l}
begin(s) \geqslant t + \delta \wedge S_1(\langle\rangle, 0, \aleph \upharpoonright t) \wedge\\
S_2(s \dot{-} (t+\delta), \gamma \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
P_1 \overset{t}{\triangleright} P_2\ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P_1 \underset{t}{\lightning} P_2$

$$\begin{array}{l}
P_1\ \mathbf{sat}\ S_1(s,\gamma,\aleph)\\
P_2\ \mathbf{sat}\ S_2(s,\gamma,\aleph)\\
\left.\begin{array}{l}
(S_1(s \upharpoonright t, end(s \upharpoonright t), \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\})\\
\wedge\ S_2(s \dot{-} (t+\delta), \gamma \dot{-} (t+\delta), \aleph \dot{-} (t+\delta))\\
\wedge\ s \uparrow (t, t+2\delta) = \langle\rangle \wedge \gamma \geqslant end(s)
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\left.\begin{array}{l}
\exists t' \bullet S_1(s \frown \langle(t',\checkmark)\rangle, t', \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\})\\
\wedge\ \checkmark \notin \sigma(s) \wedge \checkmark \notin \sigma(\aleph \uparrow t') \wedge \gamma = t'
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\left.\begin{array}{l}
s = s' \frown \langle(\gamma,\checkmark)\rangle \wedge \exists t' \bullet S_1(s \frown \langle(t',\checkmark)\rangle, t', \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\})\\
\wedge\ \checkmark \notin \sigma(s') \wedge \checkmark \notin \sigma(\aleph \uparrow [t',\gamma))
\end{array}\right\} \Rightarrow S(s,\gamma,\aleph)\\
\hline
P_1 \underset{t}{\lightning} P_2\ \mathbf{sat}\ S(s,\gamma,\aleph)
\end{array}$$

Rule $P_1 \underset{t}{\lightning} P_2$ special case version

$$
\begin{array}{l}
P_1 \ \mathbf{sat} \ \checkmark \notin \sigma(s) \\
P_1 \ \mathbf{sat} \ S_1(s, \gamma, \aleph) \\
P_2 \ \mathbf{sat} \ S_2(s, \gamma, \aleph) \\
\left.\begin{array}{l}
(S_1(s \restriction t, end(s \restriction t), \aleph \restriction t \cup [0, t) \times \{\checkmark\}) \\
\wedge\ S_2(s \mathbin{\dot{-}} (t + \delta), \gamma \mathbin{\dot{-}} (t + \delta), \aleph \mathbin{\dot{-}} (t + \delta)) \\
\wedge\ s \uparrow (t, t + 2\delta) = \langle\rangle \wedge \gamma \geqslant end(s)
\end{array}\right\} \Rightarrow S(s, \gamma, \aleph) \\
\hline
P_1 \underset{t}{\lightning} P_2 \ \mathbf{sat} \ S(s, \gamma, \aleph)
\end{array}
$$

4.7 Soundness and Completeness

Both soundness and completeness of the set of inference rules for TM_{FI} follow directly from the clauses of the semantic equation defining $\mathcal{E}_{TI}$, in exactly the same way as soundness and completeness followed for the proof system for TM_F

5 Aspects of Good Behaviour

In this section we first define three important aspects of desirable behaviour: non-retraction, responsiveness, and promptness. For each of the definitions, we examine which syntactic operators preserve it; whether it is expressible as a behavioural specification and hence a subject for tbe proof systems; whether the defiuition is closed; and in some cases, the circumstances under which they are preserved by hiding and alphabet parallel, the constructors used in modelling communication. We also examine the interaction of the properties we have defined. We make a number of definitions which will be useful for dealing with timewise refinements, and with communication. Many of tbese properties are defined on TM_F; they can be immediately extended to apply to processes in TM_{FS}, since we have the identity $\mathcal{F}_T[\![P]\!] = fail(\mathcal{E}_T[\![P]\!])$. Hence a specification on TM_F will apply equally to processes in TM_{FS} by applying the definition to the set of failures of the TM_{FS} process.

One of the most useful facts we can know about a process is that it is *non-retracting*: that once it has offered to follow some course of action then that offer will remain until either it is accepted or some other observable event occurs. An imperceptible change of state will not cause the offer to be retracted. In addition to being desirable for its own sake, we will see that this property will be valuable (in conjunction with promptness or bounded stabiIity) when consideriug timewise refinements of parallel compositions (see e.g. theorem 6.3.27), timewise refinements of networks (e.g. theorem 7.4.5), and when chaining bnffers together.

Another useful property is *responsiveness*, a strong form of liveness. It requires of a process that there is a bound on the time by which a response must be forthcoming. A process is (t, A)-responsive if it will always offer an event from the set A within time t. We will illustrate the use of this property in specification and verification in chapter 8.

The specification *promptness*, like responsiveness, places a bound on the length of time it is necessary to wait for a process to respond, but it does not require that the process is live; it specifies that if the process must respond, then it will do so within a bounded time. This may hold of nnstable processes, but otherwise the notion is closely related to stability: if a process responds, then it must do so by the time it becomes stable. We later define *t-stability*, which holds of processes that always stabilise within time t. Hence a t-stable process will always be prompt (see theorem 5.6.6). As we have mentioned, this property in conjunction with non-retraction will be useful in timewise refinement

Other specifications defined in this chapter are *impartiality* and *limitedness*. A process is impartial on a set A if it treats each element of that set in the same way: at any moment, either the entire set is available, or it is refusible. This will be useful in identifying prompt buffers. A process is limited on a set if there is a bound on the number of events it can perform from that set without performing some event from outside the set. Limitedness is a property required to enable timewise refinement for the hiding operator (see lemmas 6.3.31, 6.3.33).

This chapter presents a catalogue of definitions and laws concerning these concepts and some variants. Examples of their application in both specification and verification are presented in chapter 8.

5.1 Non-retraction

The concept of non-retraction is very much a state-based concept rather than one concerning the set of possible observable behaviours of a process. The intuitive idea we are trying to capture is that once a process is in a particular state where an event is on offer, then it cannot undergo an internal change of state whereby that offer is no longer made. However, this state-based notion cannot be captured by a predicate on the semantics of processes. Consider how the notion of non-retraction would apply to the process

$$R \;\; \widehat{=} \;\; STOP \sqcap ((a \rightarrow STOP) \overset{2}{\triangleright} STOP) \sqcap (a \rightarrow STOP)$$

If we say that there are three states in which R could be, one of which will retract the offer of a after two units of time, then clearly there is a state of R where the retraction of an offer can occur.

In contrast, we would like to allow that the process Q defined by

$$Q \;\; \widehat{=} \;\; STOP \sqcap (a \rightarrow STOP)$$

is non-retracting, since there is no possible state for Q in which the retraction of an offer is possible.

It turns out that $\mathcal{F}_T[\![R]\!]$ is equal to $\mathcal{F}_T[\![Q]\!]$ (even though *operationally* R is not equal to Q). Any specification that captures non-retraction must have that R is non-retracting if and only if Q is, since their semantics are equal.

This issue arises because we cannot deduce which state R is in from a given failure. If it is possible after a particular failure to either accept or reject a (for example, for R at time 1, after the failure $(\langle\rangle, \emptyset)$), and yet we do not offer a at that time, then we cannot reason about the subsequent behaviour of the process depending on whether or not it would have accepted. We think of a failure of a process as a particular experiment on that process; the trace is the result of the experiment where the refusal set has been offered to the process. Semantically, a process is defined to be the set of all possible behaviours. Hence for any particular failure $(s, \aleph)$ if (t, a) is not in $\aleph$ or in s, then we have no way of saying that a *would* have happened at t, unless all possible failures which are identical to $(s, \aleph)$ up to t cannot refuse a at time t. In other words, saying that a would have occurred at t if it had been offered is equivalent to saying that there is no state in which a is refusible at time t. So a non-retracting process will be one which, if it can refuse an event a at time t, must have been able to refuse that event continuously since the last visible event. Hence it is only *obliged* to continuously offer an event or set of events after it is certain to have offered it.

This leads us to our definition of non-retraction.

Definition 5.1.1 *A TCSP process P is non-retracting if for any $u \frown w \in traces(P)$ such that $begin(w) \geqslant t_2 > t_1 \geqslant end(u)$, and $Y \subseteq \Sigma$ we have*

$$(u \frown w, ([t_1, t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!] \Rightarrow (u \frown w, ([end(u), t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]$$

Example

The process

$$P \mathrel{\hat=} a \rightarrow WAIT\ 3\ ; b \rightarrow STOP$$

is non-retracting. It has the behaviour

$$(\langle (1, a), (5, b) \rangle, [3, 4) \times \{b\} \cup [9, 12) \times \{c\})$$

as a possible failure, so we may conclude that

$$(\langle (1, a), (5, b) \rangle, [1, 4) \times \{b\} \cup [9, 12) \times \{c\})$$

is a possible failure. The fact that b is refusible for some time following the performance of the event a allows us to deduce that it is also refusible from the time the a was performed. We also have that the failure

$$(\langle (1, a), (5, b) \rangle, [4, 7) \times \{a\})$$

is a possible behaviour of P, and hence that

$$(\langle (1, a), (5, b) \rangle, [1, 7) \times \{a\})$$

is a possible behaviour.

It is useful to be able to identify non-retracting processes from their syntax.

Theorem 5.1.2 *If P,Q and $P(a)(a \in A)$ are all non-retracting, then so are the following:*

$$\bot,\ STOP,\ SKIP,\ WAIT\ t,\ WAIT\ t\,;P,\ a \to P,\ a : A \to P(a),$$
$$P \sqcap Q,\ \textstyle\bigsqcap_{a \in A} P(a),\ P \,\Box\, Q,\ P \parallel Q,\ P\ {}_X\!\parallel_Y Q,\ P \,|||\, Q,\ f(P),\ f^{-1}(P).$$

We show later that the specification *non-retracting* is closed in TM_F and so it follows, if F preserves non-retraction, that $\mu X \bullet F(X)$ is non-retracting.

Observe that even if P and Q are non-retracting, $P \setminus X$ and $P\,;Q$ need not be non-retracting in general, since they both hide events whose visibility may have been necessary to ensure the non-retraction of P. For example, $((WAIT\ 1\,; a \to STOP) \,\Box\, b \to STOP)$ is non-retracting, since the offer of b will only be withdrawn if a is performed. If we now hide the performance of a then we obtain

$$((WAIT\ 1\ ; a \to STOP) \,\Box\, b \to STOP) \setminus a \equiv (b \to STOP) \overset{1}{\triangleright} STOP$$

which retracts the offer of b after 1 unit of time. For the same reasons, timeout and tireont do not in general preserve non-retraction.

Generalisations

We generalise the concept of non-retraction in several ways. We may be concerned only with a subset X of the events P is capable of doing, requiring only that no offers from the set X will be retracted. The specification *non-retracting on* X is defined as follows:

Definition 5.1.3 *P is* non-retracting on X *if for any trace $u^\frown w \in traces(P)$, $begin(w) \geqslant t_2 > t_1 \geqslant end(u)$, and $Y \subseteq X$ we have*

$$(u^\frown w, ([t_1, t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!] \Rightarrow (u^\frown w, ([end(u), t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]$$

A non-retracting process is non-retracting on any subset of Σ, and more generally any process that is non-retracting on X is also non-retracting on any set $Y \subseteq X$.

Another form of non-retraction is *strong non-retraction* on a set X. A process is strongly non-retracting on X if it will not retract an offer from the set X until an event from that set is performed.

Definition 5.1.4 *A process P is* strongly non-retracting on X *if for any trace $u\frown w \in traces(P)$, $begin(w) \geqslant t_2 > t_1 \geqslant end(u)$, and $Y \subseteq X$ we have*

$$\begin{aligned}&(u\frown w, ([t_1, t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]\\ &\quad \Rightarrow \quad (u\frown w, ([end(u \upharpoonright X), t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]\end{aligned}$$

Example

The process

$$(a \rightarrow STOP) \mid\mid\mid ((b \rightarrow STOP) \overset{1}{\triangleright} c \rightarrow d \rightarrow STOP)$$

is strongly non-retracting on $\{a\}$. We have that

$$(\langle(1, a), (3, c), (6, d)\rangle, [4, 5) \times \{a\})$$

is a failure of P, with a refusihle over an interval, so we may conclude that a is refusible from the previous occurrence of a, and deduce that

$$(\langle(1, a), (3, c), (6, d)\rangle, [1, 5) \times \{a\})$$

is also a possible behaviour of P.

Strong non-retraction may be generalised to the concept of *non-retraction on X until Z*, where offers from X cannot be retracted until an event in Z occurs. We insist that $X \subseteq Z$, since we would wish the acceptance of an offered event from X to allow the retractiou of that offer, siuce it has been accepted.

Definition 5.1.5 *A process P is* non-retracting on X until Z *if for any $u\frown w \in traces(P)$, $begin(w) \geqslant t_2 > t_1 \geqslant end(u)$, and $Y \subseteq X$ we have*

$$\begin{aligned}&(u\frown w, ([t_1, t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]\\ &\quad \Rightarrow \quad (u\frown w, ([end(u \upharpoonright Z), t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!]\end{aligned}$$

Example

Tbe process

$$P \mathrel{\widehat{=}} (b \rightarrow STOP) \mathbin{|||} (a \rightarrow STOP \mathbin{\Box} c \rightarrow d \rightarrow STOP)$$

is non-retracting on $\{a\}$ until $\{a, c\}$. We have that

$$(\langle (1, c), (3, b), (6, d) \rangle, [4, 5) \times \{a, b\})$$

is a failure of P, so we may deduce that

$$(\langle (1, c), (3, b), (6, d) \rangle, [1, 5) \times \{a\} \cup [4, 5) \times \{b\})$$

is also a possible behaviour: the event a is refusible from the last occurrence of an event from $\{a, c\}$. We infer nothing further about the refusal of b.

It follows that if P is strongly non-retracting on X, then for each $x \in X$ we have that P is non-retracting on x until X.

A final generalisation of the concept is *non-retraction when* S, where S is a predicate on traces. We may wish that a process is non-retracting only under certain conditions, characterised by S. For example, we may wish that a buffer is non-retracting only when it is empty. The condition on the trace that captures this requirement is that the number of outputs is equal to the number of inputs.

The notion of non-retracting when S also generalises the other versions of non-retraction. We define it for the most general case:

Definition 5.1.6 *A process* P *is* non-retracting on X until Z when S *if for any* $u^\frown w \in traces(P)$*,* $begin(w) \geqslant t_2 > t_1 \geqslant end(u)$*, and* $Y \subseteq X$ *we have*

$$\begin{aligned} &(u^\frown w, ([t_1, t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!] \wedge S(u) \\ &\quad \Rightarrow \ (u^\frown w, ([end(u \upharpoonright Z), t_2) \times Y) \cup \aleph) \in \mathcal{F}_T[\![P]\!] \end{aligned}$$

Example

The process

$$\mu X \bullet a \rightarrow (b \rightarrow X \overset{5}{\triangleright} STOP)$$

is non-retracting on $\{a, b\}$ until $\{a, b\}$ when $s = \langle\rangle \vee last(s) = b$.

This is the most general form of non-retraction; all the other forms are special cases of it:

- Non-retraction is non-retraction on Σ until Σ when $TRUE$
- non-retraction on X is non-retraction on X until Σ when $TRUE$
- strong non-retraction on X is non-retraction on X until X when $TRUE$
- non-retraction on X until Z is non-retraction on X until Z when $TRUE$
- non-retraction when S is non-retraction on Σ nntil Σ when S
- non-retraction on X when S is non-retraction on X until Σ when S; and strong non-retraction on X when S is non-retraction on X until X when S

Hence in order to show that each of the varieties of non-retraction is closed in TM_F we need only prove it for the most general case. This involves a straightforward application of Reed's theorem 9.6.3 applied to TM_F, which states that if a predicate S is such that

$$\neg S(P) \Rightarrow \exists\, t \in [0, \infty) \bullet (P(t) = Q(t) \Rightarrow \neg S(Q))$$

then S is closed in TM_F.

Theorem 5.1.7 *The predicate 'non-retracting on X until Z when S' is closed.*

Proof If a process P is *not* non-retracting on X until Z when S, then

$$\begin{aligned}&\exists\, u^\frown w \in traces(P), begin(w) \geqslant t_2 > t_1 \geqslant end(u), \aleph, Y \subseteq X \bullet\\ &\quad (u^\frown w, [t_1, t_2) \times Y \cup \aleph) \in \mathcal{F}_T[\![P]\!] \wedge S(u) \wedge\\ &\quad (u^\frown w, [end(u \restriction Z), t_2) \times Y \cup \aleph) \notin \mathcal{F}_T[\![P]\!]\end{aligned}$$

Now let Q be an arbitrary TM_F process such that

$$Q(max\{end(u^\frown w), t_2\} + 1) \;=\; P(max\{end(u^\frown w), t_2\} + 1)$$

It is clear that

$$\begin{aligned}&\exists\, u^\frown w \in traces(Q), begin(w) \geqslant t_2 > t_1 \geqslant end(u), Y \subseteq X \bullet\\ &\quad (u^\frown w, [t_1, t_2) \times Y \cup \aleph) \in \mathcal{F}_T[\![Q]\!] \wedge S(u) \wedge\\ &\quad (u^\frown w, [end(s \restriction Z), t_2) \times Y \cup \aleph) \notin \mathcal{F}_T[\![Q]\!]\end{aligned}$$

so we conclude that Q is not non-retracting on X until Z when S. Therefore the predicate 'non-retracting on X until Z when S' is closed for any X, Z and S. □

It follows immediately that all the particular cases of it are also closed.

Most of the constructors that preserve non-retraction also preserve the more general *non-retracting on X until Z.*

Theorem 5.1.8 *If P, Q and $P(a) (a \in A)$ are all non-retracting on X until Z, then so are the following:*

$$\bot,\ STOP,\ SKIP,\ WAIT\ t,\ WAIT\ t\,;P,\ a \rightarrow P,\ a : A \rightarrow P(a),$$
$$P \sqcap Q,\ \textstyle\bigsqcap_{a \in A} P(a),\ P \Box Q,\ P \parallel Q,\ P\ {}_X\|_Y\ Q,\ P \mid\mid\mid Q$$

Further, if $inits(P) \cap X = \emptyset$ then $P \overset{t}{\triangleright} Q$ is non-retracting on X until Z.

In general, if P is non-retracting on X until Z, alphabet transformation function f does not yield non-retracting on $f(X)$ until $f(Z)$. Consider

$$P \ \hat{=}\ (a \rightarrow STOP \overset{1}{\triangleright} STOP)$$

Then P is non-retracting on $\{b\}$ until $\{b\}$. But if the function f has $f(a) = f(b) = c$, then $f(P)$ is not non-retracting on $\{c\}$ until $\{c\}$.

However, we do obtain the following result:

Theorem 5.1.9 *If process P is non-retracting on $f^{-1}(X)$ until $f^{-1}(Z)$, then $f(P)$ is non-retracting on X until Z.*

Hence, if f is one-one, then P non-retracting on X until Z does imply $f(P)$ is non-retracting on $f(X)$ until $f(Z)$

The case for inverse functions is simpler:

Theorem 5.1.10 *If P is non-retracting on X until Z, then $f^{-1}(P)$ is non-retracting on $f^{-1}(X)$ until $f^{-1}(Z)$*

The most general form of non-retraction, *non-retracting on X until Z when S*, is not preserved by the interleaving operator or by the alphabetised parallel operator. For example, the processes

$$\begin{aligned} P &\ \hat{=}\ a \rightarrow STOP \overset{1}{\triangleright} STOP \\ Q &\ \hat{=}\ b \rightarrow STOP \overset{1}{\triangleright} STOP \end{aligned}$$

are non-retracting when $s \neq \langle\rangle$, but the processes $P \mid\mid\mid Q$ and $P\ {}_{\{a\}}\|_{\{b\}}\ Q$ are not non-retracting when $s \neq \langle\rangle$. We have $(\langle(0, a)\rangle, [1, 2) \times \{b\}) \in \mathcal{F}_T[\![P \mid\mid\mid Q]\!]$ and $\langle(0, a)\rangle \neq \langle\rangle$, but $(\langle(0, a)\rangle, [0, 2) \times \{b\}) \notin \mathcal{F}_T[\![P \mid\mid\mid Q]\!]$. The same failures illustrate the case of the parallel operator.

The process constructors which do preserve non-retraction on X until Z when S are given by the followiug:

Theorem 5.1.11 *If P, Q and $P(a)(a \in A)$ are all non-retracting on X until Z when $S(s)$, then so are the following:*

$$\perp,\ STOP,\ SKIP,\ WAIT\ t,\ WAIT\ t\,;P,$$
$$P \sqcap Q,\ \textstyle\bigsqcap_{a \in A} P(a),\ P \mathbin{\Box} Q,\ P \parallel Q$$

In addition, $a \to P$ and $a : A \to P(a)$ are non-retracting on X until Z when $S(tail(s))$. Further, if f is one-one, then $f(P)$ is non-retracting on $f(X)$ until $f(Z)$ when $S(f^{-1}(s))$.

As before, if f is not one-one then this is not necessarily the case.

Theorem 5.1.12 *Non-retraction cannot be written as a behavioural specification.*

Proof An application of theorem 4.4.2 will yield this result. Defining

$$\begin{array}{rcl} P & \widehat{=} & (a \to STOP) \underset{t}{\lightning} STOP \\ Q & \widehat{=} & STOP \end{array}$$

we have $P \sqcap Q$ is non-retracting, but P is not. □

It follows that none of the generalisations of non-retraction can be written as behavioural specifications.

Theorem 5.1.13 *Non-retraction has the following properties:*

1. *If P is non-retracting on A until B, and $A \subseteq X \setminus Y$, $B \subseteq X$, then $P\ {}_X\|_Y\ Q$ is non-retracting on A until B*
2. *If P is non-retracting on A until B, and $C \cap B = \emptyset$, then $P \setminus C$ is non-retracting on A until B*
3. *If P is strongly non-retracting on A and $A \subseteq X \setminus C$, then $(P\ {}_X\|_Y\ Q) \setminus C$ is strongly non-retracting on A*

4. *P is non-retracting on A and non-retracting on B if and only if P is non-retracting on $A \cup B$*

5. *If Q is strongly non-retracting on A, and $\sigma(P) \cap A = \emptyset$, then $P \underset{t}{\lightning} Q$ is strongly non-retracting on A*

The following technical results will be required later (see e.g. theorem 5.3.12, theorem 6.3.27):

Lemma 5.1.14 *If P is non-retracting on Y, then*

$$(s, \aleph) \in \mathcal{F}_T[\![P]\!] \quad \Rightarrow \quad (s, \aleph') \in \mathcal{F}_T[\![P]\!]$$

where $\aleph' = \{(t, a) \mid a \in Y \wedge \exists\, t' \geqslant t \geqslant end(s) \bullet (t', a) \in \aleph\}$

Proof This follows straightforwardly from the fact that $\aleph$ is a finite union of refusal tokens, so the definition of non-retraction will apply to each of them in turn. □

Corollary 5.1.15 *If P is non-retracting on Y, and $T > end(s), T' > 0$, then*

$$\begin{aligned}(s, \aleph) \in \mathcal{F}_T[\![P]\!] \wedge A \subseteq \sigma(\aleph \uparrow [T, T + T')) \cap Y \\ \Rightarrow (s, \aleph \cup [end(s), T) \times A) \in \mathcal{F}_T[\![P]\!]\end{aligned}$$

Proof Follows immediately from lemma 5.1.14, (where $[end(s), T) \times A) \subseteq \aleph'$). □

Lemma 5.1.16 *If P is non-retracting on Y, and $A \subseteq Y, T > end(s), T' > 0$, then*

$$(s, \aleph) \in \mathcal{SI}_T[\![P]\!] \wedge A \subseteq \sigma(\aleph \uparrow [T, T + T')) \quad \Rightarrow \quad (s, \aleph \cup [end(s), T) \times A) \in \mathcal{SI}_T[\![P]\!]$$

Proof

$$\forall\, t \geqslant T + T' \bullet (s, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!] \wedge A \subseteq \sigma(\aleph \upharpoonright t \uparrow [T, T + T')$$

so by corollary 5.1.15 we obtain

$$\forall\, t \geqslant T + T' \bullet (s, \aleph \upharpoonright t \cup [end(s), T) \times A) \in \mathcal{F}_T[\![P]\!]$$

so

$$(s, \aleph \cup [end(s), T) \times A) \in \mathcal{SI}_T[\![P]\!]$$

as required. □

5.2 Responsiveness

Responsiveness on a set of events A is a particularly strong form of liveness concerning that set. To state that a process will respond within time t to the set of events A is to say that it cannot refnse to perform all of that set over an interval of length t. We call such a process (t, A)-responsive.

Definition 5.2.1 *A process P is (t, A)-responsive if it satisfies*

$$\forall (s, \aleph) \in \mathcal{F}_T[\![P]\!], T \in [0, \infty) \bullet [T, T+t) \times A \subseteq \aleph \Rightarrow (s \upharpoonright A) \uparrow [T, T+t) \neq \langle\rangle$$

Example

The process

$$\mu X \bullet a \rightarrow X$$

is $(1, \{a\})$-responsive (when $2\delta < 1$).

Corollary 5.2.2 *In behavioural specification form we have that a process P is (t, A)-responsive if*

$$P \quad \textbf{sat} \quad \forall T \in [0, \infty) \bullet ([T, T+t) \times A \subseteq \aleph \Rightarrow (s \upharpoonright A) \uparrow [T, T+t) \neq \langle\rangle)$$

Recall that for a process to refuse a set A over a particular time interval, on a given trace, does not imply that the trace performs no event in A over that interval; rather it means that it can perform no more events in A than it has in fact performed. In the case of the definition for responsiveness it implies that at least one event in A can always be performed during an interval of length t.

Clearly a process which is capable of terminating is not (t, A)-responsive for any t,A, since after termination it will refuse A for all time. This raises the question of which processes *are* (t, A)-responsive.

Theorem 5.2.3 *If P,Q and $P(a)(a \in B)$ are (t, A)-responsive for some t $(> \delta)$ and A, then we have:*

- *$a \rightarrow P$ is $(t + \delta, A \cup \{a\})$-responsive (which is $(t + \delta, A)$-responsive if $a \in A$)*
- *$WAIT\ t'$; P is $(t + t', A)$-responsive*
- *$a : B \rightarrow P(a)$ is $(t + \delta, B)$-responsive if $A \subseteq B, A \neq \emptyset$*
- *$\prod_{a \in B} P(a)$ is (t, A)-responsive*

- *WAIT* t' ; $a : B \rightarrow P(a)$ *is* $(max\{t'', t + \delta\}, B)$*-responsive if* $A \subseteq B, A \neq \emptyset, t'' > t'$
- $P \Box Q$, $P \sqcap Q$ *and* $P \mid\mid\mid Q$ *are all* (t, A)*-responsive*
- *If* $t' > t$ *and* $B \supseteq A$ *then* P *is* (t', B)*-responsive*

We can write (t, A)-responsive as a behavioural specification, so it is continuous, and so we obtain the result that if $F \circ WAIT\,\delta$ (F composed with *WAIT delta*) preserves (t, A)-responsive, then $\mu X \bullet F(X)$ is (t, A)-responsive.

Generalisations

As we remarked earlier, a process capable of terminating cannot be responsive. However, in many cases such a process will be sequentially composed with another process, and in such circumstances we are not concerned with its behaviour following termination. We therefore generalise the notion of responsiveness to *responsive before termination*.

Definition 5.2.4 *A process P is* (t, A)*-responsive until termination if*

$$P \quad \mathbf{sat} \quad \forall\, T \in [0, \infty) \bullet (([T, T + t) \times A \cup [0, T + t) \times \{\checkmark\}) \subseteq \aleph \\ \Rightarrow (s \upharpoonright A) \uparrow [T, T + t] \neq \langle\rangle \vee \checkmark \in \sigma(s))$$

Example

The process

$$a \rightarrow WAIT\ 3\ ; b \rightarrow SKIP$$

is $(4, \{a, b\})$-responsive until termination. However, the process

$$a \rightarrow WAIT\ 3\ ; b \rightarrow STOP$$

is not responsive until termination, since termination must be successful.

Theorem 5.2.5 *All the process constructors that preserve* (t, A)*-responsive also preserve* (t, A)*-responsive until termination. In addition, if P and Q are* (t, A)*-responsive until termination, and R is* (t, A)*-responsive, then we have*

- *SKIP is* (t, A)*-responsive until termination, for any* $t > 0$
- *WAIT* t *is* (t', A)*-responsive until termination, for any* $t' > t$

- *$P \overset{t'}{\rhd} Q$ is $(2t + \delta, A)$-responsive until termination*
- *$P \underset{t'}{\lightning} Q$ is $(2t + 2\delta, A)$-responsive until termination*
- *$P \,;\, R$ is $(2t, A)$-responsive*
- *$P \mathbin{\mathring{;}} R$ is $(2t + \delta, A)$-responsive*

If we are concerned that a process has a bound on response on set A, but we are not concerned with the value of that bound, then we say that the process is A-responsive.

Definition 5.2.6 *A TCSP process P is A-responsive if $\exists\, t \in [0, \infty)$ such that P is (t, A)-responsive.*

Theorem 5.2.7 *If P and Q, are A-responsive then so are*

$$a \to P,\ P \mathbin{\Box} Q,\ P \sqcap Q,\ WAIT\ t\,;P,\ P \mathbin{|||} Q,\ P\,;\,Q$$

To say a process is responsive is to say that it will respond to some event from Σ.

Definition 5.2.8 *A proeess is responsive if it is Σ-responsive.*

Definition 5.2.9 *A proeess P is A-responsive until termination if $\exists\, t \in [0, \infty)$ such that P is (t, A)-responsive until termination.*

Theorem 5.2.10 *If $a \in A$, then if P and Q are A-responsive until termination, then so are*

$$SKIP,\ WAIT\,t,\ a \to P,\ P \mathbin{\Box} Q\ P \sqcap Q,$$
$$WAIT\ t\,;P,\ P \mathbin{|||} Q,\ P \mathbin{\mathring{;}} Q,\ P \overset{t}{\rhd} Q,\ P \underset{t}{\lightning} Q$$

Further, if $A \cap B \neq \emptyset$ then $a : B \to P(a)$ is A-responsive until termination.

The predicate A-responsive is not closed. For instance, if $F(X) \mathrel{\widehat{=}} WAIT\ 1\,;X$ then $\mu\, X \bullet F(X) = \bot$, which is not $\{a\}$-responsive although each of the approximations $F^n(\mu\, X \bullet a \to X)$ is $\{a\}$-responsive. It follows that A-responsiveness cannot be expressed as a behavioural specification. The function F also preserves A-responsive until termination, so it follows that the predicate A-responsive until termination is not closed, and so it cannot be expressed as a behavioural specification.

Theorem 5.2.11 *Responsiveness has the following properties:*

1. *If P is an A-responsive process and $A \subseteq X \setminus Y$ then $P \,{}_X\|_Y\, Q$ is an A-responsive process*
2. *If P is an A-responsive process and $A \cap B = \emptyset$, then $P \setminus B$ is A-responsive*

We are often interested in the responsiveness of a process immediately following an action, especially if the process will be used with an interrupt operator (see e.g. section 8.2): where we are not concerned with behaviour of the argument process following the interrupt. We say that a process is *immediately responsive* if it cannot refuse a set over an entire time interval immediately after its last event.

Definition 5.2.12 *A process P is* immediately (t, A)-responsive *if*

$$P \quad \mathbf{sat} \quad [end(s), end(s) + t) \times A \not\subseteq \aleph$$

Example

The process

$$\mu X \bullet ((a \rightarrow X) \overset{1}{\triangleright} STOP)$$

is immediately $(3, \{a\})$ responsive, since it cannot refuse a for the 3 time units following the occurrence of an a, (or before it has done anything).

Theorem 5.2.13 *If P, Q and $P(a)$ are all immediately (t, A)-responsive, then so are the following:*

- $a \rightarrow P$
- $a : B \rightarrow P(a)$ *if* $A \cap B \neq \emptyset$
- $\sqcap P(a)$, $P \sqcap Q$, $P \,\Box\, Q$
- $WAIT\ t'\ ;\ a : B \rightarrow P(a)$ *if* $t' < t, A \cap B \neq \emptyset$
- $P \,|||\, Q$
- $P \overset{t'}{\triangleright} Q$ *if* $t' \geqslant t$

Further,

- $f^{-1}(P)$ *is immediately* $(t, f^{-1}(A))$*-responsive*

- *if P is immediately $(t, f^{-1}(A))$-responsive, then $f(P)$ is immediately (t, A)-responsive*

The behavioural nature of its definition entails that the predicate 'immediately (t, A)-responsive' is closed.

The following results will prove useful later.

Theorem 5.2.14 *If P is immediately (t, A)-responsive, $t' \geqslant t$, $B \supseteq A$, then P is immediately (t', B)-responsive.*

Theorem 5.2.15 *If P is immediately (t, A)-responsive, and Q is (t', A)-responsive, then $P \underset{t}{\lightning} Q$ is $(t + t' + 2\delta, A)$-responsive*

Theorem 5.2.16 *If P is immediately (t, A)-responsive, and Q is immediately (t', A)-responsive, then $P \underset{t}{\lightning} Q$ is immediately $(t + t' + 2\delta, A)$-responsive*

Theorem 5.2.17 *If P is immediately (t, A)-responsive, and alphabet transformation f preserves "immediately (t, A)-responsive", then the recursive process $\mu X \bullet P \underset{t}{\lightning} f(X)$ is $(2t + 3\delta, A)$-responsive*

5.3 Promptness

We define promptness as follows:

Definition 5.3.1 *A process P is t-*prompt *if for any $(s, \aleph) \in \mathcal{F}_T[\![P]\!]$ we have:*

$$T \geqslant end(s) \wedge [T, T + t) \times X \subseteq \aleph \;\Rightarrow\; (s, \aleph \cup [T, \infty) \times X) \in \mathcal{SI}_T[\![P]\!]$$

This definition specifies that if the process can refuse the set X over an interval of length t, then it can refuse X for all time.

Example

The process

$$P \mathrel{\widehat=} (a \to STOP) \overset{2}{\rhd} b \to STOP$$

is 3-prompt: from the information that

$$(\langle\rangle, [0, 1) \times \{b\} \cup [3, 6) \times \{a\}) \in \mathcal{F}_T[\![P]\!]$$

we may conclude that

$$(\langle\rangle, [0, 1) \times \{b\} \cup [3, \infty) \times \{a\}) \in \mathcal{SI}_T[\![P]\!]$$

However, P is not 1-prompt; it is not the case that

$$(\langle\rangle, [0, \infty) \times \{b\} \cup [3, 6) \times \{a\}) \in \mathcal{SI}_T[\![P]\!]$$

We may not conclude that $\{b\}$ is infinitely refusible from the observation that it is refused for 1 time unit.

We generalise the definition:

Definition 5.3.2 *A process is* prompt *if it is t-prompt for some t.*

A weaker version will be sufficient to establish some timewise refinement results in the next chapter:

Definition 5.3.3 *A process P is* weakly t-prompt *if for any $s \in traces(\mathcal{F}_T[\![P]\!])$ we have:*

$$\begin{aligned} T' \geqslant end(s) + t \wedge T \leqslant T' - t \wedge (s, [T, T') \times X) \in \mathcal{F}_T[\![P]\!] \\ \Rightarrow (s, [T, \infty) \times X) \in \mathcal{SI}_T[\![P]\!] \end{aligned}$$

Example

The process

$$(a \to STOP) \overset{2}{\triangleright} b \to STOP$$

is weakly 3-prompt. We may conclude from

$$(\langle(2, b)\rangle, [1, 5) \times \{b\}) \in \mathcal{F}_T[\![P]\!]$$

that

$$(\langle(2, b)\rangle, [1, \infty) \times \{b\}) \in \mathcal{SI}_T[\![P]\!]$$

However, in contrast with the previous example, weak 3-promptness is not strong enough to conclude

$$(\langle\rangle, [0, 1) \times \{b\} \cup [3, \infty) \times \{a\}) \in \mathcal{SI}_T[\![P]\!]$$

from

$$(\langle\rangle, [0, 1) \times \{b\} \cup [3, 6) \times \{a\}) \in \mathcal{F}_T[\![P]\!]$$

Definition 5.3.4 *A process is* weakly prompt *if it is weakly t-prompt for some t.*

We may strengthen the definition so that event a is not possible after its refusal for length of time t:

Definition 5.3.5 *A process P is* strongly t-prompt *if*

$$P \quad \mathbf{sat} \quad ([T, T+t) \times \{a\} \subseteq \aleph \wedge s \uparrow (T, T+t) = \langle\rangle) \Rightarrow \mathit{first}(s \upharpoonright (T+t)) \neq a$$

It is clear that a process which is strongly t-prompt is also t-prompt, since it follows from the axioms of TM_F that if a timed event is not possible, then it must be refusible. However, the converse is not the case: the process

$$STOP \overset{2}{\triangleright} (STOP \sqcap a \rightarrow STOP)$$

is 1-prompt, but it is not strongly 1-prompt.

Theorem 5.3.6 *If P, Q, and $P(a) (a \in A)$ are t-prompt ($t > \delta$) then we have*

- $P \sqcap Q$, $P \,\Box\, Q$, $P \,|||\, Q$, $\sqcap_{a \in A} P(a)$, $f(P)$, $f^{-1}(P)$ *are all t-prompt*
- $STOP$, $\bot$, $SKIP$ *are t-prompt*
- $a \rightarrow P$, $a : A \rightarrow P(a)$ *are $t + \delta$-prompt*
- $WAIT\ t'$ *is t prompt when* $t' \leqslant t$
- $P \,;\, Q$ *is $2t + \delta$-prompt*
- $WAIT\ t' \,;\, P$ *is $t' + t$-prompt*

However, for any T, $P \parallel Q$ need not be T-prompt, since they could fail to synchronise for intervals longer than T. For example, defining

$$\begin{array}{rcl} P & \widehat{=} & \mu X \bullet (a \rightarrow STOP) \overset{1}{\triangleright} (STOP \overset{3}{\triangleright} X) \\ Q & \widehat{=} & (WAIT\ 2 \,;\, P) \underset{T}{↯} a \rightarrow STOP \end{array}$$

we have P and Q are both 4-prompt, but the process $P \parallel Q$ is not t-prompt for any $t \leqslant T$, since P and Q are unable to synchronise within T time units but are guaranteed to synchronise eventually.

Generalisations

We may generalise the specification t-prompt in the following ways:

Definition 5.3.7 *A process P is* t-prompt on X *if for any* $(s, \aleph) \in \mathcal{F}_T\llbracket P \rrbracket$*:*

$$\forall\, T \geqslant end(s),\, Y \subseteq X \bullet [T, T+t) \times Y \subseteq \aleph \;\;\Rightarrow\;\; (s, \aleph \cup [T, \infty) \times Y) \in \mathcal{SI}_T\llbracket P \rrbracket$$

Example

The process

$$a \rightarrow WAIT\; 3\,;\, b \rightarrow a \rightarrow STOP$$

is 1-prompt on a but not on b.

Theorem 5.3.8 *t-prompt enjoys the following properties:*

- *P is t-prompt on $A \cup B$ if and only if it is t-prompt on A and on B*
- *If P is t-prompt on A and $A \subseteq X \setminus Y$ then $P\;{}_X\|_Y\; Q$ is t-prompt on A.*

Proof The only non-trivial case is that of proving P is t-prompt on A and on B implies P is t-prompt on $A \cup B$: Assume P is t-prompt on A and on B. Consider $(s, \aleph) \in \mathcal{F}_T\llbracket P \rrbracket$ with $T \geqslant end(s) \wedge [T, T+t) \times (A \cup B) \subseteq \aleph$. Then

$$\begin{array}{ll}
 & (s, \aleph \cup [T, T+t) \times A \cup [T, \infty) \times B) \in \mathcal{SI}_T\llbracket P \rrbracket \\
\Rightarrow & \forall\, T' \bullet (s, \aleph \cup [T, T+t) \times A \cup [T, T') \times B) \in \mathcal{F}_T\llbracket P \rrbracket \\
\Rightarrow & \forall\, T' \bullet (s, \aleph \cup [T, \infty) \times A \cup [T, T') \times B) \in \mathcal{SI}_T\llbracket P \rrbracket \\
\Rightarrow & \forall\, T' \bullet (s, \aleph \cup [T, T') \times A \cup [T, T') \times B) \in \mathcal{F}_T\llbracket P \rrbracket \\
\Rightarrow & \forall\, T' \bullet (s, \aleph \cup [T, T') \times (A \cup B)) \in \mathcal{F}_T\llbracket P \rrbracket \\
\Rightarrow & (s, \aleph \cup [T, \infty) \times (A \cup B)) \in \mathcal{SI}_T\llbracket P \rrbracket
\end{array}$$

□

Definition 5.3.9 *A process P is* t-prompt on X when S *if for any* $(s, \aleph) \in \mathcal{F}_T\llbracket P \rrbracket$ *:*

$$\forall\, T \geqslant end(s) \bullet S(s) \wedge [T, T+t) \times X \subseteq \aleph \;\;\Rightarrow\;\; (s, \aleph \cup [T, \infty) \times X) \in \mathcal{SI}_T\llbracket P \rrbracket$$

We may also define the notions of

- t-prompt on X when S
- strongly t-prompt on X
- strongly t-prompt when S and
- strongly t-prompt on X when S

in the obvious way.

Example

The process

$$\mu X \bullet \begin{array}{l} (b \to ((b \to X) \sqcap a \to X) \\ \Box\, c \to \mathit{WAIT}\ 3\ ;\ a \to X) \end{array}$$

is both 1-prompt on $\{a\}$ when $\mathit{last}(s) = b$, and strongly 1-prompt on $\{a\}$ when $\mathit{last}(s) = b$. However, it is neither 1-prompt, nor 1-prompt on $\{a\}$.

Theorem 5.3.10 *It is not possible to write t-prompt as a behavioural specification.*

Proof Consider the processes P and Q defined by

$$\begin{array}{rcl} P & \mathrel{\widehat=} & \mathit{WAIT}\ 4\ ;\ a \to \mathit{STOP} \\ Q & \mathrel{\widehat=} & \mathit{STOP} \end{array}$$

Then $P \sqcap Q$ is 2-prompt, but P is not, so the result follows from theorem 4.4.2. □

It follows that none of the generalisations can be written as a behavioural specification. However, recall that strong t-promptness is expressible as a behavioural specification, and so it is closed.

The other forms of t-promptness are also continuous. To prove this, we consider only the most general form: t-prompt on X when S.

Theorem 5.3.11 *t-prompt on X when S is closed*

Proof Assume that P is a process which is not t-prompt on X when S. Then

$$\begin{array}{l} \exists (s, \aleph) \in \mathcal{F}_T[\![P]\!], T \geqslant \mathit{end}(s), Y \subseteq X \bullet \\ \quad S(s) \wedge [T, T+t) \times Y \subseteq \aleph \wedge (s, \aleph \cup [T, \infty) \times Y) \notin \mathcal{F}_T[\![P]\!] \end{array}$$

But if $(s, \aleph \cup [T, \infty) \times Y) \notin \mathcal{SI}_T[\![P]\!]$, then

$$\exists T' > T \bullet (s, \aleph \cup [T, T') \times Y) \notin \mathcal{SI}_T[\![P]\!]$$

Define $T'' = 1 + max\{end(s, \aleph), T'\}$. Then $P(T'') = Q(T'') \Rightarrow Q$ is not t-prompt on X when S. Hence by Reed's theorem 9.6.3 (see page 41) we conclude that the specification 't-prompt on X when S' is closed. □

Hence all the forms of promptness are closed.

Theorem 5.3.12 *If P and Q are prompt, and P is non-retracting on A and Q is non-retracting on B and $A \cup B = X \cap Y$, then $P\ {}_X\|_Y\ Q$ is prompt*

Proof Let P be t_1-prompt, and Q be t_2-prompt. We will prove that $P\ {}_X\|_Y\ Q$ is $t_1 + t_2$-prompt. Consider $(s, \aleph) \in \mathcal{SI}_T[\![P\ {}_X\|_Y\ Q]\!]$, and that $[T, T + t_1 + t_2) \times Z \subseteq \aleph$ for some $T \geqslant end(s)$. Without loss of generality, assume $Z \subseteq X \cup Y$. Define

$$\begin{array}{rcl} s_1 & = & s \upharpoonright X \\ s_2 & = & s \upharpoonright Y \end{array}$$

Then

$$\begin{array}{rl} \exists \aleph_1, \aleph_2 \bullet & (s_1, \aleph_1) \in \mathcal{SI}_T[\![P]\!] \wedge (s_2, \aleph_2) \in \mathcal{SI}_T[\![Q]\!] \wedge \aleph = \aleph_1 \cup \aleph_2 \wedge \\ & \aleph_1 \upharpoonright (X \setminus Y) = \aleph \upharpoonright (X \setminus Y) \wedge \aleph_2 \upharpoonright (Y \setminus X) = \aleph \upharpoonright (Y \setminus X) \end{array}$$

and so

$$\begin{array}{rl} & [T, T + t_1 + t_2) \times (Z \cap (X \setminus Y)) \subseteq \aleph_1 \\ \wedge & [T, T + t_1 + t_2) \times (Z \cap (Y \setminus X)) \subseteq \aleph_2 \end{array}$$

and so

$$\begin{array}{rl} & (s_1, \aleph_1 \cup [T, \infty) \times (Z \cap (X \setminus Y))) \in \mathcal{SI}_T[\![P]\!] \\ \wedge & (s_2, \aleph_2 \cup [T, \infty) \times (Z \cap (Y \setminus X))) \in \mathcal{SI}_T[\![Q]\!] \end{array}$$

Define

$$\begin{array}{rcl} Z_1 & = & X \cap Y \cap Z \cap \sigma(\aleph_1 \uparrow [T + t_1, T + t_1 + t_2)) \\ Z_2 & = & X \cap Y \cap Z \cap \{a \mid [T + t_1, T + t_1 + t_2) \times \{a\} \subseteq \aleph_2\} \\ Z_3 & = & X \cap Y \cap Z \cap \sigma(\aleph_2 \uparrow [T + t_1, T + t_1 + t_2)) \\ Z_4 & = & X \cap Y \cap Z \cap \{a \mid [T + t_1, T + t_1 + t_2) \times \{a\} \subseteq \aleph_1\} \end{array}$$

Then lemma 5.1.15 yields that

$$\begin{array}{ll} & (s_1, \aleph_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T, T + t_1) \times Z_1) \in \mathcal{F}_T [\![P]\!] \\ \Rightarrow & (s_1, \aleph_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T + t_1, T + t_1 + t_2) \times Z_1 \cup Z_4) \in \mathcal{F}_T [\![P]\!] \\ \Rightarrow & (s_1, \aleph_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T + t_1, \infty) \times Z_1 \cup Z_4) \in \mathcal{F}_T [\![P]\!] \end{array}$$

By symmetry we also obtain that

$$(s_2, \aleph_2 \cup [T, \infty) \times (Z \cap (Y \setminus X)) \cup [T + t_1, \infty) \times Z_2 \cup Z_3) \in \mathcal{F}_T [\![Q]\!]$$

and so

$$\left(s, \begin{array}{l} [T, \infty) \times (Z \cap (X \setminus Y) \cup [T, \infty) \times (Z \cap (Y \setminus X)) \\ \cup \aleph_1 \cup \aleph_2 \cup [T + t_1, \infty) \times (Z_1 \cup Z_2 \cup Z_3 \cup Z_4) \end{array} \right) \in \mathcal{SI}_T [\![P \ {}_X\|_Y \ Q]\!]$$

thus we obtain

$$(s, \aleph \cup [T, \infty) \times Z) \in \mathcal{F}_T [\![P \ {}_X\|_Y \ Q]\!]$$

as required. □

Lemma 5.3.13 *If P and Q are weakly prompt, and P is non-retracting on A and Q is non-retracting on B and $A \cup B = X \cap Y$, then $P \ {}_X\|_Y \ Q$ is weakly prompt*

Proof The proof is entirely similar to that for theorem 5.3.12

Corollary 5.3.14 *If P and Q are prompt, and P is non-retracting on $X \cap Y$, then $P \ {}_X\|_Y \ Q$ is prompt*

5.4 Impartiality

A process which is impartial on a set A always makes all its offers from set A at the same time.

A process P is impartial on a set A if it either offers all of the set or none of it.

Definition 5.4.1 *A process P is* impartial on A *if*

$$\begin{array}{l} \forall (s, \aleph) \in \mathcal{SI}_T [\![P]\!], a \in A \ \bullet \ (s^\frown \langle (t, a) \rangle, \aleph) \in \mathcal{SI}_T [\![P]\!] \Rightarrow \\ \qquad \forall b \in A \bullet (s^\frown \langle (t, b) \rangle, \aleph \cup (I(\aleph \upharpoonright A) \times A) \in \mathcal{SI}_T [\![P]\!] \end{array}$$

Example

The process

$$\mu X \bullet (a \rightarrow X \,\Box\, b \rightarrow X \,\Box\, c \rightarrow X)$$

is impartial on $\{a, b, c\}$, but not on $\{a, b, c, d\}$.

Every process is impartial on the empty set, and on singleton sets.

Theorem 5.4.2 *If P, Q, and $P(a)$ are all impartial on A, then so are the following:*

- $STOP$, $\bot$
- $SKIP$, $WAIT\,t$ *if* $\checkmark \notin A$ *or* $A \subseteq \{\checkmark\}$
- $P \,\Box\, Q$, $P \sqcap Q$, $\bigsqcap_a P(a)$
- $P \parallel Q$, $P \mid\mid\mid Q$, $P \fatsemi Q$, $WAIT\,t \,;P$
- $a : B \rightarrow P(a)$ *if* $A \subseteq B$ *or* $A \cap B = \emptyset$
- $P\;{}_X\|_Y\; Q$ *if* $(A \subseteq X \vee A \cap X = \emptyset$ *and* $(B \subseteq Y \vee B \cap Y = \emptyset$
- $P \setminus B$ *if* $A \subseteq B$ *or* $A \cap B = \emptyset$
- $P \overset{t}{\rhd} Q$ *and* $P \underset{t}{\lightning} Q$
- $f^{-1}(P)$ *is impartial on* $f^{-1}(A)$
- *If* P *is impartial on* $f^{-1}(A)$ *then* $f(P)$ *is impartial on* A

Theorem 5.4.3 *"Impartial on A" is closed*

Proof Consider $\{P_n\} \rightarrow P$, with each P_n impartial on A. Now let

$$(s^\frown\langle(t, a)\rangle, \aleph) \in \mathcal{SI}_T\llbracket P\rrbracket$$

Given $T(> t)$ we have

$$\begin{aligned}
& \exists\, N_T \bullet n > N_T \Rightarrow P_n \restriction T = P \restriction T \\
\Rightarrow\;& (s^\frown\langle(t, a)\rangle, \aleph \restriction T) \in \mathcal{SI}_T\llbracket P_n\rrbracket \\
\Rightarrow\;& \forall\, b \in A \bullet (s^\frown\langle(t, b)\rangle, \aleph \restriction T \cup (I((\aleph \restriction T) \restriction A) \times A) \in \mathcal{SI}_T\llbracket P_n] \\
\Rightarrow\;& \forall\, b \in A \bullet (s^\frown\langle(t, b)\rangle, \aleph \restriction T \cup (I((\aleph \restriction T) \restriction A) \times A) \in \mathcal{SI}_T\llbracket P]
\end{aligned}$$

This is true for all T, so

$$\forall b \in A \bullet (s ^\frown \langle (t, b) \rangle, \aleph \cup (I(\aleph \upharpoonright A) \times A) \in \mathcal{SI}_T \llbracket P \rrbracket$$

as required. □

Theorem 5.4.4 *If P is responsive on X, Q is responsive on Y, P or Q is impartial on $X \cap Y$, and P or Q is non-retracting, then $P\ {}_X\|_Y\ Q$ is responsive on $X \cup Y$*

Proof We will prove the case where P is impartial on $X \cap Y$ and Q is non-retracting. The other cases are proved in a similar way. Let P be (t_1, X)-responsive, and let Q be (t_2, Y)-responsive. We will prove that $P\ {}_X\|_Y\ Q$ is $(t_1 + t_2, X \cup Y)$-responsive.

Assume not, for a contradiction. Then for some $(s, \aleph) \in \mathcal{F}_T \llbracket P\ {}_X\|_Y\ Q \rrbracket$, and for some $T \in [0, \infty)$, we have

$$([T, T + t_1 + t_2) \times X \cup Y) \subseteq \aleph \wedge (s \upharpoonright (X \cup Y) \uparrow [T, T + t_1 + t_2)) = \langle\rangle$$

and without loss of generality we may assume

$$end(s) \leqslant T + t_1 + t_2 \wedge end(\aleph) \leqslant T + t_1 + t_2$$

Hence $end(s) \leqslant T$. Define

$$\begin{aligned} s_1 &= s \upharpoonright X \\ s_2 &= s \upharpoonright Y \end{aligned}$$

Then

$$\begin{aligned} \exists \aleph_1, \aleph_2 \bullet \quad & \aleph = \aleph_1 \cup \aleph_2 \wedge (s_1, \aleph_1) \in \mathcal{F}_T \llbracket P \rrbracket \wedge (s_2, \aleph_2) \in \mathcal{F}_T \llbracket Q \rrbracket \\ & \wedge \aleph \upharpoonright (X \setminus Y) = \aleph_1 \upharpoonright (X \setminus Y) \wedge \aleph \upharpoonright (Y \setminus X) = \aleph_2 \upharpoonright (Y \setminus X) \end{aligned}$$

and so

$$[T, T + t_1 + t_2) \times (X \setminus Y) \subseteq \aleph_1 \ \wedge \ [T, T + t_1 + t_2) \times (Y \setminus X) \subseteq \aleph_2$$

We have two cases to consider:

Case $[T + t_2, T + t_2 + t_1) \subseteq I(\aleph_1 \upharpoonright (X \cap Y))$

$$\begin{aligned} &\Rightarrow \ (s_1, [T + t_2, T + t_2 + t_1) \times X) \in \mathcal{F}_T \llbracket P \rrbracket \\ &\Rightarrow \ (s_1 \upharpoonright X) \uparrow (T + t_2, T + t_2 + t_1) \neq \langle\rangle \\ &\Rightarrow \ (s \upharpoonright (X \cup Y)) \uparrow (T, T + t_2 + t_1) \neq \langle\rangle \end{aligned}$$

which yields a contradiction.

Case $[T + t_2,\ T + t_2 + t_1) \not\subseteq I(\aleph_1 \upharpoonright (X \cap Y))$

$$\begin{aligned}
&\Rightarrow \exists [t', t'') \subseteq [T + t_2, T + t_2 + t_1) \bullet [t', t'') \times (X \cap Y) \subseteq \aleph_2 \\
&\Rightarrow [t', t'') \times Y \subseteq \aleph_2 \\
&\Rightarrow (s_2, [end(s_2), t'') \times Y) \in \mathcal{F}_T[\![Q]\!] \\
&\Rightarrow (s_2, [T, T + t_2) \times Y) \in \mathcal{F}_T[\![Q]\!] \\
&\Rightarrow (s_2 \upharpoonright Y) \uparrow (T, T + t_2] \neq \langle\rangle \\
&\Rightarrow (s \upharpoonright (X \cup Y)) \uparrow (T, T + t_2 + t_1] \neq \langle\rangle
\end{aligned}$$

which yields a contradiction. □

5.5 Limited on A

A process is limited on A if there is a bound on the amount of internal chatter from A it may perform when the set A is hidden.

Definition 5.5.1 *A TCSP process P is* n-limited *on A if*

$$\forall s \in traces(\mathcal{F}_T[\![P \setminus A]\!]), w \in traces(\mathcal{F}_T[\![P]\!]) \bullet w \setminus A = s \Rightarrow \#(w \upharpoonright end(s)) < n$$

Example

The process

$$\mu X \bullet a \to b \to c \to X$$

is 3-limited on $\{a, b\}$. However, the process

$$\mu X \bullet a \to X \,\Box\, b \to X$$

is not n-limited on $\{a\}$ for any n; there is no bound on the amount of possible internal chatter.

Theorem 5.5.2 *n-limited on A is closed in TM_F*

Proof By an application of Reed's theorem 9.6.3 (see page 41)

Definition 5.5.3 *A TCSP process P is* limited on *A if there is some n such that P is n-limited on A*

Theorem 5.5.4 *'Limited' enjoys the following properties:*

- *Every process is limited on $\emptyset$.*
- *If P is limited on A and $A \subseteq X$ then $P\ {}_X\|_Y\ Q$ is limited on A*
- *If P is limited on $A \cup B$ then $P \setminus A$ is limited on B.*

We also define a weaker notion, where the bound on chatter may depend on the observations:

Definition 5.5.5 *A TCSP process P is weakly limited on A if*

$$\forall s \in traces(\mathcal{F}_T[\![P \setminus A]\!]), \exists n, \forall w \in traces(\mathcal{F}_T[\![P]\!]) \bullet w \setminus A = s \Rightarrow \#w < n$$

Example

The counter process

$$\mu\, Y \bullet (\mu\, X \bullet up \rightarrow (X \,\text{;}\, down \rightarrow SKIP) \sqcap down \rightarrow SKIP) \,\text{;}\, Y$$

has the number of possible *down* events it can perform bounded by the number of *up* events that have occurred. It is therefore weakly limited on $\{down\}$, but it is not limited on $\{down\}$.

Clearly any process that is limited on A is also weakly limited on A

Theorem 5.5.6 *If P is limited on A, Q is limited on B, and $A \cap Y = B \cap X = \emptyset$, then $P\ {}_X\|_Y\ Q$ is limited on $A \cup B$*

Proof We have

$$\exists n \bullet w \in traces(P) \wedge s \in traces(P \setminus A) \wedge w \setminus A = s \Rightarrow \#w \restriction end(s) < n$$

and

$$\exists m \bullet w \in traces(Q) \wedge s \in traces(Q \setminus A) \wedge w \setminus A = s \Rightarrow \#w \restriction end(s) < m$$

Now consider

$$w \in traces(P\ {}_X\|_Y\ Q) \wedge s \in traces((P\ {}_X\|_Y\ Q) \setminus A \cup B) \wedge w \setminus A \cup B = s$$

Since

$$A \cap Y = B \cap X = \emptyset$$

we have

$$s \in traces(P \setminus A \ {}_{X \setminus A}\|_{Y \setminus B} \ Q \setminus B)$$

so

$$s \upharpoonright X = s \upharpoonright (X \setminus A) \in traces(P \setminus A)$$

Now $w \setminus A \cup B = s$ so

$$s \upharpoonright X = (w \setminus A \cup B) \upharpoonright X = (w \setminus A) \upharpoonright X = (w \upharpoonright X) \setminus A$$

Now

$$w \upharpoonright X \in traces(P)$$

and so

$$\#(w \upharpoonright X) \upharpoonleft end(s \upharpoonright X) < n$$

By symmetry,

$$\#(w \upharpoonright Y) \upharpoonleft end(s \upharpoonright Y) < m$$

Hence

$$\#(w \upharpoonright X \cup Y) \upharpoonleft max\{end(s \upharpoonright X), end(s \upharpoonright Y)\} < m + n$$

and so $\#w \upharpoonleft end(s) < m + n$ as required. □

Theorem 5.5.7 *If P is limited on $A \cup B$ and is weakly prompt, then $P \setminus A$ is limited on B and is weakly prompt.*

Proof Let P be n-limited on $A \cup B$. We first prove that $P \setminus A$ is n-limited on B. Consider s and w such that

$$s \in traces(P \setminus A \setminus B) \wedge w \in traces(P \setminus A) \wedge w \setminus B = s$$

Then

$$\exists\, u \bullet u \in traces(P) \wedge u \setminus A = w$$

Then

$$u \setminus (A \cup B) = s$$

so

$$\#u \upharpoonleft end(s) < n$$

so $\#w \upharpoonleft end(s) < n$. □.

Now let P be weakly t-prompt. We will prove that $P \setminus A$ is weakly nt-prompt. For $s \in traces(\mathcal{F}_T[\![P \setminus A]\!])$ let

$$T' \geqslant end(s) + nt, T \leqslant T' - nt, (s, [T, T') \times X) \in \mathcal{F}_T[\![P \setminus A]\!]$$

From lemma A.1.2 we have that

$$\exists\, w \bullet (w, [T, T') \times X \cup [0, T') \times A) \in \mathcal{F}_T[\![P]\!] \wedge w \setminus A = s$$

Since P is n-limited, and $T' \geqslant end(s) + nt$, we have that

$$\exists\, T'' \bullet T \leqslant T'' \leqslant T' - t \wedge w \uparrow (T'', T'' + t] = \langle\rangle$$

But then

$$(w \upharpoonright (T'' + t), [T, T'' + t) \times X \cup [0, T'' + t) \times A) \in \mathcal{F}_T[\![P]\!]$$

so by weak promptness of P we obtain

$$(w \upharpoonright (T'' + t), [T, \infty) \times X \cup [0, \infty) \times A) \in \mathcal{SI}_T[\![P]\!]$$

so $(s, [T, \infty) \times X) \in \mathcal{SI}_T[\![P]\!]$ as required. □

Theorem 5.5.8 *If P is responsive on B and limited on A, and $A \subseteq B$, then $P \setminus A$ is responsive on $B - A$.*

The definitions may also be extended to *CSP* processes:

Definition 5.5.9 *A CSP process P is* n-limited on A in M_T *if*

$$\forall\, tr \in \mathcal{T}[\![P]\!] \bullet (w \in A^* \wedge w \textbf{ in } tr) \Rightarrow \#w < n \bullet w \setminus A = s$$

A *CSP* process P is limited on A in M_T if there is some n such that P is n-limited on A in M_T

We also define a weaker version:

Definition 5.5.10 *A CSP process P is weakly limited on A in M_T if*

$$\forall\, s \in \mathcal{T}[\![P \setminus A]\!], \exists\, n, \forall\, w \in \mathcal{T}[\![P]\!] \bullet (w \setminus A = s \Rightarrow \#tr < n)$$

5.6 Bounded Stability

A process is stable if all of its behaviours are stable:

Definition 5.6.1 *A TCSP process P is* stable *if*

$$(s, \alpha) \in stab(\mathcal{E}_T[\![P]\!]) \Rightarrow \alpha < \infty$$

Definition 5.6.2 *A TCSP process P is* stable when S *if*

$$((s,\alpha) \in stab(\mathcal{E}_T [\![P]\!]) \wedge S(s)) \Rightarrow \alpha < \infty$$

A process P is t-stable if it always stabilises within length of time t of performing its last action.

Definition 5.6.3 *A TCSP process P is* t-stable *if*

$$(s,\alpha) \in stab(\mathcal{E}_T [\![P]\!]) \Rightarrow \alpha \leqslant end(s) + t$$

Example

The process

$$\mu X \bullet ((a \rightarrow X) \overset{2}{\triangleright} b \rightarrow a \rightarrow X)$$

is 3-stable.

Definition 5.6.4 *A process is* boundedly stable *if it is t-stable for some t.*

Theorem 5.6.5 *If P,Q, and $P(a)$ are t-stable, and $t' \leqslant t$, then*

- $P \parallel Q$, $P \;{}_X\!\|_Y\; Q$, $P \,|||\, Q$, $P \,\Box\, Q$, $P \sqcap Q$ *are all t-stable*
- $P \,\mathring{,}\, Q$ *is $(2t + \delta)$-stable*
- $a \rightarrow P$ *and* $a : A \rightarrow P(a)$ *are $(t + \delta)$-stable*
- $WAIT\, t' \,;\, a \rightarrow P$ *and* $WAIT\, t' \,;\, a : A \rightarrow P(a)$ *are $(t + \delta)$-stable*
- $f(P)$ *and* $f^{-1}(P)$ *are t-stable*
- $WAIT\, t' \,;\, P$ *is $(t + \delta)$-stable*

Further, t-stable is elosed in TM_{FS}.

Theorem 5.6.6 *If P is t-stable then it is t'-prompt for any $t' > t$*

Theorem 5.6.7 *If P is boundedly stable and weakly limited on A, then $P \setminus A$ is stable.*

Proof Let P he t-stable. Consider $s \in traces(P \setminus A)$. Then

$$\exists\, n \bullet w \in traces(P) \wedge w \setminus A = s \Rightarrow \#w \upharpoonleft end(s) < n$$

We will prove that

$$(s, \alpha, \emptyset) \in \mathcal{E}_T[\![P \setminus A]\!] \Rightarrow \alpha \leqslant end(s) + nt$$

By the definition of the hiding operator, we have

$$(w, \alpha, [0, \beta) \times A) \in \mathcal{E}_T[\![P]\!]$$

for some α, β such that $end(w) \leqslant \beta \leqslant \alpha$. Now if

$$\langle (t_1, a_1), (t_2, a_2) \rangle \text{ in } w \upharpoonleft end(s)$$

then axiom 8 of TM_{FS} yields that $t_2 \leqslant t_1 + t$, since if $t_2 > t_1 + t$ then

$$(w \upharpoonright ((t_2 + t_1 + t)/2), [0, (t_2 + t_1 + t)/2) \times A)$$

is a failure of P, whose stability value is less than $(t_2 + t_1 + t)/2$, and so

$$(w \upharpoonright ((t_2 + t_1 + t)/2)^\frown \langle (t_2, a_2) \rangle, [0, (t_2 + t_1 + t)/2) \times A)$$

is not a failure of P. This contradicts the fact that $(w \upharpoonright t_2, [0, t_2) \times A)$ is a failure of P.

Hence

$$end(w) \leqslant end(s) + \#(w \upharpoonleft end(s))t \leqslant end(s) + (n - 1)t$$

and so

$$(w, \alpha, \emptyset) \in \mathcal{E}_T[\![P]\!] \Rightarrow \alpha \leqslant end(s) + nt$$

so β defined above is less than $end(s) + nt$.

Therefore $(s, \alpha, \emptyset) \in \mathcal{E}_T[\![P \setminus A]\!] \Rightarrow \alpha < \infty$, and so $P \setminus A$ is stable. □

Theorem 5.6.8 *If P is boundedly stable and limited on A, then $P \setminus A$ is boundedly stable.*

Proof The proof is similar to the proof of the previous theorem. □

6 Timewise Refinement

In producing a refinement relation $\sqsubseteq$ ('refined by') between the various models of Reed's hierarchy (see appendix B.1), we aim to provide a method of transforming proof obligations between models. The proof strategy is as follows: in order to prove $T(Q)$ for a specification T and process Q, we aim to find a specification S on a different (simpler) model, and a process P, such that

- $S(P)$
- $P \sqsubseteq Q$
- $\forall P, Q \bullet ((S(P) \wedge P \sqsubseteq Q) \Rightarrow T(Q))$

The tbird condition may be thought of as the corresponding refinement relation between S and T. Tbe establishment of the second and third conditions will reduce the proof obligation $T(Q)$ to the proof obligation $S(P)$. We illustrate this proof strategy in chapter 8, where it is used in the verification of each of three protocols.

The projection mappings between the various models of the hierarchy represent mappings of behavioural information. In moving from a higher to a lower model, we restrict tbe aspects of bebaviour we are able to describe. The mapping Π can be thought of as a translation of information about the possible behaviours of a process into a more restricted language, which can not talk about the behaviours in so much detail. For example, if Π is tbe mapping from TM_{FS} to M_T, then the information that $(\langle(1,a),(3,b)\rangle, 4, [0,3) \times \{b\})$ is a possible behaviour of a process in TM_{FS} translates under the mapping Π to the information that $\langle a, b\rangle$ is a possible behaviour in M_T.

In considering refinement relations between processes in different models in the bierarchy, we will need to use the projection mappings to translate the semantics of the process in the bigher model into sets of behaviours in the lower model, for comparison with the lower process. Our ability to make more detailed observations in the higher model will enable us to make finer distinctions, and thus exclude some behaviours allowed by the coarser model. For example, tbe information in M_T that event a is a possible first event in both P and Q yields that it is also a possible first event in $P \parallel Q$. But in models which also contain timing information, it may be apparent that there is no time at which P and Q could synchronise on a. We would therefore expect $\Pi(Q) \subseteq P$ (rather than the stronger condition $\Pi(Q) = P$) to be a sufficient condition for P to be refined by Q.

We are particularly interested in *timewise* refinement relations: those refinement relations between models which contain no timing information and those which do contain such information. Such a relation will hold between an untimed process and a *timewise refinement* of it. We will focus attention on two timewise refinement relations in particular, although we will define others at the end of the chapter.

We first define a mapping $\Theta : \mathit{TCSP} \rightarrow \mathit{CSP}$, which removes the timing information from the syntactic description of a process. We will be interested in conditions which yield $\Theta(Q) \sqsubseteq Q$, since in those cases the production of an untimed process P such that $P \sqsubseteq Q$ will reduce to an application of Θ to Q.

The first relation we will consider is the weak timewise refinement relation, $\sqsubseteq_t$. It is a relation between the untimed traces model M_T and the timed failures stability model TM_{FS}. We will see that our definition of $P \sqsubseteq_t Q$ is equivalent to $\Pi(Q) \subseteq P$. We obtain the result that $\Theta(Q) \sqsubseteq_t Q$ for all *TCSP* processes. We also extend $\sqsubseteq_t$ to a refinement relation between specifications; this leads to a notion of 'translation' of untimed behavioural specifications into timed ones.

The second refinement relation, $\sqsubseteq_f$, is strong timewise refinement. It will be between the failures model M_F and the timed failures stability model TM_{FS}. We will see that the projection mapping $\Pi : TM_{FS} \rightarrow M_F$ is too coarse for the refinement relation between M_F and TM_{FS}: while we still have that $\Pi(Q) \subseteq P \Rightarrow P \sqsubseteq_f Q$, there will also be some refinements Q of P where $\Pi(Q) \not\subseteq P$. It turns out that $\Theta(Q)$ is not $\sqsubseteq_f$ refined by Q in general, but we obtain useful conditions on Q for those cases where it does hold.

We will also define a subsidiary refinement relation which is intended as an aid to deciding when the refinement relation $\sqsubseteq_f$ holds between processes. The refinement relation $\sqsubseteq_{ts}$ is defined between M_F and TM_{FS}^*, and it is shown that for any *TCSP* process Q we have

$$P \sqsubseteq_{ts} \mathcal{E}_T^*[\![Q]\!] \Rightarrow P \sqsubseteq_f \mathcal{E}_T[\![Q]\!]$$

6.1 The mapping Θ

The mapping Θ removes the explicit timing information from the description of a *TCSP* process, mapping it to a *CSP* process.

Definition 6.1.1 *The function* $\Theta : TCSP \rightarrow CSP$ *is defined as follows:*

$$
\begin{array}{rcl}
\Theta(STOP) & \mathrel{\widehat=} & STOP \\
\Theta(\bot) & \mathrel{\widehat=} & STOP \\
\Theta(SKIP) & \mathrel{\widehat=} & SKIP \\
\Theta(WAIT\ t) & \mathrel{\widehat=} & SKIP \\
\Theta(a \rightarrow P) & \mathrel{\widehat=} & a \rightarrow \Theta(P) \\
\Theta(a : A \rightarrow P_a)) & \mathrel{\widehat=} & a : A \rightarrow \Theta(P_a)) \\
\Theta(P \sqcap Q) & \mathrel{\widehat=} & \Theta(P) \sqcap \Theta(Q) \\
\Theta(\bigsqcap_{i \in I} P_i)) & \mathrel{\widehat=} & \bigsqcap_{i \in I} \Theta(P_i) \\
\Theta(P \mathbin{\Box} Q) & \mathrel{\widehat=} & \Theta(P) \mathbin{\Box} \Theta(Q) \\
\Theta(P \parallel Q) & \mathrel{\widehat=} & \Theta(P) \parallel \Theta(Q) \\
\Theta(P \mathbin{{}_A\|_B} Q) & \mathrel{\widehat=} & \Theta(P) \mathbin{{}_A\|_B} \Theta(Q) \\
\Theta(P \mathbin{|||} Q) & \mathrel{\widehat=} & \Theta(P) \mathbin{|||} \Theta(Q) \\
\Theta(WAIT\ t\,; P) & \mathrel{\widehat=} & \Theta(P) \\
\Theta(P \,; Q) & \mathrel{\widehat=} & \Theta(P) \,; \Theta(Q) \\
\Theta(P \setminus A) & \mathrel{\widehat=} & \Theta(P) \setminus A \\
\Theta(f(P)) & \mathrel{\widehat=} & f(\Theta(P)) \\
\Theta(f^{-1}(P) & \mathrel{\widehat=} & f^{-1}(\Theta(P)) \\
\Theta(X) & \mathrel{\widehat=} & X \\
\Theta(\mu X \bullet P) & \mathrel{\widehat=} & \mu X \bullet \Theta(P)
\end{array}
$$

The definition of timeout yields the definition

$$\Theta(P \overset{t}{\triangleright} Q) \mathrel{\widehat=} \Theta(Q) \sqcap (\Theta(P) \mathbin{\Box} \Theta(Q))$$

Examples

$$\begin{aligned}&\Theta(\mathit{WAIT}\ 5 \Box a \rightarrow \mathit{SKIP})\,;\mathit{WAIT}\ 3\,; b \rightarrow \mathit{WAIT}\ 4)\\ &\quad = \ (\mathit{SKIP} \Box a \rightarrow \mathit{SKIP})\,; b \rightarrow \mathit{SKIP}\end{aligned}$$

$$\begin{aligned}&\Theta(\mu X \bullet (\mathit{WAIT}\ 3\,; a \rightarrow X) \overset{2}{\rhd} b \rightarrow X)\\ &\quad = \ \mu X \bullet ((b \rightarrow X) \sqcap (a \rightarrow X \Box b \rightarrow X))\end{aligned}$$

Observe in the second example that without the timing information we will be unable to deduce that timeout occurs before a becomes available.

6.2 Weak Timewise Refinement

The projection mapping $\Pi : TM_{FS} \rightarrow M_T$ is given by

$$\Pi(S) \mathrel{\widehat{=}} \mathit{tstrip}(\mathit{traccs}(S))$$

We first define the weak timewise relation $\sqsubseteq_t$ as a relation between M_T and TM_{FS} (both models reproduced in appeudix B.2):

Definition 6.2.1 *If $Q_1 \in M_T$ and $Q_2 \in TM_{FS}$, then*

$$Q_1 \sqsubseteq_t Q_2 \ \mathrel{\widehat{=}}\ \Pi(Q_2) \subseteq Q_1$$

We extend the definition to a relation between *CSP* environments and *TCSP* environments, as follows:

Definition 6.2.2

$$\sigma \sqsubseteq_t \rho \ \mathrel{\widehat{=}}\ \forall X : \mathit{var} \bullet \sigma(X) \sqsubseteq_t \rho(X)$$

We will extend the notation further, and define $\sqsubseteq_t$ to be relation between *CSP* and *TCSP*: if P is a *CSP* term, and Q is a *TCSP* term, then we write $P \sqsubseteq_t Q$ to mean that P is weakly refined by Q. This is defined as follows:

Definition 6.2.3 *If P is a CSP term, and Q is a TCSP term, then*

$$P \sqsubseteq_t Q \ \mathrel{\widehat{=}}\ \sigma \sqsubseteq_t \rho \Rightarrow \mathcal{T}[\![P]\!]\sigma \sqsubseteq_t \mathcal{E}_T[\![Q]\!]\rho$$

Observe that the *TCSP* process *STOP* will refine any *CSP* process.

We now consider which *TCSP* processes bave $\Theta(Q) \sqsubseteq_t Q$

Definition 6.2.4 *A TCSP term Q preserves $\sqsubseteq_t$-refinement if*

$$\sigma \sqsubseteq_t \rho \Rightarrow \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma \sqsubseteq_t \mathcal{E}_T\llbracket Q \rrbracket \rho$$

Lemma 6.2.5 *For any process $Q_1 \in M_T$, and any process $Q_2 \in TM_{FS}$, the predicates S_1 on TM_{FS} and S_2 on M_T defined by $S_1(X) \mathrel{\widehat{=}} Q_1 \sqsubseteq_t X$ and $S_2(X) \mathrel{\widehat{=}} X \sqsubseteq_t Q_2$ are both continuous, satisfiable predicates.*

Proof We deal first with S_1. Let $\{P_i\}$ be a convergent sequence of processes in TM_{FS} whose limit is P. Consider $s \in traces(P)$. Then $\exists\, i \bullet s \in traces(P_i)$, so $tstrip(s) \in Q_1$. Hence $\Pi(P) \subseteq Q_1$ as required. Further, we have $S_1(STOP)$, and so S_1 is both continuous and satisfiable.

We now turn our attention to S_2. Let $\{P_i\}$ be a convergent sequence of processes in M_T whose limit is P. Consider $s \in traces(Q_2)$. Then $\forall\, i \bullet tstrip(s) \in P_i$, and so $tstrip(s) \in P$. Hence we have $S_2(P)$, so S_2 is continuous. It is also satisfiable, since $S_2(RUN)$ (where $RUN = a : \Sigma \rightarrow RUN$). □

Lemma 6.2.6 *If $P \sqsubseteq_t Q$, then $\mu X \bullet P \sqsubseteq_t \mu X \bullet Q$*

Proof Assume $\sigma \sqsubseteq_t \rho$, and that $P \sqsubseteq_t Q$. Let $W_\delta(X)$ be the semantic function corresponding to $WAIT\,\delta\,; X$. Consider $P_1 \in TM_{FS}$ such that

$$\mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma \sqsubseteq_t P_1$$

Now $\Pi(P_1) = \Pi(W_\delta(P_1))$, so we have

$$\Pi(W_\delta(P_1)) \subseteq \mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma$$

so

$$\sigma[\mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma / X] \sqsubseteq_t \rho[W_\delta(P_1)/X]$$

and so

$$\mathcal{T}\llbracket P \rrbracket (\sigma[\mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma / X]) \sqsubseteq_t \mathcal{E}_T\llbracket Q \rrbracket (\rho[W_\delta(P_1)/X])$$

so

$$\mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma \sqsubseteq_t \mathcal{E}_T\llbracket Q \rrbracket (\rho[W_\delta(P_1)/X])$$

But

$$\mathcal{E}_T\llbracket \mu X \bullet Q \rrbracket \rho = fix(\lambda P_1 \bullet \mathcal{E}_T\llbracket Q \rrbracket (\rho[W_\delta(P_1)/X])$$

Defining

$$S_l(P_l) \mathrel{\widehat{=}} (\mathcal{T}\llbracket \mu X \bullet P \rrbracket \sigma \sqsubseteq_t P_l)$$

we have from lemma 6.2.5 that S_l is continuous and satisfiable, and we have just obtained that S_l is preserved by the contraction mapping above, whose fixed point is $\mathcal{E}_T\llbracket \mu X \bullet Q \rrbracket \rho$, which yields that $\mu X \bullet P \sqsubseteq_t \mu X \bullet Q$. □

Theorem 6.2.7 *All TCSP terms Q preserve $\sqsubseteq_t$-refinement.*

Proof By structural induction. We examine each case in turn, assuming that the syntactic subcomponents preserve $\sqsubseteq_t$-refinement, and that $\sigma \sqsubseteq_t \rho$.

$$\begin{aligned}
\Pi(\mathcal{E}_T\llbracket STOP \rrbracket \rho) = \{\langle\rangle\} &\subseteq \{\langle\rangle\} = \mathcal{T}\llbracket \Theta(STOP) \rrbracket \sigma \\
\Pi(\mathcal{E}_T\llbracket \bot \rrbracket \rho) = \{\langle\rangle\} &\subseteq \{\langle\rangle\} = \mathcal{T}\llbracket \Theta(\bot) \rrbracket \sigma \\
\Pi(\mathcal{E}_T\llbracket SKIP \rrbracket \rho) = \{\langle\rangle, \langle\checkmark\rangle\} &\subseteq \{\langle\rangle, \langle\checkmark\rangle\} = \mathcal{T}\llbracket \Theta(SKIP) \rrbracket \sigma \\
\Pi(\mathcal{E}_T\llbracket WAIT\ t \rrbracket \rho) = \{\langle\rangle, \langle\checkmark\rangle\} &\subseteq \{\langle\rangle, \langle\checkmark\rangle\} = \mathcal{T}\llbracket \Theta(WAIT\ t) \rrbracket \sigma
\end{aligned}$$

We next consider the one-place operators.

$$\begin{aligned}
\Pi(\mathcal{E}_T\llbracket a \to Q \rrbracket \rho) &= \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid tr \in \Pi(\mathcal{E}_T\llbracket Q \rrbracket \rho)\} \\
&\subseteq \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid tr \in \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma)\} \\
&\subseteq \mathcal{T}\llbracket \Theta(a \to Q) \rrbracket \sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T\llbracket WAIT\ t\,;Q \rrbracket \rho) &= \{tr \mid tr \in \Pi(\mathcal{E}_T\llbracket Q \rrbracket \rho)\} \\
&\subseteq \{tr \mid tr \in \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma)\} \\
&\subseteq \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T\llbracket Q \setminus A \rrbracket \rho) &= \{tr \setminus a \mid tr \in \Pi(\mathcal{E}_T\llbracket Q \rrbracket \rho)\} \\
&\subseteq \{tr \setminus A \mid tr \in \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma)\} \\
&\subseteq \mathcal{T}\llbracket \Theta(Q \setminus A) \rrbracket \sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T\llbracket f(Q) \rrbracket \rho) &= \{f(tr) \mid tr \in \Pi(\mathcal{E}_T\llbracket Q \rrbracket \rho)\} \\
&\subseteq \{f(tr) \mid tr \in \mathcal{T}\llbracket \Theta(Q) \rrbracket \sigma)\} \\
&\subseteq \mathcal{T}\llbracket \Theta(f(Q)) \rrbracket \sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![f^{-1}(Q)]\!]\rho) &= \{tr \mid f(tr) \in \Pi(\mathcal{E}_T[\![Q]\!]\rho)\} \\
&\subseteq \{tr \mid f(tr) \in \mathcal{T}[\![\Theta(Q)]\!]\sigma)\} \\
&\subseteq \mathcal{T}[\![\Theta(f^{-1}(Q))]\!]\sigma
\end{aligned}$$

We now consider the two place operators.

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![Q_1 \sqcap Q_2]\!]\rho) &= \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \cup \Pi(\mathcal{E}_T[\![Q_2]\!]\rho) \\
&\subseteq \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \cup \mathcal{T}[\![\Theta(Q_2)]\!]\sigma \\
&\subseteq \mathcal{T}[\![\Theta(Q_1 \sqcap Q_2)]\!]\sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![Q_1 \Box Q_2]\!]\rho) &= \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \cup \Pi(\mathcal{E}_T[\![Q_2]\!]\rho) \\
&\subseteq \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \cup \mathcal{T}[\![\Theta(Q_2)]\!]\sigma \\
&\subseteq \mathcal{T}[\![\Theta(Q_1 \Box Q_2)]\!]\sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![Q_1 \parallel Q_2]\!]\rho) &= tstrip(traces(\mathcal{E}_T[\![Q_1]\!]\rho) \cap traces(\mathcal{E}_T[\![Q_2]\!]\rho)) \\
&\subseteq \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \cap \Pi(\mathcal{E}_T[\![Q_2]\!]\rho) \\
&\subseteq \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \cap \mathcal{T}[\![\Theta(Q_2)]\!]\sigma \\
&\subseteq \mathcal{T}[\![\Theta(Q_1 \parallel Q_2)]\!]\sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![Q_1\, {}_A\|_B\, Q_2]\!]\rho) &= tstrip(\{s \mid s = s \upharpoonright A \cup B \wedge s \upharpoonright A \in traces(\mathcal{E}_T[\![Q_1]\!]\rho) \\
&\qquad\qquad \wedge\, s \upharpoonright B \in traces(\mathcal{E}_T[\![Q_2]\!]\rho)\}) \\
&\subseteq \{tr \mid tr = tr \upharpoonright A \cup B \wedge tr \upharpoonright A \in \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \\
&\qquad \wedge\, tr \upharpoonright B \in \Pi(\mathcal{E}_T[\![Q_2]\!]\rho)\} \\
&\subseteq \{tr \mid tr = tr \upharpoonright A \cup B \wedge tr \upharpoonright A \in \mathcal{T}_T[\![\Theta(Q_1)]\!]\sigma \\
&\qquad \wedge\, tr \upharpoonright B \in \mathcal{T}_T[\![\Theta(Q_2)]\!]\sigma\} \\
&\subseteq \mathcal{T}[\![\Theta(Q_1\, {}_A\|_B\, Q_2)]\!]\sigma
\end{aligned}$$

$$\begin{aligned}
\Pi(\mathcal{E}_T[\![Q_1 \,|||\, Q_2]\!]\rho) &= tstrip(\{s \mid \exists\, u, v \bullet u \in traces(\mathcal{E}_T[\![Q_1]\!]\rho) \\
&\qquad\qquad \wedge\, v \in traces(\mathcal{E}_T[\![Q_2]\!]\rho) \wedge s \in Tmerge(u, v)\}) \\
&\subseteq \{tr \mid \exists\, u, v \bullet u \in \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \wedge v \in \Pi(\mathcal{E}_T[\![Q_2]\rho) \\
&\qquad \wedge\, s \in Merge(u, v)\} \\
&\subseteq \{tr \mid \exists\, u, v \bullet u \in \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \wedge v \in \mathcal{T}[\![\Theta(Q_2)]\!]\sigma \\
&\qquad \wedge\, s \in Merge(u, v)\} \\
&\subseteq \mathcal{T}[\![\Theta(Q_1 \,|||\, Q_2)]\!]\sigma
\end{aligned}$$

$$\begin{array}{rcl}
\Pi(\mathcal{E}_T[\![Q_1 \fatsemi Q_2]\!]\rho) & \subseteq & tstrip(\{s \mid \checkmark \notin \sigma(s) \wedge s \in traces(\mathcal{E}_T[\![Q_1]\!]\rho)\} \\
 & & \qquad\qquad \cup \{(s^\frown(w + (t+\delta)) \mid \checkmark \notin \sigma(s) \wedge \\
 & & \qquad\qquad\qquad s^\frown\langle(t,\checkmark)\rangle \in traces(\mathcal{E}_T[\![Q_1]\!]\rho) \wedge \\
 & & \qquad\qquad\qquad w \in traces(\mathcal{E}_T[\![Q_2]\!]\rho)\} \\
 & \subseteq & \{tr \mid \checkmark \notin \sigma(tr) \wedge tr \in \Pi(\mathcal{E}_T[\![Q_1]\!]\rho)\} \\
 & & \cup\{tr^\frown tr' \mid \checkmark \notin \sigma(tr) \wedge tr^\frown\langle\checkmark\rangle \in \Pi(\mathcal{E}_T[\![Q_1]\!]\rho) \\
 & & \qquad\qquad \wedge\ w \in \Pi(\mathcal{E}_T[\![Q_2]\!]\rho)\} \\
 & \subseteq & \{tr \mid tr \in \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \wedge \checkmark \notin \sigma(tr)\} \\
 & & \cup\{tr^\frown tr' \mid tr^\frown\langle\checkmark\rangle \in \mathcal{T}[\![\Theta(Q_1)]\!]\sigma \wedge tr' \in \mathcal{T}[\![\Theta(Q_2)]\!]\sigma\} \\
 & \subseteq & \mathcal{T}[\![\Theta(Q_1)\,;\Theta(Q_2)]\!]\sigma \\
 & \subseteq & \mathcal{T}[\![\Theta(Q_1 \fatsemi Q_2)]\!]\sigma
\end{array}$$

We next consider the indexed operators.

$$\begin{array}{rcl}
\Pi(\mathcal{E}_T[\![a : A \rightarrow Q_a]\!]\rho) & = & \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid a \in A \wedge tr \in \Pi(\mathcal{E}_T[\![Q_a]\!]\rho)\} \\
 & \subseteq & \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid a \in A \wedge tr \in \mathcal{T}[\![\Theta(Q_a)]\!]\sigma\} \\
 & \subseteq & \mathcal{T}[\![\Theta(a : A \rightarrow Q_a)]\!]\sigma
\end{array}$$

$$\begin{array}{rcl}
\Pi(\mathcal{E}_T[\![\bigsqcap_{a\in A} Q_a]\!]\rho) & = & \bigcup_{a\in A} \Pi(\mathcal{E}_T[\![Q_a]\!]\rho) \\
 & \subseteq & \bigcup_{a\in A} \mathcal{T}[\![\Theta(Q_a)]\!]\sigma \\
 & \subseteq & \mathcal{T}[\![\Theta(\bigsqcap_{a\in A} Q_a]\!]\sigma
\end{array}$$

The process variable case is trivial:

$$\begin{array}{rcl}
\Pi(\mathcal{E}_T[\![X]\!]\rho) & = & \Pi(\rho(X)) \\
 & \subseteq & \sigma(X) \\
 & \subseteq & \sigma(\Theta(X)) \\
 & \subseteq & \mathcal{T}[\![\Theta(X)]\!]\sigma
\end{array}$$

The recursion case follows from lemma 6.2.6 :

$$\begin{array}{rcl}
\Theta(\mu\, X \bullet Q) & = & \mu\, X \bullet \Theta(Q) \\
 & \sqsubseteq_t & \mu\, X \bullet Q
\end{array}$$

□

Corollary 6.2.8 *Evcry TCSP process Q has $\Theta(Q) \sqsubseteq_t Q$*

For any *TCSP* process Q we have a simple method for producing a *CSP* process which is refined by Q. This will allow reduction of proof obligations on Q to proof obligations on $\Theta(Q)$.

Observe that the ordinary sequential composition operator ";" does not preserve timewise refinement in general, since it can introduce traces that are not present in the untimed version, by means of the closure operator. For example, the process

$$(a \rightarrow STOP \,|||\, SKIP)\,;\, b \rightarrow STOP$$

has $\langle (0, b), (0, a) \rangle$ as a possible trace, but the corresponding untimed trace, $\langle b, a \rangle$ is not a possible trace of the untimed equivalent of that process. This problem cannot arise when the delayed sequential composition operator is used.

The operator Θ extends in the obvious way to fuuctions on *TCSP* processes built out of the basic process constructors. We immediately obtain the following corollary:

Corollary 6.2.9 *If $\underline{P}$ is a vector of CSP processes, $\underline{Q}$ is a vector of TCSP processes, and $\forall i \bullet \underline{P}_i \sqsubseteq_t \underline{Q}_i$, and F is a function built from TCSP process constructors, then $\Theta(F)(\underline{P}) \sqsubseteq_t F(\underline{Q})$*

Follows immediately from theorem 6.2.7 by considering F as a *TCSP* term with free variables. □

Specifications

We can extend the refinement relation to a relation between predicates on untimed processes and predicates on timed processes. Our definition of the refinement relation between specifications is motivated by our proof strategy: if we know that $S_1 \sqsubseteq_t S_2$, then in order to establish $S_2(Q)$ it will be enough to establish $S_1(P)$ for some P such that $P \sqsubseteq_t Q$.

Definition 6.2.10 *If S_1 is a predicate on elements of M_T, and S_2 is a predicate on elements of TM_{FS}, then we define:*

$$S_1 \sqsubseteq_t S_2 \;\; \widehat{=} \;\; \forall Q \bullet (\exists P \bullet S_1(P) \wedge P \sqsubseteq_t Q \Rightarrow S_2(Q))$$

The following lemma provides an example.

Lemma 6.2.11 *'weakly limited on A'* $\sqsubseteq_t$ *'weakly limited on A'*

The predicate 'weakly limited on A' on M_T is refined by the predicate 'weakly limited on A' on TM_{FS}.

Our proof strategy is then encapsulated by the following theorem:

Theorem 6.2.12 *If $P \sqsubseteq_t Q$, $S \sqsubseteq_t T$ and $S(P)$, then $T(Q)$*

Proof Follows immediately from the definition of $\sqsubseteq_t$ on specifications □

The set of specifications $\{T \mid S \sqsubseteq_t T\}$, for a given S, is a complete lattice under the $\Leftarrow$ ordering. Its top element will therefore be the strongest refinement of S: it will be denoted $sr(S)$. Any specification weaker than $sr(S)$ will be a timewise refinement of S, and by its definition every timewise refinement of S is weaker than (or equivalent to) $sr(S)$. We therefore obtain that $sr(S)$ is the (unique) predicate that satisfies

$$S \sqsubseteq_t sr(S) \wedge \forall S_2 \bullet ((sr(S) \Rightarrow S_2) \Leftrightarrow S \sqsubseteq_t S_2)$$

It follows from the definition of $\sqsubseteq_t$ for specifications that T defined by

$$T(Q) \mathrel{\hat{=}} \exists P \bullet (S(P) \wedge P \sqsubseteq_t Q)$$

satisfies the above equation for $sr(S)$. Hence we obtain a characterisation for $sr(S)$ that will allow us to identify the strongest refinement of a given specification. This will be particularly straightforward for behavioural specifications, which will make our proof strategy immediately applicable to such specifications.

$$sr(S)(Q) \Leftrightarrow \exists P \bullet S(P) \wedge P \sqsubseteq_t Q$$

The predicate $sr(S)$ holds of precisely those processes which are refinements of processes captured by S.

Corollary 6.2.13 *If $S(\Theta(Q))$ then $sr(S)(Q)$*

Theorem 6.2.14 *If A is a predicate on untimed traces, then*

$$sr(X \textbf{ sat } A(tr))(P) \Leftrightarrow P \textbf{ sat } A'(s, \alpha, \aleph)$$

where $A'(s, \alpha, \aleph) \Leftrightarrow A(tstrip(s))$

Proof We must prove that

$$\exists P \bullet P \ \mathbf{sat}\ A(tr) \wedge P \sqsubseteq_t Q \Leftrightarrow Q \ \mathbf{sat}\ A'(s, \alpha, \aleph)$$

We prove first that

$$\exists P \bullet P \ \mathbf{sat}\ A(tr) \wedge P \sqsubseteq_t Q \Rightarrow Q \ \mathbf{sat}\ A'(s, \alpha, \aleph)$$

We have $P \sqsubseteq_t Q$, so $\Pi(\mathcal{E}_T[\![Q]\!]) \subseteq \mathcal{T}[\![P]\!]$. Also, $tr \in \mathcal{T}[\![P]\!] \Rightarrow A(tr)$. Now

$$\begin{array}{rcl} (s, \alpha, \aleph) \in \mathcal{E}_T[\![Q]\!] & \Rightarrow & s \in traces(\mathcal{E}_T[\![Q]\!]) \\ & \Rightarrow & tstrip(s) \in \mathcal{T}[\![Q]\!] \\ & \Rightarrow & A(tstrip(s)) \\ & \Rightarrow & A'(s, \alpha, \aleph) \end{array}$$

which yields the required result.

We now prove

$$Q \ \mathbf{sat}\ A'(s, \alpha, \aleph) \Rightarrow \exists P \bullet P \ \mathbf{sat}\ A(tr) \wedge P \sqsubseteq_t Q$$

Any $Q \in TCSP$ will have that $U = tstrip(traces(\mathcal{E}_T[\![Q]\!]))$ satisfies the trace axioms (immediate from axioms (1) and (2) of TM_{FS}). Now for a given trace $u = \langle u_1, u_2, \ldots, u_n \rangle$, define

$$P_u = u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_n \rightarrow STOP$$

Defining

$$P = \bigsqcap_{u \in U} P_u$$

we have $\mathcal{T}[\![P]\!] = U$, and so P **sat** $A(tr)$ and $P \sqsubseteq_t Q$. □

Example

The theorem allows the translation of a behavioural specification by an alteration to its syntax. For example:

$$sr(X \ \mathbf{sat}\ \#(tr \upharpoonright out) \leqslant \#(tr \upharpoonright in)) \ = \ X \ \mathbf{sat}\ \#(tstrip(s) \upharpoonright out) \leqslant \#(tstrip(s) \upharpoonright in)$$

We may also define a function on timed predicates complementary to the strongest refinement function. The set of predicates on untimed processes refined by a given timed predicate T is a complete lattice under the implication order, so we may identify the weakest predicate refined by T. We call it the *weakest coarsening* of T, denoted $wc(T)$. We then obtain

$$wc(T)(P) \ \Leftrightarrow \ \forall Q \bullet (P \sqsubseteq_t Q \Rightarrow T(Q))$$

Theorem 6.2.15 *If A is a predicate on untimed traces, then*

$$wc(X \text{ sat } A'(s,\alpha,\aleph))(P) \Leftrightarrow P \text{ sat } A(tr)$$

where $A'(s,\alpha,\aleph) \Leftrightarrow A(tstrip(s))$

Proof If P **sat** $A(tr)$, then whenever $P \sqsubseteq_t Q$ we must have

$$s \in traces(Q) \Rightarrow tstrip(s) \in traces(P) \Rightarrow A(tstrip(s))$$

and so Q **sat** $A'(s,\alpha,\aleph)$.

If $\neg(P \text{ sat } A(tr))$ then $\exists\, tr \in traces(P) \bullet \neg A(tr)$. Let $tr = \langle a_1, a_2, \ldots a_n\rangle$. Then $Q = a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n \rightarrow STOP$ is certainly a refinement of P, and clearly $\neg(Q \text{ sat } A'(s,\alpha,\aleph))$ as required. □

Hence for untimed behavioural specifications S we have $S = wc(sr(S))$. This is not true in general for non-behavioural specifications. For example, consider

$$S(P) \Leftrightarrow \langle a, b\rangle \in traces(P)$$

Then for any *TCSP* process Q we have

$$(\Theta(Q) \sqcap a \rightarrow b \rightarrow STOP) \sqsubseteq_t Q$$

so $sr(S)$ holds of Q. Hence $sr(S) = TRUE$, and so $wc(sr(S)) = TRUE$.

6.3 Strong Timewise Refinement

In this section we consider two closely related forms of strong timewise refinement. We will examine their relationship in some detail. We perform the same analysis that we have just carried out for weak timewise refinement. However, we will find that not all processes preserve strong timewise refinement, so for a given *TCSP* process Q it will not in general be trivial to find an untimed process refined by Q. This makes reduction of proof obligations harder for those processes which do not have $\Theta(Q)$ refined by Q. We will also see that the refinement relation can be extended to a relation between predicates, in the same way as the $\sqsubseteq_t$-relation was extended; we are again able to easily characterise the strongest refinement of a behavioural specification.

We think of a timewise refinement of a process in M_F as the resolution of when events are refusible as well as when they are possible; but it is not so clear what information in TM_{FS} corresponds to a failure in M_F. An untimed refusal set does not correspond to the set of events refused at some time during the performance of the trace, nor even to the set of events refused at some time after the end of the trace. If the process were to stabilise, then we would expect it to correspond to a set of events refusible after stability. However, not all processes are stable, so we must find some other characterisation.

We interpret the failure information (tr, X) in a timed context by saying that there is some time, after a timed version of tr has been performed, after which X may be continuously refused. We would expect, if $P_{\mathfrak{t}}$ were a timewise refinement of P_l in this sense, that

$$(s, [t, \infty) \times X) \in SI(P_{\mathfrak{t}}) \Rightarrow (tstrip(s), X) \in P_l$$

We would not expect tbe converse implication , since the introduction of timing information will in general mean that some traces are uo longer possible, and also that some sets may never be refusible.

If P_l is a process in M_F, and $P_{\mathfrak{t}}$ is a process in TM_{FS}, then we write $P_l \sqsubseteq_f P_{\mathfrak{t}}$ to say that $P_{\mathfrak{t}}$ is a *strong timewise refinement* of P_l. This is defined as follows:

Definition 6.3.1

$$\begin{aligned} P_l \sqsubseteq_f P_{\mathfrak{t}} \;\; \widehat{=} \;\; & \forall (s, \aleph) \in SI(P_{\mathfrak{t}}), X \in \mathbf{P}(\Sigma) \bullet \\ & \exists t \bullet ([t, \infty) \times X \subseteq \aleph) \Rightarrow (tstrip(s), X) \in P_l \end{aligned}$$

We extend the relation $\sqsubseteq_f$ to a relation between environments for M_F and environments for TM_{FS}.

Definition 6.3.2 *If $\sigma : var \rightarrow M_F$ and $\rho : var \rightarrow TM_{FS}$ are environments to their respective models, then*

$$\sigma \sqsubseteq_f \rho \;\; \widehat{=} \;\; \forall X : var \bullet \sigma(X) \sqsubseteq_f \rho(X)$$

Finally, we extend $\sqsubseteq_f$ to a relation between CSP terms and $TCSP$ terms.

Definition 6.3.3 *If P is a CSP term, and Q is a $TCSP$ term, then*

$$P \sqsubseteq_f Q \;\; \widehat{=} \;\; (\sigma \sqsubseteq_f \rho) \Rightarrow \mathcal{F}[\![P]\!]\sigma \sqsubseteq_f \mathcal{E}_T[\![Q]\!]\rho$$

We introduce one more form of timewise refinement, which will be useful in resolving when $P \sqsubseteq_f Q$. It is a natural form of timewise refinement between M_F and TM_{FS}^*, although we must bear in mind that TM_{FS}^* does not deal satisfactorily with unstable processes, because of its treatment of failure sets on unstable traces (if a trace has stability value ∞ then every set is refusible after that trace).

Definition 6.3.4 *If $P_1 \in M_F$ and $P_2 \in TM_{FS}^*$, then*

$$P_1 \sqsubseteq_{ts} P_2 \;\; \widehat{=} \;\; \forall (s, \alpha, X) \in P_2 \bullet (thstrip(s), X) \in P_1$$

If $\sigma : var \to M_F$ and $\rho : var \to TM_{FS}^*$ then

$$\sigma \sqsubseteq_{ts} \rho \;\; \widehat{=} \;\; \forall X : var \bullet \sigma(X) \sqsubseteq_{ts} \rho(X)$$

If P is a CSP term, and Q is a TCSP term, then

$$P \sqsubseteq_{ts} Q \;\; \widehat{=} \;\; \sigma \sqsubseteq_{ts} \rho \Rightarrow \mathcal{F}[\![P]\!]\sigma \sqsubseteq_{ts} \mathcal{E}_T^*[\![Q]\!]\rho$$

We may now define what it means for a term to preserve strong timewise refinement:

Definition 6.3.5 *If Q is a TCSP term, then*

- *Q preserves $\sqsubseteq_f$-refinement if $\Theta(Q) \sqsubseteq_f Q$*
- *Q preserves $\sqsubseteq_{ts}$-refinement if $\Theta(Q) \sqsubseteq_{ts} Q$*

Theorem 6.3.6 *Each of the following predicates on M_F is continuous and satisfiable, for any $Q_1 \in TM_{FS}$, $Q_2 \in TM_{FS}^*$:*

$$\begin{aligned} S_1(X) \;\; &\widehat{=} \;\; X \sqsubseteq_f Q_1 \\ S_2(X) \;\; &\widehat{=} \;\; X \sqsubseteq_{ts} Q_2 \end{aligned}$$

Proof Let $\{P_i\}$ be a convergent sequence of processes in M_F whose limit is P. Assume that $S_1(P_i)$ for each i. Let s and X be such that

$$\exists \aleph, t \bullet (s, \aleph) \in SI(Q_1) \wedge ([t, \infty) \times X \subseteq \aleph)$$

Then by the definition of $\sqsubseteq_f$ we have that $(tstrip(s), X) \in P_i$ for all i, and so $(tstrip(s), X) \in P$. Hence $P \sqsubseteq_f Q_1$, and so S_1 is continuous. Further the process $CHAOS$, defined by

$$CHAOS \;\; \widehat{=} \;\; \mu X \bullet (a : \Sigma \to X) \sqcap \perp$$

satisfies S_1, since it exhibits every possible (*trace, refusal*) pair.

The proof for the continuity and satisfiability of S_2 is entirely similar. □

Conversely, neither $S_1(X) \mathrel{\widehat{=}} P \sqsubseteq_{ts} X$ nor $S_2(X) \mathrel{\widehat{=}} P \sqsubseteq_{ts} X$ are continuous for general M_F processes P, although they may be continuous for some P (e.g. $CHAOS$). Consider

$$\begin{array}{rcl} P_n & \widehat{=} & WAIT\ n\ ;\ a \rightarrow STOP \\ a \rightarrow STOP & \sqsubseteq_f & P_n \\ a \rightarrow STOP & \not\sqsubseteq_f & \lim_{n\rightarrow\infty} P_n = \bot \\ a \rightarrow STOP & \sqsubseteq_{ts} & P_n \\ a \rightarrow STOP & \not\sqsubseteq_{ts} & \lim_{n\rightarrow\infty} P_n = \bot \end{array}$$

Both $a \rightarrow STOP \sqsubseteq_f X$ and $a \rightarrow STOP \sqsubseteq_{ts} X$ hold of each P_n, but neither holds of $\lim_{n\rightarrow\infty} P_n$.

However, the predicate

$$S_2(X) \mathrel{\widehat{=}} P \sqsubseteq_{ts} X$$

is continuous for t-stable and t-prompt processes, since for those processes we can deduce information about infinite behaviours in a bounded time:

Theorem 6.3.7 *Each of the following predicates is continuous for any $P \in M_F$, $t > 0$:*

$$\begin{array}{rcl} S_1(X) & \widehat{=} & P \sqsubseteq_f X \wedge X \textit{ is } t\textit{-prompt} \\ S_2(X) & \widehat{=} & P \sqsubseteq_f X \wedge X \textit{ is } t\textit{-stable} \end{array}$$

Proof We prove S_1 is continuous. The proof for S_2 is entirely similar. Assume $\{Q_i\} \rightarrow Q$, with $S_1(Q_i)$ for each i. Now consider $(s, \aleph) \in \mathcal{SI}_T[\![Q]\!]$ with $[t', \infty) \times X \subseteq \aleph$. Then $[t', t' + t) \times X \subseteq \aleph$, so

$$(s, [t', t' + t) \times X) \in fail(\mathcal{E}_T[\![Q]\!]$$

and so for some n we bave

$$(s, [t', t' + t) \times X) \in fail(\mathcal{E}_T[\![Q_n]\!]$$

Now Q_n is t-prompt, so we deduce

$$(s, [t', \infty) \times X) \in fail(\mathcal{E}_T[\![Q_n]\!]$$

and since $P \sqsubseteq_f Q_n$ we have $(tstrip(s), X) \in \mathcal{F}[\![P]\!]$, as required. □

Lemma 6.3.8 *For any CSP term P, TCSP term Q and $t \geqslant 0$ we have*

$$\begin{array}{rcl} P \sqsubseteq_{ts} Q & \Rightarrow & P \sqsubseteq_{ts} WAIT\ t\ ;\ Q \\ P \sqsubseteq_f Q & \Rightarrow & P \sqsubseteq_f WAIT\ t\ ;\ Q \end{array}$$

Proof Assume $P \sqsubseteq_{ts} Q$, and that $\sigma \sqsubseteq_{ts} \rho$. Then

$$\begin{array}{rcl} (s, \alpha, X) \in \mathcal{E}_T^*[\![WAIT\, t\, ; Q]\!]\rho & \Rightarrow & (s - t, \alpha - t, X) \in \mathcal{E}_T^*[\![Q]\!]\rho \\ & \Rightarrow & (thstrip(s - t), X) \in \mathcal{F}[\![P]\!]\sigma \\ & \Rightarrow & (thstrip(s), X) \in \mathcal{F}[\![P]\!]\sigma \end{array}$$

so $P \sqsubseteq_{ts} WAIT\, t\, ; Q$. □

Assume $P \sqsubseteq_f Q$, and that $\sigma \sqsubseteq_f \rho$. Then

$$\begin{array}{rl} & (s, \aleph) \in \mathcal{SI}_T[\![WAIT\, t\, ; Q]\!]\rho \wedge [t', \infty) \times X \subseteq \aleph \\ \Rightarrow & (s - t, \aleph \dot{-} t) \in \mathcal{SI}_T[\![Q]\!]\rho \wedge [t' \dot{-} t, \infty) \times X \subseteq \aleph \dot{-} t \\ \Rightarrow & (tstrip(s - t), X) \in \mathcal{F}[\![P]\!]\sigma \\ \Rightarrow & (tstrip(s), X) \in \mathcal{F}[\![P]\!]\sigma \end{array}$$

so $P \sqsubseteq_f WAIT\, t\, ; Q$. □

Lemma 6.3.9 *Let P be a CSP term that represents a contraction mapping, and let Q be a TCSP term.*

- *If $P \sqsubseteq_f Q$ then $\mu X \bullet P \sqsubseteq_f \mu X \bullet Q$*
- *If $P \sqsubseteq_{ts} Q$ then $\mu X \bullet P \sqsubseteq_{ts} \mu X \bullet Q$*

Proof $\mathcal{F}[\![\mu X \bullet P]\!]$ is the unique fixed point of the contraction mapping $C \equiv \lambda Y \bullet \mathcal{F}[\![P]\!](\sigma[Y/X])$ on M_F. Now let $\sigma \sqsubseteq_f \rho$. Assume $P_I \sqsubseteq_f \mathcal{E}_T[\![\mu X \bullet Q]\!]\rho$. Then from lemma 6.3.8 we have $P_I \sqsubseteq_f \mathcal{E}_T[\![WAIT\, \delta\, ; \mu X \bullet Q]\!]\rho$, so

$$\sigma[P_I/X] \sqsubseteq_f \rho[\mathcal{E}_T[\![WAIT\, \delta\, ; \mu X \bullet Q]\!]\rho/X]$$

so we have

$$\mathcal{F}[\![P]\!](\sigma[P_I/X]) \sqsubseteq_f \mathcal{E}_T[\![Q]\!](\rho[\mathcal{E}_T[\![WAIT\, \delta\, ; \mu X \bullet Q]\!]\rho/X])$$

and so

$$C(P_I) \sqsubseteq_f \mathcal{E}_T[\![\mu X \bullet Q]\!]\rho$$

Hence C preserves $S(X) \equiv X \sqsubseteq_f \mathcal{E}_T[\![\mu X \bullet Q]\!]\rho$, and we have from theorem 6.3.6 that S is continuous and satisfiable, so it follows (from the fact that C is a contraction mapping) that $S(\mathit{fix}C)$, as required.

The proof for $\sqsubseteq_{ts}$ is entirely similar. □

We will prove that the refinement relation $\sqsubseteq_f$ holds whenever $\sqsubseteq_{ts}$ holds. However, the converse is not the case. Consider

$$a \rightarrow STOP \quad \sqsubseteq_f \quad \mu X \bullet ((a \rightarrow STOP) \overset{1}{\triangleright} X)$$

But $\mathcal{E}^*_T[\![\mu X \bullet ((a \to STOP) \stackrel{1}{\triangleright} X)]\!]$ has stability value ∞ associated with the empty trace. Hence axiom 13 for TM^*_{FS} tells us that $(\langle\rangle, \{a\})$ is a possible failure of the process. But it is not a possible failure of $\mathcal{F}[\![a \to STOP]\!]$, so

$$a \to STOP \quad \not\sqsubseteq_{ts} \quad \mu X \bullet ((a \to STOP) \stackrel{1}{\triangleright} X)$$

In order to prove that $P \sqsubseteq_{ts} Q \Rightarrow P \sqsubseteq_f Q$, we must first obtain some subsidiary results.

Definition 6.3.10 *A trace s in $T\widehat{\Sigma}^*_{\leqslant}$ is* reflected in $\aleph$ *if*

$$\langle(t, \hat{a})\rangle \textbf{ in } s \wedge t > 0 \Rightarrow \exists\, t' < t \bullet [t', t) \times \{a\} \subseteq \aleph$$

When timed refusals are present, we dednce that an event happens at the instant it becomes available from the observation on $\aleph$ that it was refused for some interval up to that time. Hence a trace s is reflected in $\aleph$ if there is evidence in $\aleph$ that the hatted events in s occurred at the instant they became available. We write $R(s, \aleph)$ to abbreviate 's is reflected in $\aleph$'. The following simple consequences of this definition will be used in the next theorem.

Corollary 6.3.11 *The following hold of the relation R:*

- $R(s, \aleph) \Rightarrow R(s \dot{-} t, \aleph \dot{-} t)$
- *If $R(s, \aleph)$ and $s \in Tmerge(u, v)$ then $R(u, \aleph)$ and $R(v, \aleph)$.*
- *If $R(s, \aleph)$ and $\aleph = \aleph_1 \cup \aleph_2$ then $\exists\, s_1, s_2 \bullet s = s_1 \vee s_2 \wedge R(s_1, \aleph_1) \wedge R(s_2, \aleph_2)$*

We will now define a relation between semantic sets P_1 in TM_{FS} and P_2 in TM^*_{FS}. Tbis relation captures the sitnation where all the possible behaviours described by P_1 are also contained (in a different form) in P_2. We may therefore think of P_1 as a more precise description of a system than P_2; tbe projection $\Pi(P_1)$ (see page 185) of P_1 into TM^*_{FS} is more deterministic than P_2[1]. The relation describes the following: if $(\tilde{s}, \alpha, \aleph) \in P_1$, then we deduce that the events refused after stability will be a possible refusal set (after s) in P_2; further, since s is reflected in $\aleph$, the refusal information contained in $\aleph$ is sufficient to deduce which events in s may be hatted; finally, since TM^*_{FS} does not make so many distinctions, the stability value associated with the failure (s, X) will in general be higher than that associated with $(\tilde{s}, \aleph)$ in TM_{FS}.

Wben this relation holds between P_1 and P_2, we say P_2 follows P_1:

[1] P is more deterministic than Q if $Q = P \sqcap Q$ ('more deterministic' is reflexive)

Definition 6.3.12 *If $P_2 \in TM^*_{FS}$ and $P_1 \in TM_{FS}$, then P_2* follows *P_1 if for any $s \in T\hat{\Sigma}^*_{\leqslant}$ we have*

$$(\tilde{s}, \alpha, \aleph) \in P_1 \wedge R(s, \aleph) \Rightarrow \exists \beta \geqslant \alpha \bullet (s, \beta, \sigma(\aleph \upharpoonright \alpha)) \in P_2$$

We abbreviate S_2 follows S_1 to $F(S_2, S_1)$ If $\rho_1 : var \rightarrow TM_{FS}$ and $\rho_2 : var \rightarrow TM^*_{FS}$, then ρ_2 *follows* ρ_1 if for all process variables X we have $F(\rho_2(X), \rho_1(X))$.

If P and Q are $TCSP$ terms, then Q follows P if

$$F(\rho_2, \rho_1) \Rightarrow F(\mathcal{E}^*_T[\![Q]\!]\rho_2, \mathcal{E}_T[\![P]\!]\rho_1)$$

The next theorem tells us that the semantics of P in TM_{FS} is always more precise than its semantics in TM^*_{FS}. This result has interesting consequences. The projection mapping $\Pi : TM_{FS} \rightarrow TM^*_{FS}$ (see page 185) maps a process P to the most deterministic process in TM^*_{FS} that follows P. Thus we will obtain the following corollary:

Corollary 6.3.13 *Let $\Pi : TM_{FS} \rightarrow TM^*_{FS}$ be the projection mapping given in [Ree88]. Then $\Pi(\mathcal{E}_T[\![P]\!])$ follows $\mathcal{E}_T[\![P]\!]$, and is more deterministic than $\mathcal{E}^*_T[\![P]\!]$*

The consequence of the next theorem that is presently most useful is that it will allow us to conclude that $\sqsubseteq_{ts}$-refinement is stronger than $\sqsubseteq_f$-refinement, which will aid us in establishing the presence of the $\sqsubseteq_f$-refinement relation.

Theorem 6.3.14 *If P is a $TCSP$ term, then P follows P*

Proof This theorem is proved by structural induction on P. The difficult cases are indexed nondeterminism, parallel combination, hiding, sequential composition, and recursion. We present the proofs for each of these.

We proceed by structural induction, assuming the result holds for the syntactic subcomponents of the composite processes, and that in each case we have $F(\rho_2, \rho_1)$. We will continue to use the notation $R(s, \aleph)$ to mean that s is reflected in $\aleph$.

Case $\bigsqcap_i P_i$

Consider $(\tilde{s}, \alpha, \aleph) \in \mathcal{E}_T[\![\bigsqcap_i P_i]\!]$. Then define

$$\begin{aligned} S &= \{\gamma \mid \exists\, i \bullet (\tilde{s}, \gamma, \aleph) \in \mathcal{E}_T[\![P_i]\!]\rho_1\} \\ T &= \{\beta \mid \exists\, i \bullet (s, \beta, \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}^*_T[\![P_i]\!]\rho_2\} \end{aligned}$$

Then $\alpha = sup(S)$. Now if $T \neq \emptyset$ then $(s, sup(T), \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}_T^*[\![\bigsqcap_i P_i]\!]\rho_2$. So to prove the required result, we need only show that $\gamma \in S \Rightarrow \exists\, \beta \geqslant \gamma \bullet \beta \in T$

$$\begin{array}{rl} & \gamma \in S \\ \Rightarrow & \exists\, i \bullet (\tilde{s}, \gamma, \aleph) \in \mathcal{E}_T[\![P_i]\!]\rho_1 \\ \Rightarrow & \exists\, i, \beta \geqslant \gamma \bullet (s, \beta, \sigma(\aleph \upharpoonright \gamma)) \in \mathcal{E}_T^*[\![P_i]\!]\rho_2 \\ \Rightarrow & \exists\, i, \beta \geqslant \gamma \bullet (s, \beta, \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}_T^*[\![P_i]\!]\rho_2 \\ \Rightarrow & \exists\, \beta \geqslant \gamma \bullet \beta \in T \end{array}$$

as required.

Case $P \parallel Q$

Now consider $(\tilde{s}, \alpha, \aleph) \in \mathcal{E}_T[\![P \parallel Q]\!]\rho_1$, where $R(s, \aleph)$. Define:

$$\begin{array}{rcl} S & = & \{max\{\alpha_1, \alpha_2\} \mid \exists\, \aleph_1, \aleph_2 \bullet \aleph = \aleph_1 \cup \aleph_2 \wedge \\ & & \qquad (\tilde{s}, \alpha_1, \aleph_1) \in \mathcal{E}_T[\![P]\!]\rho_1 \wedge (\tilde{s}, \alpha_2, \aleph_2) \in \mathcal{E}_T[\![Q]\!]\rho_1\} \end{array}$$

$$T \;=\; \{max\{\beta_1, \beta_2\} \mid \exists\, s_1, s_2, X_1, X_2 \bullet s = s_1 \vee s_2 \wedge X_1 \cup X_2 = \sigma(\aleph \upharpoonright \alpha)\}$$

Then $\alpha = sup(S)$. Now given $\aleph_1$ and $\aleph_2$ such that $\aleph = \aleph_1 \cup \aleph_2$ we have from corollary 6.3.11 that there are s_1 and s_2 such that $s = s_1 \vee s_2$ and $R(s_1, \aleph_1)$ and $R(s_2, \aleph_2)$, so

$$\begin{array}{rcl} S & = & \{max\{\alpha_1, \alpha_2\} \mid \exists\, \aleph_1, \aleph_2, s_1, s_2 \bullet \aleph = \aleph_1 \cup \aleph_2 \wedge s = s_1 \vee s_2 \wedge \\ & & \qquad (\tilde{s_1}, \alpha_1, \aleph_1) \in \mathcal{E}_T[\![P]\!]\rho_1 \wedge (\tilde{s_2}, \alpha_2, \aleph_2) \in \mathcal{E}_T[\![Q]\!]\rho_1 \\ & & \qquad R(s_1, \aleph_1) \wedge R(s_2, \aleph_2)\} \end{array}$$

Consider $\gamma \in S$. Then $\gamma = max\{\alpha_1, \alpha_2\}$, where

$$(\tilde{s_1}, \alpha_1, \aleph_1) \in \mathcal{E}_T[\![P]\!]\rho_1 \wedge R(s_1, \aleph) \wedge (\tilde{s_2}, \alpha_2, \aleph_2) \in \mathcal{E}_T[\![Q]\!]\rho_1 \wedge R(s_2, \aleph)$$

Hence

$$\exists\, \beta_1 \geqslant \alpha_1, \beta_2 \geqslant \alpha_2 \bullet (s_1, \beta_1, \sigma(\aleph_1 \upharpoonright \alpha_1)) \in \mathcal{E}_T^*[\![P]\!]\rho_2 \wedge (s_2, \beta_2, \sigma(\aleph_2 \upharpoonright \alpha_2)) \in \mathcal{E}_T^*[\![Q]\!]\rho_2$$

But $\alpha_1 \leqslant \alpha$ and $\alpha_2 \leqslant \alpha$, so

$$\sigma(\aleph_1 \upharpoonright \alpha) \subseteq \sigma(\aleph_1 \upharpoonright \alpha_1) \;\wedge\; \sigma(\aleph_2 \upharpoonright \alpha) \subseteq \sigma(\aleph_2 \upharpoonright \alpha_2)$$

so

$$(s_1, \beta_1, \sigma(\aleph_1 \upharpoonright \alpha)) \in \mathcal{E}_T^*[\![P]\!]\rho_2 \;\wedge\; (s_2, \beta_2, \sigma(\aleph_2 \upharpoonright \alpha)) \in \mathcal{E}_T^*[\![Q]\!]\rho_2$$

Therefore, $max\{\beta_1, \beta_2\} \in T$, so $\exists\, \gamma' \geqslant \gamma \bullet \gamma' \in T$. Hence

$$(s, sup(T), \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}_T^*[\![P \parallel Q]\!]\rho_2$$

and $sup(T) \geqslant sup(S)$. □

The cases for the alphabet parallel operator and the interleaving operator are very similar, and again use the properties presented in corollary 6.3.11

Case $P \,\text{;}\, Q$

Now consider $(\tilde{s}, \alpha, \aleph) \in \mathcal{E}_T[\![P \,\text{;}\, Q]\!]\rho_1$, where $R(s, \aleph)$. Define:

$$\begin{aligned} S \;=\;& \{\alpha \mid \forall I \bullet (\tilde{s}, \alpha, \aleph \cup (I \times \{\checkmark\})) \in \mathcal{E}_T[\![P]\!]\rho_1 \wedge \checkmark \notin \sigma(\tilde{s})\} \\ & \cup \{a + t + \delta \mid \checkmark \notin \sigma(\tilde{s} \upharpoonright t) \wedge (\tilde{s} \dot{-} (t+\delta), \alpha, \aleph \dot{-} (t+\delta)) \in \mathcal{E}_T[\![Q]\!]\rho_1 \\ & \qquad \wedge ((\tilde{s} \upharpoonright t)^\frown\langle(t, \checkmark)\rangle, \aleph \upharpoonright t \cup [0, t) \times \{\checkmark\}) \in fail(\mathcal{E}_T[\![P]\!]\rho_1)\} \end{aligned}$$

$$\begin{aligned} T \;=\;& \{\alpha \mid (s, \alpha, X \cup \{\checkmark\}) \in \mathcal{E}^*_T[\![P]\!]\rho_2 \wedge \checkmark \notin \sigma(s)\} \\ & \cup \{a + (t+\delta) \mid (s \upharpoonright t)^\frown\langle(t, \hat{\checkmark})\rangle \in traces(\mathcal{E}^*_T[\![P]\!]\rho_2) \wedge \checkmark \notin \sigma(s \upharpoonright t) \\ & (s \dot{-} (t+\delta), \alpha, \sigma(\aleph \upharpoonleft \alpha)) \in \mathcal{E}^*_T[\![Q]\!]\rho_2\} \end{aligned}$$

Then $\alpha = sup(S)$, and if T is not empty then $(s, sup(T), \sigma(\aleph \upharpoonleft \alpha)) \in \mathcal{E}^*_T[\![Q]\!]\rho_2$. It is sufficient to prove that $\gamma \in S \Rightarrow \exists\, \gamma' \geqslant \gamma \bullet \gamma' \in T$. So consider $\gamma \in S$. Then either:

$$\begin{aligned} & (\tilde{s}, \gamma, \aleph \cup [0, \gamma + 1) \times \{\checkmark\}) \in \mathcal{E}_T[\![P]\!]\rho_1 \wedge \checkmark \notin \sigma(\tilde{s}) \wedge R(s, \aleph \cup [0, \gamma + 1) \times \{\checkmark\} \\ & \;\Rightarrow\; \exists\, \gamma' \geqslant \gamma \bullet (s, \gamma', \sigma(\aleph \upharpoonleft \gamma) \cup \{\checkmark\}) \in \mathcal{E}^*_T[\![P]\!]\rho_2 \\ & \;\Rightarrow\; \exists\, \gamma' \geqslant \gamma \bullet (s, \gamma', \sigma(\aleph \upharpoonleft \alpha) \cup \{\checkmark\}) \in \mathcal{E}^*_T[\![P]\!]\rho_2 \\ & \;\Rightarrow\; \exists\, \gamma' \geqslant \gamma \bullet \gamma' \in T \end{aligned}$$

or:

$$\begin{aligned} & \gamma = \beta + (t+\delta) \wedge \checkmark \notin \sigma(\tilde{s} \upharpoonright t) \\ & \wedge ((\tilde{s} \upharpoonright t)^\frown\langle(t, \checkmark)\rangle, \aleph \upharpoonright t \cup [0, t) \times \{\checkmark\}) \in fail(\mathcal{E}_T[\![P]\!]\rho_1) \\ & \wedge (\tilde{s} \dot{-} (t+\delta), \beta, \aleph \dot{-} (t+\delta)) \in \mathcal{E}_T[\![Q]\!]\rho_1 \\ & \wedge R((\tilde{s} \upharpoonright t)^\frown\langle(t, \hat{\checkmark})\rangle, \aleph \upharpoonright t \cup [0, t) \times \{\checkmark\})) \\ \Rightarrow\;& \langle s \upharpoonright t^\frown\langle(t, \hat{\checkmark})\rangle \in traces(\mathcal{E}^*_T[\![P]\!]\rho_2) \\ & \wedge \exists\, \beta' \geqslant \beta \bullet (s \dot{-} (t+\delta), \beta', \sigma((\aleph \dot{-} (t+\delta)) \upharpoonleft \beta)) \in \mathcal{E}^*_T[\![Q]\!]\rho_2 \\ \Rightarrow\;& \langle s \upharpoonright t^\frown\langle(t, \hat{\checkmark})\rangle \in traces(\mathcal{E}^*_T[\![P]\!]\rho_2) \\ & \wedge (s \dot{-} (t+\delta), \beta', \sigma((\aleph) \upharpoonleft \alpha)) \in \mathcal{E}^*_T[\![Q]\!]\rho_2 \\ \Rightarrow\;& \beta' + (t+\delta) \in T \wedge \beta' + (t+\delta) \geqslant \gamma \\ \Rightarrow\;& \exists\, \gamma' \geqslant \gamma \bullet \gamma' \in T \end{aligned}$$

□

Case $P \setminus A$

To prove this case, we first require a technical sublemma:

Sublemma 6.3.15 *If* $(s,\alpha,\aleph) \in \mathcal{E}_T\llbracket P \setminus A\rrbracket\rho \wedge \alpha < \infty$ *then*

$$\forall \nu > \alpha \bullet \alpha = sup\{\beta \mid \exists\, w \bullet s = w \setminus A \wedge (w,\beta,\aleph \cup [\mathit{0},\nu) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho\}$$

Proof Consider $(s,\alpha,\aleph) \in \mathcal{E}_T\llbracket P \setminus A\rrbracket\rho$, and $\nu > \alpha$. Define

$$\begin{array}{rcl} S &=& \{\beta \mid \exists\, w \bullet s = w \setminus A \wedge \exists\, \alpha \geqslant \beta \geqslant end(w) \bullet (w,\alpha,\aleph \cup [\mathit{0},\beta) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho\} \\ S_\nu &=& \{\beta \mid \exists\, w \bullet s = w \setminus A \wedge (w,\beta,\aleph \cup [\mathit{0},\nu) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho\} \end{array}$$

Then from the definition of hiding, $\alpha = sup(S)$. Now lemma A.1.2 yields that $\gamma \in S \Rightarrow \exists\,\gamma' \geqslant \gamma \bullet \gamma' \in S_\nu$, so we obtain $sup(S_\nu) \geqslant sup(S)$

We wish to prove $sup(S_\nu) = sup(S)$, so we assume $sup(S_\nu) > sup(S)$ for a contradiction:

$$\begin{array}{rl} & sup(S_\nu) > sup(S) \\ \Rightarrow & \exists\,\beta \in S_\nu \bullet \beta > \alpha \\ \Rightarrow & \exists\, w \bullet s = w \setminus A \wedge (w,\beta,\aleph \cup [\mathit{0},\nu) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho \\ \Rightarrow & \exists\,\beta' \geqslant \beta, w \bullet s = w \setminus A \wedge (w,\beta',\aleph \cup [\mathit{0}, min\{\nu,\beta\}) \times A) \in \mathcal{E}_T\llbracket P\rrbracket \\ \Rightarrow & min\{\nu,\beta\} \in S \\ \Rightarrow & sup(S) \geqslant min\{\nu,\beta\} > \alpha \end{array}$$

which yields a contradiction. □

We can now handle the hiding case:

Let $(\bar{s},\alpha,\aleph) \in \mathcal{E}_T\llbracket P \setminus A\rrbracket\rho_I \wedge R(s,\aleph)$. Define:

$$\begin{array}{rcl} S &=& \{\beta \mid \exists\, w \bullet \tilde{s} = w \setminus A \wedge \exists\,\alpha' \geqslant \beta \geqslant end(w) \bullet \\ && \qquad (w,\alpha',\aleph \cup [\mathit{0}, max\{end(\aleph),\beta\}) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho_I\} \\ S_\gamma &=& \{\beta \mid \exists\, w \bullet s = w \setminus A \wedge w \text{ is A-active} \wedge (\tilde{w},\beta,\aleph \cup [\mathit{0},\gamma) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho_I\} \\ T &=& \{\beta \mid \exists (w,\beta) \in stab(\mathcal{E}^*_T\llbracket P\rrbracket\rho_E) \bullet w \setminus A = s \wedge w \text{ is A-active}\} \end{array}$$

Case $\alpha = \infty$

The result follows immediately, since $\aleph \upharpoonright \infty = \emptyset$

Case $\alpha < \infty$

For any γ, if $\beta \in S_\gamma$ then $\exists\,\beta' \in T \bullet \beta' \geqslant \beta$. Hence if $\forall\gamma \bullet sup(S_\gamma) = \infty$ then $sup(T) = \infty$, and $(s,\infty,\sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}^*_T\llbracket P\rrbracket\rho_E$.

If $\exists\,\gamma \bullet sup(S_\gamma) = \alpha' < \infty$, then set $\nu = max\{\gamma,\alpha',\alpha\} + \mathit{1}$. Then from sublemma 6.3.15 we obtain $sup(S_\nu) = \alpha$, and so $sup(T) \geqslant \alpha$. Also,

$$\begin{array}{rl} & \beta \in S_\nu \\ \Rightarrow & \exists\, w \bullet s = w \setminus A \wedge w \text{ is A-active} \wedge (\tilde{w},\beta,\aleph \cup [\mathit{0},\nu) \times A) \in \mathcal{E}_T\llbracket P\rrbracket\rho_I\} \\ \Rightarrow & \exists\, w \bullet s = w \setminus A \wedge w \text{ is A-active} \wedge (w,\sigma(\aleph \upharpoonright \beta) \cup A) \in fail(\mathcal{E}^*_T\llbracket P\rrbracket\rho_E) \\ \Rightarrow & (s,\sigma(\aleph \upharpoonright \alpha)) \in fail(\mathcal{E}^*_T\llbracket P \setminus A\rrbracket\rho_E) \end{array}$$

□

Case $\mu X \bullet P$

Define $G(Y) = P[(WAIT\,\delta\,;\,Y)/X]$. We will first prove that $G^n(STOP)$ follows $G^n(STOP)$ for all n.

Base Case: $STOP$ follows $STOP$ is immediate.

Case $n + 1$:

Assume $F(G^n(STOP), G^n(STOP))$ and $F(\rho_2, \rho_1)$.

$$\begin{array}{rcl}
\mathcal{E}^*_T[\![G^{n+1}(STOP)]\!]\rho_2 & = & \mathcal{E}^*_T[\![G(G^n(STOP))]\!]\rho_2 \\
 & = & \mathcal{E}^*_T[\![P]\!](\rho_2[\mathcal{E}^*_T[\![WAIT\,\delta\,;\,G^n(STOP)]\!]\rho_2/X]) \\
\text{which follows} & & \mathcal{E}_T[\![P]\!](\rho_1[\mathcal{E}_T[\![WAIT\,\delta\,;\,G^n(STOP)]\!]\rho_1/X]) \\
 & = & \mathcal{E}_T[\![G^{n+1}(STOP)]\!]\rho_1
\end{array}$$

Hence we have that $\forall\, n \bullet F(G^n(STOP), G^n(STOP))$.

Now $\{G^n(STOP)\}$ has limit $\mu X \bullet P$ in both TM_{FS} and TM^*_{FS}. Consider $(\tilde{s}, \alpha, \aleph) \in \mathcal{E}_T[\![\mu X \bullet P]\!]$.

Case $\alpha < \infty$

Then $\exists\, N \bullet N\delta > max\{end(s, \aleph), \alpha\}$. So then

$$\forall\, n > N \bullet (\tilde{s}, \alpha, \aleph) \in \mathcal{E}_T[\![G^n(STOP)]\!]$$

But $\forall n \bullet F(G^n(STOP), G^n(STOP))$, so

$$\forall\, n > N, \exists\, \beta \geqslant \alpha \bullet (s, \beta, \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}^*_T[\![F^n(STOP)]\!]$$

Hence we obtain

$$\exists\, \beta \geqslant \alpha \bullet (s, \beta, \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}^*_T[\![\mu X \bullet F(X)]\!]$$

Case $\alpha = \infty$

$$\begin{array}{rl}
\Rightarrow & sup\{\alpha_n \mid (\tilde{s}, \alpha_n) \in stab(\mathcal{E}_T[\![F^n(STOP)]\!])\} = \infty \\
\Rightarrow & sup\{\beta_n \mid (s, \beta_n) \in stab(\mathcal{E}^*_T[\![F^n(STOP)]\!])\} = \infty \\
\Rightarrow & (s, \infty) \in stab(\mathcal{E}^*_T[\![\mu X \bullet F(X)]\!]) \\
\Rightarrow & (s, \infty, \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{E}^*_T[\![\mu X \bullet F(X)]\!]
\end{array}$$

□

Corollary 6.3.16 $(s, \infty, \aleph) \in \mathcal{E}_T\llbracket P \rrbracket \Rightarrow (s, \infty) \in stab(\mathcal{E}_T^*\llbracket P \rrbracket)$

We are now in a position to prove

Theorem 6.3.17 *For any CSP process P and TCSP process Q we have*

$$P \sqsubseteq_{ts} Q \Rightarrow P \sqsubseteq_{f} Q$$

Proof Assume $P \sqsubseteq_{ts} Q$. Consider $(s, \aleph) \in \mathcal{SI}_T\llbracket Q \rrbracket$, with $[t, \infty) \times X \subseteq \aleph$ for some t. Then $(s, \alpha, \emptyset) \in \mathcal{E}_T\llbracket Q \rrbracket$ for some α.

Case $\alpha = \infty$

Then

$$\begin{aligned} & (s, \infty) \in stab(\mathcal{E}_T^*\llbracket Q \rrbracket) \\ \Rightarrow\ & (s, X) \in fail(\mathcal{E}_T^*\llbracket Q \rrbracket) \\ \Rightarrow\ & (tstrip(s), X) \in \mathcal{F}\llbracket P \rrbracket \end{aligned}$$

Case $\alpha < \infty$

Then from axiom 10 for TM_{FS} we have

$$\begin{aligned} & \exists\, \beta \leqslant \alpha \bullet (s, \beta, [t, max\{t, \alpha\} + 1) \times X) \in \mathcal{E}_T\llbracket Q \rrbracket \\ \Rightarrow\ & (s, X) \in fail(\mathcal{E}_T^*\llbracket Q \rrbracket) && \text{since } F(Q, Q) \\ \Rightarrow\ & (tstrip(s), X) \in \mathcal{F}\llbracket P \rrbracket \end{aligned}$$

□

Theorem 6.3.18 *All the process constructors except hiding, infinite nondeterminism, and infinite-to-one alphabet renaming preserve* $\sqsubseteq_{ts}$ *refinement.*

Proof By a straightforward case analysis. □

We identify sufficient conditions on timed processes for the preservation of $\sqsubseteq_{ts}$-refinement:

Definition 6.3.19 *A set of processes* $\{Q_i\}$ *has bounded stability if*

$$sup\{\alpha \mid \exists\, i \bullet (s, \alpha) \in stab(\mathcal{E}_T^*\llbracket Q_i \rrbracket)\} = \infty \Rightarrow \exists\, i \bullet (s, \infty) \in stab(\mathcal{E}_T^*\llbracket Q_i \rrbracket)$$

Lemma 6.3.20 *If* $\{Q_i\}$ *has bounded stability and* $\{P_i\}$ *is such that* $\forall\, i \bullet P_i \sqsubseteq_{ts} Q_i$ *then* $\bigsqcap_i P_i \sqsubseteq_{ts} \bigsqcap_i Q_i$

Proof Consider $(s, \alpha, X) \in \mathcal{E}_T^* [\![\bigsqcap_i Q_i]\!]$. We have $\alpha = sup(S)$, where

$$S = \{ \beta \mid \exists\, i \bullet (s, \beta, X) \in \mathcal{E}_T^* [\![Q_i]\!] \}$$

Case $\alpha = \infty$

Then by bounded stability we have

$$\exists\, i \bullet (s, \infty) \in stab(\mathcal{E}_T^* [\![Q_i]\!])$$

so

$$(s, \infty, X) \in \mathcal{E}_T^* [\![Q_i]\!])$$

Hence $(tstrip(s), X) \in \mathcal{F} [\![P_i]\!]$, and so $(tstrip(s), X) \in \mathcal{F} [\![\bigsqcap_i P_i]\!]$.

Case $\alpha < \infty$

Then

$$\begin{array}{ll} & \exists\, i \bullet (s, \beta, X) \in \mathcal{E}_T^* [\![Q_i]\!] \\ \Rightarrow & (tstrip(s), X) \in \mathcal{F} [\![P_i]\!] \\ \Rightarrow & (tstrip(s), X) \in \mathcal{F} [\![\bigsqcap_i P_i]\!] \end{array}$$

□

Definition 6.3.21 *A process P is bounded under A if*

$$\begin{array}{lll} \forall s \in traces(\mathcal{E}_T^* [\![P]\!]) & \bullet & sup\{ \beta \mid \exists (w, \beta) \in stab(\mathcal{E}_T^* [\![P]\!]) \bullet w \setminus A = s \} = \infty \\ & & \Leftrightarrow \exists (w, \infty) \in stab(\mathcal{E}_T^* [\![P]\!]) \bullet w \setminus A = s \end{array}$$

Lemma 6.3.22 *If Q is bounded under A and $P \sqsubseteq_{ts} Q$ then $P \setminus A \sqsubseteq_{ts} Q \setminus A$*

Proof Consider $(s, \alpha, X) \in \mathcal{E}_T^* [\![Q \setminus A]\!]$.

Case $\alpha = \infty$

Then by bounded stability we have

$$\begin{array}{ll} & \exists (w, \infty) \in stab(\mathcal{E}_T^* [\![Q]\!]) \bullet w \text{ is } A\text{-active} \wedge w \setminus A = s \\ \Rightarrow & (w, \infty, X \cup A) \in \mathcal{E}_T^* [\![Q]\!] \\ \Rightarrow & (thstrip(w), X \cup A) \in \mathcal{F} [\![P]\!] \\ \Rightarrow & (thstrip(w) \setminus A, X) \in \mathcal{F} [\![P \setminus A]\!] \\ \Rightarrow & (thstrip(s), X) \in \mathcal{F} [\![P \setminus A]\!] \end{array}$$

Case $\alpha < \infty$

$$\begin{array}{ll} \Rightarrow & \exists\, w \bullet (w, X \cup A) \in fail(\mathcal{E}_T^* [\![Q]\!]) \wedge w \setminus A = s \\ \Rightarrow & (thstrip(w), X \cup A) \in \mathcal{F} [\![P]\!] \\ \Rightarrow & (thstrip(w) \setminus A, X) \in \mathcal{F} [\![P \setminus A]\!] \\ \Rightarrow & (thstrip(s), X) \in \mathcal{F} [\![P \setminus A]\!] \end{array}$$

□

Corollary 6.3.23 *If P is such that*

- *if $Q \setminus A$ is a syntactic component of P then Q is bounded under A*
- *if $\bigsqcap_{i \in I} P_i$ is a syntactic component of P then the set $\{P_i\}$ is boundedly stable*
- *if $f(Q)$ is a syntactic component of P then f is finite to one*

then $\Theta(P) \sqsubseteq_{ts} P$

Proof Follows immediately by structural induction, using theorem 6.3.18 and lemmas 6.3.20 and 6.3.22. □

This yields a sufficient condition for a process to be a strong timewise refinement of its image under Θ.

Corollary 6.3.24 *If P is such that*

- *if $Q \setminus A$ is a syntactic component of P then Q is bounded under A*
- *if $\bigsqcap_{i \in I} P_i$ is a syntactic component of P then the set $\{P_i\}$ is boundedly stable*
- *if $f(Q)$ is a syntactic component of P then f is finite to one*

then $\Theta(P) \sqsubseteq_f P$

We have obtained, via $\sqsubseteq_{ts}$-refinement, a sufficient condition for *TCSP terms* to preserve $\sqsubseteq_f$-refinement. We now examine which *TCSP operators* preserve $\sqsubseteq_f$-refinement. We shall find that it is not preserved by parallel composition, despite the fact tbat *TCSP* terms built with that operator do preserve timewise refinement (under the appropriate conditions for hiding and non-deterministic choice).

Lemma 6.3.25 *Each of the basic TCSP processes STOP, SKIP, WAIT t, $\perp$, and X satisfy $\Theta(Q) \sqsubseteq_f Q$.*

Proof Immediate, since they each satisfy $\Theta(Q) \sqsubseteq_{ts} Q$ □

Lemma 6.3.26 *The following TCSP process constructors preserve $\sqsubseteq_f$-refinement:*

$$a \rightarrow P,\ a : A \rightarrow P(a).\ P \sqcap Q,\ \bigsqcap_i P_i$$
$$P \,\Box\, Q,\ P \,|||\, Q,\ P \,;\, Q,\ f^{-1}(P)$$

Proof Straightforward □

However, the '$\|$' and '${}_{X_1}\|_{X_2}$' operators do not in general preserve $\sqsubseteq_f$-refinement. Consider

$$\begin{array}{rcl} P & \mathrel{\widehat{=}} & a \rightarrow STOP \\ Q_1 & \mathrel{\widehat{=}} & \mu\, X \bullet (a \rightarrow STOP \overset{1}{\triangleright} WAIT\ 3\ ; X) \\ Q_2 & \mathrel{\widehat{=}} & WAIT\ 2\ ; Q_1 \end{array}$$

Then $P \sqsubseteq_f Q_1$ and $P \sqsubseteq_f Q_2$, but

$$P \parallel P \not\sqsubseteq_f Q_1 \parallel Q_2$$

since $P \parallel P$ is unable to refuse a after the empty trace, but $Q_1 \parallel Q_2$ can refuse it. Formally:

$$\begin{array}{rcl} (\langle\rangle, \{a\}) & \notin & \mathcal{F}[\![P \parallel P]\!] \\ (\langle\rangle, [0,\infty) \times \{a\}) & \in & \mathcal{SI}_T[\![Q_1 \parallel Q_2]\!] \end{array}$$

We therefore seek conditions where $\sqsubseteq_f$-refinement is preserved by parallel composition.

Theorem 6.3.27 *If P_1 and P_2 are CSP processes, Q_1 and Q_2 are TCSP processes, $P_1 \sqsubseteq_f Q_1$, $P_2 \sqsubseteq_f Q_2$, both Q_1 and Q_2 are prompt on $X_1 \cap X_2$, Q_1 is non-retracting on Y_1 and Q_2 is non-retracting on Y_2, and $X_1 \cap X_2 \subseteq Y_1 \cup Y_2$ then*

$$P_1 \;{}_{X_1}\|_{X_2}\; P_2 \sqsubseteq_f Q_1 \;{}_{X_1}\|_{X_2}\; Q_2$$

Proof Let Q_1 and Q_2 be t'-prompt. Consider $(s, \aleph) \in \mathcal{SI}_T[\![Q_1 \;{}_{X_1}\|_{X_2}\; Q_2]\!]$, and $[t,\infty) \times A \subseteq \aleph$. Then

$$\begin{array}{l} \exists\, s_1, s_2, \aleph_1, \aleph_2 \bullet s \in s_1 \;{}_{X_1}\|_{X_2}\; s_2 \wedge \aleph = \aleph_1 \cup \aleph_2 \wedge \sigma(\aleph_1) \cap (X_2 \setminus X_1) = \emptyset \wedge \\ \qquad \sigma(\aleph_2) \cap (X_1 \setminus X_2) = \emptyset \wedge (s_1, \aleph_1) \in \mathcal{SI}_T[\![Q_1]\!] \wedge (s_2, \aleph_2) \in \mathcal{SI}_T[\![Q_2]\!] \end{array}$$

We require the following definitions:

$$\begin{array}{rcl} T & = & max\{end(s), t\} + t' \\ T' & = & T + t' \\ B_1 & = & A \cap Y_1 \cap (\sigma(\aleph_1 \uparrow [T, T'))) \\ B_2 & = & A \cap Y_2 \cap (\sigma(\aleph_2 \uparrow [T, T'))) \\ C_1 & = & A \cap Y_2 \setminus (\sigma(\aleph_2 \uparrow [T, T'))) \\ C_2 & = & A \cap Y_1 \setminus (\sigma(\aleph_1 \uparrow [T, T'))) \\ D_1 & = & A \cap (X_1 \setminus X_2) \\ D_2 & = & A \cap (X_2 \setminus X_1) \end{array}$$

Now $B_1 \subseteq \sigma(\aleph_1 \uparrow [T, T'))$, so by non-retraction of Q_1 on Y_1 and lemma 5.1.16 we have

$$(s_1, \aleph_1 \cup [end(s), T) \times B_1) \in \mathcal{SI}_T[\![Q_1]\!]$$

so by promptness on $Y_1 \cup Y_2$ and $B_1 \subseteq Y_1$ we have

$$(s_1, \aleph_1 \cup [T, \infty) \times B_1) \in \mathcal{SI}_T[\![Q_1]\!]$$

Now

$$[T, T') \times C_1 \subseteq \aleph_1 \cup [T, \infty) \times B_1$$

so

$$(s_1, \aleph_1 \cup [T, \infty) \times (C_1 \cup B_1)) \in \mathcal{SI}_T[\![Q_1]\!]$$

Clearly

$$[T, \infty) \times D_1 \cap \aleph_2 = \emptyset$$

and so

$$[T, \infty) \times D_1 \subseteq \aleph_1$$

Therefore

$$(s_1, \aleph_1 \cup [T, \infty) \times (D_1 \cup C_1 \cup B_1)) \in \mathcal{SI}_T[\![Q_1]\!]$$

By a symmetric argument, we obtain

$$(s_2, \aleph_2 \cup [T, \infty) \times (D_2 \cup C_2 \cup B_2)) \in \mathcal{SI}_T[\![Q_2]\!]$$

Hence by the definition of $\sqsubseteq_f$-refinement we obtain

$$(tstrip(s_1), D_1 \cup C_1 \cup B_1) \in \mathcal{F}[\![P_1]\!]$$

and

$$(tstrip(s_2), D_2 \cup C_2 \cup B_2) \in \mathcal{F}[\![P_2]\!]$$

Now

$$D_1 \cup C_1 \cup B_1 \cup D_2 \cup C_2 \cup B_2 = A$$

and

$$tstrip(s) \in tstrip(s_1) \;{}_{X_1}\|_{X_2}\; tstrip(s_2)$$

and so

$$(tstrip(s), A) \in \mathcal{F}[\![P_1 \;{}_{X_1}\|_{X_2}\; P_2]\!]$$

as required. □

Corollary 6.3.28 *If $P_1 \sqsubseteq_f Q_1$ and $P_2 \sqsubseteq_f Q_2$ and both Q_1 and Q_2 are prompt, and Q_1 is non-retracting, then*

$$P_1 \;{}_X\|_Y\; P_2 \sqsubseteq_f Q_1 \;{}_X\|_Y\; Q_2$$

Proof If Q_1 is non-retracting then it is non-retracting on $X \cap Y$; Q_1 and Q_2 are both prompt on $X \cap Y$, so the previous theorem applies with $Y_1 = X \cap Y, Y_2 = \emptyset$.

Theorem 6.3.29 *If $P_1 \sqsubseteq_f Q_1$ and $P_2 \sqsubseteq_f Q_2$ and Q_1 and Q_2 are boundedly stable, then $P_1 \;{}_X\|_Y\; P_2 \sqsubseteq_f Q_1 \;{}_X\|_Y\; Q_2$*

Proof Consider

$$(s, \aleph) \in \mathcal{SI}_T[\![Q_1 \;{}_X\|_Y\; Q_2]\!], [t, \infty) \times Z \subseteq \aleph$$

Then $Q_1 \;{}_X\|_Y\; Q_2$ is boundedly stable, so

$$(s, \alpha, [t, max\{t, \alpha\} + 1) \times Z) \in \mathcal{E}_T[\![Q_1 \;{}_X\|_Y\; Q_2]\!]$$

so for some

$$\aleph_1, \aleph_2, \alpha_1 \leqslant \alpha, \alpha_2 \leqslant \alpha$$

we have

$$(s \upharpoonright X, \alpha_1, \aleph_1) \in \mathcal{E}_T[\![P]\!] \wedge \sigma(\alpha_X) \cap (Y \setminus X) = \emptyset$$

and

$$(s \upharpoonright Y, \alpha_2, \aleph_2) \in \mathcal{E}_T[\![Q]\!] \wedge \sigma(\alpha_Y) \cap (X \setminus Y) = \emptyset$$

and

$$\aleph_1 \cup \aleph_2 = [t, max\{t, \alpha\} + 1) \times Z$$

Let

$$Z_1 = \sigma(\aleph_1 \upharpoonleft \alpha_1), Z_2 = \sigma(\aleph_2 \upharpoonleft \alpha_2)$$

Then $Z_1 \cup Z_2 = Z$ Now from theorem 3.6.6 we have that

$$(s \upharpoonright X, [\alpha_1, \infty) \times Z_1) \in \mathcal{SI}_T[\![Q_1]\!]$$

and

$$(s \upharpoonright Y, [\alpha_2, \infty) \times Z_2) \in \mathcal{SI}_T[\![Q_2]\!]$$

Hence

$$(tstrip(s \upharpoonright X), Z_1) \in \mathcal{F}[\![P_1]\!]$$

and

$$(tstrip(s \upharpoonright Y), Z_2) \in \mathcal{F}[\![P_2]\!]$$

so

$$(tstrip(s), Z_1 \cup Z_2) \in \mathcal{F}[\![P_1 \;{}_X\|_Y\; P_2]\!]$$

as required. □

If every semi-infinite behaviours of $P \setminus A$ arises from a single semi-infinite behaviour of P, rather than from a sequence of approximations, then we shall see that the operator $\setminus A$ will preserve $\sqsubseteq_f$-refinement for P. A process with this property is *well behaved* on A:

Definition 6.3.30 *A TCSP process P is well behaved on A if*

$$\begin{array}{l}(s, [t, \infty) \times X) \in \mathcal{SI}_T[\![P \setminus A]\!] \Rightarrow \\ \qquad \exists\, w \bullet \#w \text{ is finite} \wedge (w, [0, \infty) \times A \cup [t, \infty) \times X) \in \mathcal{I}_T[\![P]\!] \wedge w \setminus A = s\end{array}$$

Observe that not all processes are well behaved on A. For example, the process

$$P \mathrel{\hat=} n : \mathbf{N} \rightarrow WAIT\ n\ ; a \rightarrow STOP$$

is not well behaved on $\mathbf{N}$. When the set $\mathbf{N}$ is hidden, the choice of n made by P is not visible, so a can be refused for any finite length of time: $\forall\, n \bullet (\langle\rangle, [0, n) \times \{a\}) \in \mathcal{F}_T[\![P \setminus \mathbf{N}]\!]$. The limit of these finite approximations is therefore a semi-infinite behaviour of $P \setminus \mathbf{N}$:

$$(\langle\rangle, [0, \infty) \times \{a\}) \in \mathcal{SI}_T[\![P \setminus \mathbf{N}]\!]$$

However, there is no single semi-infinite behaviour of P that gives rise to that behaviour of $P \setminus \mathbf{N}$: that is, there is no trace w of P such that $w \setminus \mathbf{N} = \langle\rangle$ and

$$(w, [0, \infty) \times (\mathbf{N} \cup \{a\})) \in \mathcal{SI}_T[\![P]\!]$$

Hence P is not well behaved on $\mathbf{N}$ (and so we would not expect $P_I \sqsubseteq_f P$ to imply that $P_I \setminus \mathbf{N} \sqsubseteq_f P \setminus \mathbf{N}$).

Lemma 6.3.31 *If Q is well behaved on A, and $P \sqsubseteq_f Q$, then $P \setminus A \sqsubseteq_f Q \setminus A$*

Proof Consider

$$(s, \aleph) \in \mathcal{SI}_T[\![Q \setminus A]\!] \wedge ([t, \infty) \times X) \subseteq \aleph$$

Then

$$(s, [t, \infty) \times X) \in \mathcal{SI}_T[\![Q \setminus A]\!]$$

so

$$\exists\, w \bullet s = w \setminus A \wedge (w, [t, \infty) \times X \cup [0, \infty \times A) \in \mathcal{SI}_T[\![Q]\!]$$

Now $P \sqsubseteq_f Q$, so

$$(tstrip(w), A \cup X) \in \mathcal{F}[\![P]\!]$$

so

$$(tstrip(w) \setminus A, X) \in \mathcal{F}[\![P \setminus A]\!]$$

so

$$(tstrip(s), X) \in \mathcal{F}[\![P \setminus A]\!]$$

as required. □

We now identify conditions for P to be well behaved on A.

Lemma 6.3.32 *If P is weakly limited on A and weakly prompt, then P is well behaved on A*

Proof Let P be weakly t-prompt. Consider

$$(s, [t', \infty) \times B) \in \mathcal{SI}_T[\![P \setminus A]\!]$$

Then

$$\exists\, n \bullet w \in traces(\mathcal{F}_T[\![P]\!]) \wedge w \setminus A = s \Rightarrow \#w < n$$

Define $T = max\{end(s), t'\} + nt + 1$. Then

$$(s, [t', T) \times B) \in \mathcal{F}_T[\![P \setminus A]\!]$$

so

$$\exists\, w \bullet w \setminus A = s \wedge (w, [t', T) \times B \cup [0, T) \times A) \in \mathcal{F}_T[\![P]\!]$$

Then

$$\#(w \upharpoonleft end(s)) < n$$

so

$$\exists\, T' < T - t \bullet T' > t' \wedge w \uparrow [T', T' + t] = \langle\rangle$$

Define $w' = w \upharpoonright T' = w \upharpoonright T' + t$. Then

$$(w', [t', T' + t) \times B \cup [0, T' + t) \times A) \in \mathcal{F}_T[\![P]\!]$$

so since

$$T' > end(w') \wedge [T', T' + t) \times A \cup B \subseteq [t', T' + t) \times B \cup [0, T' + t) \times A$$

we have by weak promptness

$$(w', [t', \infty) \times B \cup [0, \infty) \times A) \in \mathcal{F}_T[\![P]\!]$$

as required. □

Observe that we require that P is weakly limited: if it is not, then the result need not follow. Consider the following process definition:

$$\begin{aligned} Q_{n,0} &\mathrel{\hat=} b \rightarrow STOP \\ Q_{n,m+1} &\mathrel{\hat=} n \rightarrow Q_{n,m} \\ Q &\mathrel{\hat=} n : \mathbf{N} \rightarrow Q_{n,n} \end{aligned}$$

Then Q is weakly prompt, but is not weakly limited on $\mathbb{N}$. We see that

$$(\langle\rangle, [0,\infty) \times \{b\}) \in \mathcal{SI}_T[\![Q \setminus \mathbb{N}]\!]$$

since all the finite approximations are failures of $P \setminus \mathbb{N}$. However, there is no finite trace w such that

$$(w, [0,\infty) \times \mathbb{N} \cup \{b\}) \in \mathcal{SI}_T[\![Q]\!] \wedge w \setminus \mathbb{N} = \langle\rangle$$

so Q is not well behaved on $\mathbb{N}$. Indeed, although $\Theta(Q) \sqsubseteq_f Q$, it is not the case that $\Theta(Q) \setminus \mathbb{N} \sqsubseteq_f Q \setminus \mathbb{N}$.

Corollary 6.3.33 *If P is limited on A and prompt, then P is well behaved on A*

The next corollary will be useful in the construction of networks of processes.

Corollary 6.3.34 *If $P_1 \sqsubseteq_f Q_1$ and $P_2 \sqsubseteq_f Q_2$ and Q_1 and Q_2 are both weakly prompt, Q_1 is non-retracting on A and weakly limited on A', Q_2 is non-retracting on B and weakly limited on B', and $A \cup B = A' \cup B' = X \cap Y$, then*
$(P_1 \;{}_X\|_Y\; P_2) \setminus (X \cap Y) \sqsubseteq_f (Q_1 \;{}_X\|_Y\; Q_2) \setminus (X \cap Y)$

Proof this follows directly from theorem 5.3.13 and corollary 6.3.32 □

We now present a sufficient condition for arbitrary non-deterministic choice to preserve $\sqsubseteq_f$-refinement:

Definition 6.3.35 *A set of processes $\{P_i\}$ are well behaved if*

$$\mathcal{SI}_T[\![\textstyle\bigsqcap_i P_i]\!] = \bigcup_i \mathcal{SI}_T[\![P_i]\!]$$

Not all sets of processes are well behaved. If $P_n = WAIT\ n\ ;\ a \to STOP$ then we have

$$(\langle\rangle, [0,\infty) \times \{a\}) \in \mathcal{I}_T[\![\textstyle\bigsqcap_i P_i]\!]$$

but

$$(\langle\rangle, [0,\infty) \times \{a\}) \notin \bigcup_i \mathcal{I}_T[\![P_i]\!]$$

so the set $\{P_i\}$ is not well behaved.

Lemma 6.3.36 *If $\{Q_i\}$ is well behaved, and $\forall\, i \bullet P_i \sqsubseteq_f Q_i$ then $\bigsqcap_i P_i \sqsubseteq_f \bigsqcap_i Q_i$.*

Proof Let $\{Q_i\}$ be well behaved, and $\forall\, i \bullet P_i \sqsubseteq_f Q_i$. Consider

$$(s, \aleph) \in \mathcal{SI}_T\llbracket \bigsqcap_i Q_i \rrbracket \bullet [t, \infty) \times A \subseteq \aleph$$

Then

$$\begin{array}{rl} & (s, \aleph) \in \bigcup_i \mathcal{SI}_T\llbracket Q_i \rrbracket \\ \Rightarrow & \exists\, i \bullet (s, \aleph) \in \mathcal{SI}_T\llbracket Q_i \rrbracket \\ \Rightarrow & \exists\, i \bullet (tstrip(s), A) \in \mathcal{F}\llbracket P_i \rrbracket \\ \Rightarrow & (tstrip(s), A) \in \mathcal{F}\llbracket \bigsqcap_i P_i \rrbracket \end{array}$$

as required. □

We have seen that stable and prompt processes are useful because they permit inferences concerning infinite behaviours from finite approximations to those behaviours. This enables finitary characterisations of the $\sqsubseteq_f$-refinement relation for such processes.

Stable Processes

If Q is a stable process, then we have

$$P \sqsubseteq_f Q \Leftrightarrow (tstrip(s), \sigma(\aleph \upharpoonright \alpha)) \in \mathcal{F}_T\llbracket P \rrbracket$$

This follows from the fact that for stable processes we have

$$(\exists (s, \aleph) \in \mathcal{SI}_T\llbracket P \rrbracket \bullet [t, \infty) \times X \subseteq \aleph \Leftrightarrow \exists (s, \alpha, \aleph) \in \mathcal{E}_T\llbracket P \rrbracket \bullet \sigma(\aleph \upharpoonright \alpha) = X$$

This characterisation of refinement for stable processes will be easier to work with.

Prompt Processes

If Q is a non-retracting t-prompt process then we have

$$\begin{array}{l} P \sqsubseteq_f Q \equiv \\ \quad (s, \aleph) \in \mathcal{F}_T\llbracket Q \rrbracket \wedge [end(s), end(s) + t) \times X \subseteq \aleph \Rightarrow (tstrip(s), X) \in \mathcal{F}\llbracket P \rrbracket \end{array}$$

Once again, we need only consider finite observations in order to tell whether or not a process is a refinement of another.

The Non-determinism Order

The non-determinism partial order $\sqsubseteq$ [Ros82, Hoa85] is defined between processes in a semantic model (with $\mathcal{E}$ as the corresponding semantic mapping) by

Definition 6.3.37

$$P \sqsubseteq Q \;\; \widehat{=} \;\; \mathcal{E}[\![P \sqcap Q]\!] = \mathcal{E}[\![P]\!]$$

Refinement in the non-determinism order preserves $\sqsubseteq_f$-refinement

Theorem 6.3.38

$$\begin{aligned} P \sqsubseteq P' \sqsubseteq_f Q &\;\;\Rightarrow\;\; P \sqsubseteq_f Q \\ P \sqsubseteq_f Q \sqsubseteq Q' &\;\;\Rightarrow\;\; P \sqsubseteq_f Q' \end{aligned}$$

Proof This follows immediately from the facts that

$$\begin{aligned} \mathcal{E}_T[\![P \sqcap Q]\!] = \mathcal{E}_T[\![P]\!] &\;\;\Rightarrow\;\; fail(\mathcal{E}_T[\![Q]\!]) \subseteq fail(\mathcal{E}_T[\![P]\!]) \\ \mathcal{F}[\![P \sqcap Q]\!] = \mathcal{F}[\![P]\!] &\;\;\Rightarrow\;\; fail(\mathcal{F}[\![Q]\!]) \subseteq fail(\mathcal{F}[\![P]\!]) \end{aligned}$$

□

Specifications

The treatment for $\sqsubseteq_f$-refinement of specifications is very similar to the treatment for $\sqsubseteq_t$-refinement. We extend $\sqsubseteq_f$-refinement to a relation between predicates on M_F and TM_{FS}. If S_1 is a predicate on M_F, and S_2 is a predicate on TM_{FS} then we may define $\sqsubseteq_f$ between specifications in an identical fashion to the corresponding definition for $\sqsubseteq_t$-refinement:

Definition 6.3.39

$$S_1 \sqsubseteq_f S_2 \mathrel{\widehat{=}} \forall\, Q \bullet (\exists\, P \bullet S_1(P) \wedge P \sqsubseteq_f Q \Rightarrow S_2(Q))$$

We can also define the strongest refinement $sr(S)$ of a given untimed predicate S in the same way:

$$S \sqsubseteq_f sr(S) \wedge ((S_1 \Rightarrow sr(S)) \Leftrightarrow S \sqsubseteq_f S_1)$$

It is given by a similar condition:

$$sr(S)(Q) \Leftrightarrow \exists\, P \bullet S(P) \wedge P \sqsubseteq_f Q$$

We can again capture the *weakest coarsening* of a given timed predicate T. It is given by the following:

$$wc(T)(P) \;\Leftrightarrow\; \forall\, Q \bullet (P \sqsubseteq_t Q \Rightarrow T(Q))$$

We can express the strongest refinement of a behavioural specification on M_F as a behavioural specification on the semi-infinite behaviours of a timed process:

Theorem 6.3.40

$$sr(Y \textbf{ sat } A(tr, X))(Q) \;\Leftrightarrow\; Q \textbf{ sat } (\aleph = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$$

Proof It is clear that Y **sat** $(\aleph = [t,\infty) \times X \Rightarrow A(tstrip(s), X))$ is a refinement of Y sat $A(tr, X)$. Further, if Q **sat** $(\aleph = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$, then the set

$$U = \{(tstrip(s), X) \mid \exists\, t \bullet (s, [t, \infty) \times X) \in \mathcal{SI}_T[\![Q]\!]\}$$

satisfies the axioms of M_F (page 187) and has $A(u, X)$ for any $(u, X) \in U$. We define the following processes by infinite mutual recursion, indexed by failures (u, X) and traces v:

$$\begin{array}{rcll} P_{(u,X),v} & = & a : (inits(U \upharpoonright v)) \rightarrow P_{(u,X),v^\frown\langle a\rangle} & \text{if } u \neq v \\ P_{(u,X),v} & = & a : (inits(U \upharpoonright v) - X) \rightarrow P_{(u,X),v^\frown\langle a\rangle} & \text{if } u = v \end{array}$$

Now axiom 4 for M_F yields for any $(u, X) \in U$ that

$$(u, X) \in \mathcal{F}[\![P_{(u,X),\langle\rangle}]\!] \subseteq U$$

so we obtain

$$U \subseteq \bigcup_{(u,X)\in U} (P_{(u,X),\langle\rangle}) \subseteq U$$

and hence

$$\mathcal{F}_T[\![\bigsqcap_{(u,X)\in U} (P_{(u,X),\langle\rangle})]\!] = U$$

We thus have a process, defined in terms of infinite mutual recursion, whose semantics is U. It is shown in [Ros88b, p69] that there is a simple coding trick which converts any mutual recursion into a single one. Hence there is a *CSP* process (with only single recursion) whose semantics is the set U. $\square$

Lemma 6.3.41

$$\begin{array}{rl} & Y \textbf{ sat } (\aleph = [t, \infty) \times X \Rightarrow A(tstrip(s), X)) \\ \Leftrightarrow & Y \textbf{ sat } (([t, \infty) \times X \subseteq \aleph) \Rightarrow A(tstrip(s), X)) \end{array}$$

Proof Assume Y **sat** $(\aleph = [t,\infty) \times X \Rightarrow A(tstrip(s), X))$. Then consider $(s, \aleph) \in \mathcal{SI}_T\llbracket Y \rrbracket$ such that $[t,\infty) \times X \subseteq \aleph$. By axiom 10 for TM_{FS} we deduce $(s, [t,\infty) \times X) \in \mathcal{SI}_T\llbracket Y \rrbracket$, and hence that $A(tstrip(s), X))$, as required.

The proof in the other direction is trivial. □

We define the weakest coarsening of a predicate on TM_{FS} (with respect to $\sqsubseteq_f$) in the way it was defined for $\sqsubseteq_t$-refinement:

Definition 6.3.42

$$wc(T)(P) \Leftrightarrow \forall\, Q \bullet (P \sqsubseteq_f Q \Rightarrow T(Q))$$

The weakest coarsening of T holds of P precisely when T holds of every refinement of P. We obtain the following result:

Lemma 6.3.43

$$wc(Y \textbf{ sat } (\aleph = [t,\infty) \times X \Rightarrow S(tstrip(s), X)) = Y \textbf{ sat } S(tr, X)$$

We again obtain $S = wc(sr(S))$ for behavioural specifications on M_F.

Deadlock Freedom

Following [Dat85], we approach deadlock-freedom via possibility of deadlock:

Definition 6.3.44 *A TCSP process P* may deadlock *if*

$$\exists (s, \aleph), t \bullet \mathcal{F}_T\llbracket P/((s,\aleph),t) \rrbracket = \mathcal{F}_T\llbracket P/((s,\aleph),t) \sqcap STOP \rrbracket$$

This is equivalent to saying that

$$\exists (s, \aleph), t \bullet (s, \aleph \cup [t,\infty) \times \Sigma) \in \mathcal{SI}_T\llbracket P \rrbracket$$

This leads to a definition of deadlock free.

Definition 6.3.45 *A process P is* deadlock-free *if*

$$(s, \aleph) \in \mathcal{SI}_T\llbracket P \rrbracket \Rightarrow \forall\, t \bullet [t,\infty) \times \Sigma \not\subseteq \aleph$$

Lemma 6.3.46 *sr(deadlock-free) = deadlock-free*

Proof P is deadlock-free in M_F if P **sat** $X \neq \Sigma$. Hence we obtain from theorem 6.3.40 and lemma 6.3.41 that

$$sr(Y \textbf{ sat } X \neq \Sigma) = Y \textbf{ sat } (([t,\infty) \times X \subseteq \aleph) \Rightarrow X \neq \Sigma)$$

which is equivalent to our definition of deadlock-freedom for TM_{FS}.

We also define deadlock-freedom for TM^*_{FS}.

Definition 6.3.47 *A process P is* deadlock-free in TM^*_{FS} *if*

$$s \in traces(\mathcal{E}^*_T[\![P]\!]) \Rightarrow (s, \Sigma) \notin fail(\mathcal{E}^*_T[\![P]\!])$$

Lemma 6.3.48 *If P is a deadlock-free CSP process, and Q is a TCSP process such that $P \sqsubseteq_{ts} Q$, then Q is deadlock-free in TM^*_{FS}.*

Proof Trivial

Lemma 6.3.49 *If a TCSP process Q is deadlock-free in TM^*_{FS} then it is deadlock-free in TM_{FS}*

Proof If Q is deadlock-free in TM^*_{FS} then it must be stable (since Σ is refusible after an unstable trace), and unable to refuse Σ after stability. Since Q follows Q, we also have that Q is stable in TM_{FS}, and unable to refuse Σ after stability. Hence Q is deadlock-free in TM_{FS}. □

Hence we see that deadlock-freedom is preserved by both $\sqsubseteq_f$-refinement and $\sqsubseteq_{ts}$-refinement.

6.4 Other Timewise Refinements

If we wish to refine processes to retain the stability information, we would wish our refinement to be unstable only if the original process were unstable. Hence we can define a timewise refinement relation $\sqsubseteq_s$ relation between M_S and TM_{FS}.

Definition 6.4.1 *If $S_1 \in M_S$, and $S_2 \in TM^*_{FS}$, then*

$$\begin{aligned} &S_1 \sqsubseteq_s S_2 \mathrel{\widehat=} \\ &\quad tstrip(traces(S_2)) \subseteq traces(S_1) \wedge (s,\infty) \in stab(S_2) \Rightarrow (tstrip(s),\infty) \in S_1 \end{aligned}$$

Hence this form of timewise refinement preserves stability in processes: if P is stable, and $P \sqsubseteq_s Q$, then Q is stable.

Definition 6.4.2 *We define a timewise refinement relation* $\sqsubseteq$ *between* M_S *and* TM^*_{FS} *as follows: if* $S_1 \in M_S$, *and* $S_2 \in TM^*_{FS}$, *then*

$$S_1 \sqsubseteq S_2 \mathrel{\widehat{=}}$$
$$tstrip(traces(S_2)) \subseteq traces(S_1) \wedge (s,\infty) \in stab(S_2) \Rightarrow (thstrip(s),\infty) \in S_1$$

The refinement relations $\sqsubseteq_f$ and $\sqsubseteq_s$ may be combined to yield a refinement relation between M_{FS} and TM_{FS}:

Definition 6.4.3 *The refinement relation* $\sqsubseteq$ *between* M_{FS} *and* TM_{FS} *is defined as follows: if* $S_1 \in M_{FS}$ *and* $S_2 \in TM_{FS}$, *then*

$$\begin{array}{rcl} S_1 \sqsubseteq S_2 & \widehat{=} & (\exists\, t \bullet (s,[t,\infty) \times X) \in SI(S_2)) \Rightarrow (tstrip(s), X) \in fail(S_1)) \\ & & \wedge\, (\exists\, t, \forall\, T > t \bullet (s,\infty,[t,T) \times X) \in S_2) \Rightarrow (tstrip(s),\infty,X) \in S_1 \end{array}$$

Definition 6.4.4 *The refinement relation* $\sqsubseteq$ *between* M_{FS} *and* TM^*_{FS} *is defined as follows: if* $S_1 \in M_{FS}$ *and* $S_2 \in TM^*_{FS}$, *then*

$$\begin{array}{rcll} S_1 \sqsubseteq S_2 & \widehat{=} & (s,\alpha,X) \in S_2 \Rightarrow & (thstrip(s),X) \in fail(S_1) \\ & & & \wedge\, (\alpha = \infty \Rightarrow (thstrip(s),\infty,X) \in S_1) \end{array}$$

Each of the refinement relations above extends to environments and to relations between *CSP* and *TCSP* terms, in the same way as the relations $\sqsubseteq_t$ and $\sqsubseteq_f$ were extended.

7 Communication

In this chapter we define the communication concepts of channels, input and output, pipes, the chaining operator, buffers, and networks. The treatment of buffers in particular provides illustrations of the specifications we have presented in the previous chapters. We find sufficient conditions for the preservation of $\sqsubseteq_f$-refinement by the chaining operator, and thereby obtain conditions on buffers which ensure that the pipe formed by chaining them together is a buffer. We also consider the more general network operator: we see how to modularise networks, and provide conditions for the network operator to preserve timewise refinement.

7.1 Definitions

We think of the event $c.v$ as corresponding to value v being passed along channel c. If V is the set of possible values that can pass along channel c, we define $c.V = \{c.v \mid v \in V\}$. When it is clear that we intend 'c' to represent a channel, then we use c as an abbreviation for $c.V$.

$$\begin{aligned} c?x : V \rightarrow P(x) &\mathrel{\widehat{=}} c.x : c.V \rightarrow Q(c.x) \text{ where } Q(c.x) = P(x) \\ c!x \rightarrow P &\mathrel{\widehat{=}} c.x \rightarrow P \end{aligned}$$

$c?x : V \rightarrow P(x)$ represents a process willing to input any message x from the set V along channel c, and then behave like $P(x)$. Observe that it is only well defined if the set $\{P(x) \mid x \in V\}$ is uniformly bounded. If the set of possible messages V is obvious from the coutext, then we may omit it and write $c?x \rightarrow P(x)$.

The construction $c!x \rightarrow P$ represents a process willing to output message x along channel c, and then behave like process P.

In specifications we often use the channel name to denote the projection of the trace onto the channel: the sequence of messages occurring on that channel in the trace s. For example, c denotes $tstrip(s \upharpoonright c)$. This convention will allow us to succinctly relate the messages passed along different channels in specifications. We therefore write $out \leqslant in$ as shorthand for $tstrip(s \upharpoonright out.V) \leqslant tstrip(s \upharpoonright in.V)$, to specify that the sequence of messages passed on the out channel is a prefix of the sequence passed on the in channel. ($tr_1 \leqslant tr_2$ means that tr_1 is a prefix of tr_2; also, from [PS88], $tr_1 \leqslant_n tr_2$ means that $tr_1 \leqslant tr_2$ and $\#tr_1 + n \geqslant \#tr_2$.)

Definition 7.1.1 *A* pipe *is a process P such that $\sigma(P) \subseteq in.\Sigma \cup out.\Sigma$*

The definitiou of the chaining operator is the same as that of the untimed operator, but recall that the hiding operator forces all internal communication to occur as soon as possible; its behaviour will therefore be somewhat different.

Definition 7.1.2 *We define the chaining operator $\gg$ on pipes P, Q as follows:*

$$P \gg Q \quad = \quad (S_{out,c}(P) \ {}_{\{in,c\}}\|_{\{c,out\}} \ S_{c,in}(Q)) \setminus c$$

where $S_{a,b}$ is an alphabet transformation, indexed by channel names, defined by

$$\begin{aligned} S_{a,b}(a.x) &= b.x \\ S_{a,b}(b.x) &= a.x \\ S_{a,b}(y) &= y \ \text{ if } y \notin a.\Sigma \cup b.\Sigma \end{aligned}$$

Theorem 7.1.3 *The chaining operator $\gg$ is associative*

Proof Lemma A.1.4 and the following previous results are sufficient to prove Theorem 7.1.3:

1. $P \ {}_X\|_{Y \cup Z} \ (Q \ {}_Y\|_Z \ R) = (P \ {}_X\|_Y \ Q) \ {}_{X \cup Y}\|_Z \ R$
2. $\{a, b\} \cap \sigma(P) = \{\} \Rightarrow S_{a,b}(P) = P$
3. f bijective $\Rightarrow f(P \ {}_X\|_Y \ Q) = f(P) \ {}_{f(X)}\|_{f(Y)} \ f(Q) \wedge f(P \setminus c) = f(P) \setminus f(c)$
4. $\forall \, a, b \bullet S_{a,b}$ is bijective
5. $\{a, b\} \cap \{c, d\} = \{\} \Rightarrow S_{a,b}(S_{c,d}(P)) = S_{c,d}(S_{a,b}(P))$
6. $P \setminus X \setminus Y = P \setminus Y \setminus X$

Lemma A.1.4 states that $Z \cap Y = \emptyset \Rightarrow (P \setminus Z) \ {}_X\|_Y \ Q = (P \ {}_{X \cup Z}\|_Y \ Q) \setminus Z$

Let ic, co, id, do, idc, cdo abbreviate $in \cup c, c \cup out, in \cup d, d \cup out, in \cup d \cup c, andc \cup d \cup out$ respectively. The proof is essentially the same as the proof for associativity of chaining in untimed *CSP*; the difference arises in the establishment of the previous results, in particular lemma A.1.4.

$$
\begin{aligned}
& (P \gg Q) \gg R \\
= \; & (S_{out,d}((S_{out,c}(P) \; {}_{ic}\|_{co} \; S_{c,in}(Q)) \setminus c) \; {}_{id}\|_{do} \; S_{d,in}(R)) \setminus d && \text{(defn)} \\
= \; & ((S_{out,d}(S_{out,c}(P) \; {}_{ic}\|_{co} \; S_{c,in}(Q)) \setminus S_{out,d}(c)) \; {}_{id}\|_{do} \; S_{d,in}(R)) \setminus d && (3,4) \\
= \; & (((S_{out,d}(S_{out,c}(P) \; {}_{ic}\|_{co} \; S_{c,in}(Q))) \; {}_{idc}\|_{do} \; S_{d,in}(R)) \setminus c) \setminus d && (A.1.4) \\
= \; & (((S_{out,d}(S_{out,c}(P) \; {}_{ic}\|_{co} \; S_{c,in}(Q))) \; {}_{idc}\|_{do} \; S_{d,in}(R)) \setminus d) \setminus c && (6) \\
= \; & ((((S_{out,d}(S_{out,c}(P))) \; {}_{ic}\|_{cd} \; S_{out,d}(S_{c,in}(Q))) \; {}_{idc}\|_{do} \; S_{d,in}(R)) \setminus d) \setminus c && (3,4) \\
= \; & ((S_{out,c}(P) \; {}_{ic}\|_{cdo} \; (S_{out,d}(S_{c,in}(Q)) \; {}_{cd}\|_{do} \; S_{d,in}(R))) \setminus d) \setminus c && (1,2) \\
= \; & ((S_{out,c}(P) \; {}_{ic}\|_{cdo} \; (S_{c,in}(S_{out,d}(Q)) \; {}_{cd}\|_{do} \; S_{d,in}(R))) \setminus d) \setminus c && (5) \\
= \; & ((S_{out,c}(P) \; {}_{ic}\|_{co} \; ((S_{c,in}(S_{out,d}(Q)) \; {}_{id}\|_{do} \; S_{d,in}(R)) \setminus d)) \setminus c && (A.1.4) \\
= \; & P \gg (Q \gg R) && \text{(defn)}
\end{aligned}
$$

□

7.2 Real-time Buffers

In addition to being interesting in their own right, hnffers also provide us with nice specification examples: their definition involves both safety and liveness properties, we may impose real-time constraints on them, and we may also consider the interaction of various types of buffer.

In [Hoa85] a buffer is defined as a process which has two channels, *in* and *out*, and outputs on the *out* channel exactly the same sequence of messages as it has input from the *in* channel, although possibly after some delay ('delay' in the untimed context meaning that it need not output messages as soon as they have been input, but may have the capacity for inputting further messages before outputting). This has a remarkably simple specification in the traces model — A process P is a buffer if it is a pipe such that

$$P \text{ sat } out \leqslant in$$

Unfortunately, processes such as $STOP$ satisfy this specification, not through any fault in the specification, but rather because the traces model is not powerful enough to capture the essence of a buffer. All the traces model can describe is safety properties; but we also wish to capture liveness properties.

We want a buffer to be unable to refuse an input if it is empty (that is, if $out = in$, when it has output all its input messages), and to be unable to refuse an output if it is not empty, when $out < in$. This leads us to the specification [Hoa85, p158]:

$$P \textbf{ sat } (out \leqslant in \wedge \textbf{ if } out = in \textbf{ then } in \,\tilde{\in}\, ref \text{ else } out \,\tilde{\in}\, ref)$$

if we are to call P a buffer.

We extend this definition to include time. As we decided in our discussion on timewise refinement, we consider that a process is able to refuse a set of events after a trace if there is a time after which that set can always be refused; and it is not able to refuse that set if there is no such time. We say, therefore, tbat a process is a timed buffer if there is no time after which it can always refnse to output when non-empty, and that there is no time after which it can always refuse to input if it is empty. We define a buffer formally as follows:

Definition 7.2.1 *P is a buffer if it is a pipc such that*

$$\begin{array}{lll} (s,\aleph) \in \mathcal{SI}_T[\![P]\!] \Rightarrow & (out \leqslant in) & \\ & \wedge \;\; \textbf{if} & out = in \\ & \quad\;\; \textbf{then} & \forall\, t \in [end(s), \infty), u \in C \bullet [t,\infty) \times in.u \not\subseteq \aleph \\ & \quad\;\; \textbf{else} & \forall\, t \in [end(s), \infty) \bullet [t,\infty) \times out \not\subseteq \aleph \end{array}$$

where C is the set of possible messages.

Lemma 7.2.2 *'timed buffer' = sr('untimed buffer')*

Proof Let $B(P)$ hold if and only if P is an untimed buffer, and $C(Q)$ hold if and only if Q is a timed buffer. We have

$$sr(B)(Q) \Leftrightarrow \exists\, P \bullet B(P) \wedge P \sqsubseteq_t Q$$

so we are required to show that

$$C(Q) \Leftrightarrow \exists\, P \bullet B(P) \wedge P \sqsubseteq_t Q$$

We prove first that

$$C(Q) \Leftarrow \exists\, P \bullet B(P) \wedge P \sqsubseteq_t Q$$

Consider $(s,\aleph) \in \mathcal{SI}_T[\![Q]\!], [t,\infty) \times A \subseteq \aleph$.

$$\begin{array}{ll} & (tstrip(s), A) \in \mathcal{F}[\![P]\!] \wedge out \leqslant in \\ & \wedge\; out = in \Rightarrow in.u \notin A \\ & \wedge\; out < in \Rightarrow out \not\subseteq A \\ \Rightarrow & out \leqslant in \\ & \wedge\; out = in \Rightarrow [t,\infty) \times in.u \not\subseteq \aleph \\ & \wedge\; out < in \Rightarrow [t,\infty) \times out \not\subseteq \aleph \end{array}$$

so Q is a timed buffer.

We now show that

$$C(Q) \Rightarrow \exists\, P \bullet B(P) \wedge P \sqsubseteq_f Q$$

Define:

$$\begin{array}{rcl}
B_{\langle\rangle} & = & in?x \rightarrow B_{\langle x\rangle} \\
B_{\langle y\rangle^\frown s} & = & out!y \rightarrow B_s \\
 & & \sqcap \\
 & & in?x \rightarrow B_{\langle y\rangle^\frown s^\frown \langle x\rangle} \;\Box\; out!y \rightarrow B_s \\
P & = & B_{\langle\rangle}
\end{array}$$

Then P is the most non-deterministic buffer. Its failure set is given by

$$\begin{array}{rcl}
\mathcal{F}[\![P]\!] & = & \{(s, X) \mid out \leqslant in \wedge (out = in \Rightarrow \forall\, u : C \bullet in.u \notin X) \\
 & & \qquad\qquad\qquad \wedge\ out < in \Rightarrow out \not\subseteq X\}
\end{array}$$

Consider $Q \in TCSP$ such that $C(Q)$. Let $(s, \aleph) \in \mathcal{SI}_T[\![Q]\!]$. Then $out \leqslant in$. Now let $[t, \infty) \times A \subseteq \aleph$.

Case $out = in$

Then $\forall\, u : C \bullet in.u \notin A$.

Case $out < in$

Then $out \not\subseteq A$.

Hence in either case, $(tstrip(s), A) \in \mathcal{F}[\![P]\!]$. Therefore $P \sqsubseteq_f Q$, as required.
□

Timewise refinement preserves the property tbat a process is a buffer.

Categorising buffers

Buffers may be classified according to tbeir satisfaction of the various aspects of good behaviour detailed in chapter 5. In particular, buffers which are non-retracting and prompt, or have bounded stability, are especially well behaved. As we shall see, the following kinds of buffer interact in useful ways:

- prompt buffers

- buffers with bounded stability
- non-retracting buffers
- buffers that are non-retracting on *in*, or on *out*
- buffers which are impartial on *in*
- responsive buffers

Buffers are by definition live, so promptness is stronger than responsiveness for them:

Theorem 7.2.3 *A t-prompt buffer is t-responsive*

Proof By contradiction. Assume that we have a t-prompt buffer that is not t-responsive. Then

$$\exists (s, \aleph) \in \mathcal{F}_T[\![P]\!], T \in [0, \infty) \bullet [T, T+t) \times \Sigma \subseteq \aleph \wedge s \uparrow [T, T+t] = \langle\rangle$$

So $(s \upharpoonright T+t, \aleph \upharpoonright T+t) \in \mathcal{F}_T[\![P]\!]$. But $end(s) < T$, so by promptness and the fact that $[T, T+t) \times \Sigma \subseteq \aleph \upharpoonright T+t$ we obtain

$$(s \upharpoonright T, \aleph \upharpoonright (T+t) \cup [T, \infty) \times \Sigma) \in \mathcal{F}_T[\![P]\!]$$

which contradicts the claim that P is a buffer. □

However, not every t-responsive buffer is t-prompt. For example, consider the buffer B defined by

$$\begin{array}{rcl} B & = & \mu X \bullet in?n : \mathbb{N} \rightarrow (in?y \rightarrow out!n \rightarrow out!y \rightarrow X \\ & & \qquad\qquad\qquad \Box \\ & & \qquad\qquad\qquad WAIT\ n\ ; out!n \rightarrow X \end{array}$$

This process is t-responsive for any $t > 2\delta$, but there is no t for which it is t-prompt.

Hence the specification 'responsive buffer' is strictly weaker than the specification 'prompt buffer'.

The next two theorems are useful in establishing that processes are buffers; Examples of their use will be presented in the next chapter.

Theorem 7.2.4 *If P* **sat** *$out \leqslant_1 in$, and P is $in \cup out$-responsive and impartial on in, then B is a responsive one-place buffer*

Theorem 7.2.5 *If P* **sat** *$out \leqslant_1 in$, P is a deadlock-free pipe and P is impartial on in, then P is a one-place buffer.*

COPY — a well behaved buffer

The process $COPY$ has the same definition as its untimed counterpart.

$$COPY \ \widehat{=}\ \mu X \bullet in?x \rightarrow out!x \rightarrow X$$

The results of the preceding chapters yield the following aspects of good behaviour exhibited by $COPY$:

- $COPY \sqsubseteq_t COPY$
- $COPY$ **sat** $out \leqslant_t in$, since $\Theta(COPY)$ **sat** $out \leqslant_t in$ in M_T
- $COPY \sqsubseteq_f COPY$
- $COPY$ is 2δ-stable
- $COPY$ is non-retracting
- $COPY$ is t'-prompt $(t' > 2\delta)$
- $COPY$ is t'-responsive $(t' > 2\delta)$
- $COPY$ is limited on out
- $COPY$ is a buffer

7.3 Chaining

The specification $buffer(tr)$ of a buffer in the traces model M_T is that $out \leqslant in$. It is clear that any timed buffer P must satisfy $sr(buffer(tr))$, and hence that if P and Q are two buffers, then $P \gg Q$ **sat** $sr(buffer(tr))$, since chaining preserves $buffer(tr)$ in M_T. Hence for any two buffers P and Q we obtain that $P \gg Q$ is a pipe with $out \leqslant in$. However, P and Q may fail to synchronise on their mutual channel. Consider

$$\begin{aligned} P &= (\mu X \bullet ((in?x \rightarrow STOP) \underset{2}{\lightning} P_x) \overset{1}{\rhd} WAIT\ 3\ ; X) \\ P_x &= (\mu X \bullet ((out!x \rightarrow STOP) \underset{2-\delta}{\lightning} P) \overset{1}{\rhd} WAIT\ 3\ ; X) \end{aligned}$$

P is a cyclic process of cycle length $4 + 2\delta$. When empty, it is prepared to input on the first time unit of its cycle. When nou-empty, it is prepared to output on the third time unit of the cycle. Hence P is a buffer, but $P \gg P$ is not, since no synchronisations will be possible on the internal channel.

Our interest lies in obtaining sufficient conditions for $P \gg Q$ to be a buffer. This will be the case for those buffers where $\sqsubseteq_f$-refinement is preserved by chaining. In such cases, given buffers P and Q, lemma 7.2.2 yields that we can find buffers B_1 and B_2 which are $\sqsubseteq_f$-refined by P and Q respectively, and hence that $B_1 \gg B_2 \sqsubseteq_f P \gg Q$. Buffer law L1 in [Hoa85, p159] tells us that $B_1 \gg B_2$ is a buffer in M_F, so $P \gg Q$ will be a buffer in TM_{FS}. We therefore seek to identify conditions on P and Q such that chaining preserves timewise refinement for them.

Lemma 7.3.1 *If P and Q are buffers, and P has finite capacity, then*

$$S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$$

is limited on c

Proof Follows immediately from the fact that there is some n (the capacity of P) such that for any trace s of $S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$ we have $c \leqslant_n in$. □

Lemma 7.3.2 *If P and Q are buffers then $S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$ is weakly limited on c*

Proof Follows from the fact that for any trace s of $S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$ we have $c \leqslant in$. □

Lemma 7.3.3 *If P and Q are prompt buffers, and P is non-retracting on output or Q is non-retracting on input, then $P \gg Q$ is a buffer.*

Proof There exists buffers B_1 and B_2 such that $B_1 \sqsubseteq_f P$ and $B_2 \sqsubseteq_f Q$. From theorem 6.3.27 we have that

$$S_{out,c}(B_1) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(B_2) \sqsubseteq_f S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$$

and it follows that $S_{out,c}(P) \;{}_{\{in,c\}}\|_{\{out,c\}}\; S_{in,c}(Q)$ is weakly limited on c and prompt, from lemma 7.3.2 and theorem 5.3.12 respectively, so we have from corollary 6.3.33 and lemma 6.3.31 that $B_1 \gg B_2 \sqsubseteq_f P \gg Q$, and hence that $P \gg Q$ is a buffer. □

Corollary 7.3.4 *If P and Q are prompt buffers, and P is non-retracting or Q is non-retracting, then $P \gg Q$ is a prompt buffer*

Corollary 7.3.5 *If $\{P_i\}_{i=1}^{n}$ is such that each P_i is a buffer, non-retracting on input, and prompt, then $P_1 \gg P_2 \gg \ldots \gg P_n$ is a buffer, non-retracting on input, and prompt.*

Corollary 7.3.6 *If $\{P_i\}_{i=1}^{n}$ is such that each P_i is prompt and non-retracting on output, then $P_1 \gg P_2 \gg \ldots \gg P_n$ is prompt and non-retracting on output.*

Corollary 7.3.7 *If $\{P_i\}_{i=1}^{n}$ is such that each P_i is prompt, then*

$$P_1 \gg COPY \gg P_2 \gg COPY \gg \ldots \gg P_n$$

is prompt

Lemma 7.3.8 *If Q_1 and Q_2 are boundedly stable buffers with finite capacity, then $Q_1 \gg Q_2$ is a boundedly stable buffer with finite capacity.*

Proof We have buffers B_1 and B_2 such that $B_1 \sqsubseteq_f Q_1$ and $B_2 \sqsubseteq_f Q_2$, so from theorem 6.3.29 we have

$$S_{out,c}(B_1)\ {}_{\{in,c\}}\|_{\{out,c\}}\ S_{in,c}(B_2) \sqsubseteq_f S_{out,c}(Q_1)\ {}_{\{in,c\}}\|_{\{out,c\}}\ S_{in,c}(Q_2)$$

Also, from lemma 7.3.1 we have $S_{out,c}(Q_1)\ {}_{\{in,c\}}\|_{\{out,c\}}\ S_{in,c}(Q_2)$ is limited on c, so $B_1 \gg B_2 \sqsubseteq_f Q_1 \gg Q_2$. It follows from theorem 5.6.8 that $Q_1 \gg Q_2$ is boundedly stable. □

Lemma 7.3.9 *If Q_1 and Q_2 are boundedly stable buffers, then $Q_1 \gg Q_2$ is a stable buffer.*

Proof We have buffers B_1 and B_2 such that $B_1 \sqsubseteq_f Q_1$ and $B_2 \sqsubseteq_f Q_2$, so from theorem 6.3.29 we have $S_{out,c}(B_1)\ {}_{\{in,c\}}\|_{\{out,c\}}\ S_{in,c}(B_2) \sqsubseteq_f S_{out,c}(Q_1)\ {}_{\{in,c\}}\|_{\{out,c\}}\ S_{in,c}(Q_2)$. Also, from lemma 7.3.2 we have $S_{out,c}(Q_1)_{\{in,c\}}\|_{\{out,c\}}S_{in,c}(Q_2)$ is weakly limited on c, so $B_1 \gg B_2 \sqsubseteq_f Q_1 \gg Q_2$. It follows from theorem 5.6.7 that $Q_1 \gg Q_2$ is stable. □

Theorem 7.3.10 *If P, Q and $P \gg Q$ are buffers, then*

- *if P is strongly non-retracting on input, then so is $P \gg Q$*
- *if Q is strongly non-retracting on output, then so is $P \gg Q$*
- *if P is in-responsive, then so is $P \gg Q$*
- *if P is prompt on in, then so is $P \gg Q$*

Theorem 7.3.11 *If Q_1 and Q_2 are buffers, and $P_1 \sqsubseteq_f Q_1$ and $P_2 \sqsubseteq_f Q_2$, and Q_1 is non-retracting on output or Q_2 is non-retracting on input, and both Q_1 and Q_2 are prompt, then $P_1 \gg P_2 \sqsubseteq_f Q_1 \gg Q_2$.*

Proof Every buffer has $out \leqslant in$, and so it is weakly limited on out. Hence $(S_{\{out,c\}}(Q_1)\ {}_{\{in,c\}}\|_{\{out,c\}} S_{\{in,c\}}(Q_2)$ is weakly limited on c. The result follows from corollary 6.3.34. □

Corollary 7.3.12 *If $P_1 \sqsubseteq_f Q_1$ and $P_2 \sqsubseteq_f Q_2$ and Q_1 and Q_2 are prompt, then $P_1 \gg COPY \gg P_2 \sqsubseteq_f Q_1 \gg COPY \gg Q_2$*

In order to ensure that Q_1 and Q_2 communicate despite their possible inability to synchronise, we insert $COPY$ along their mutual channel.

7.4 Networks of processes

We generalise the chaining operator to obtain networks of arbitrary size, with the internal events hidden. If we have a set of (*process,interface*) pairs $\{(P_i, X_i)\}$, where an interface is a subset of Σ, then these may be thought of as forming a network. That network is the parallel combination of all those processes with all channels internal to the network hidden. It is defined, as in [Dat85], as follows:

$$\begin{aligned} PAR(\{(P_i, X_i)\}) &= (\mathop{\|}_{X_i} P_i) \\ X &= \bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\} \\ NET(\{(P_i, X_i)\}) &= PAR(\{(P_i, X_i)\}) \setminus X \end{aligned}$$

We insist that any channel name is shared by at most two processes, so the set of interfaces must be triple disjoint:

$$i \neq j \neq k \Rightarrow X_i \cap X_j \cap X_k = \emptyset$$

In order for the network to be a refinement of a corresponding untimed network, where all the components are refinements, it will be sufficient to insist that each component is prompt, and also that each internal channel has one of its two users non-retracting on it.

Theorem 7.4.1 *If*

- $(PAR(\{(P_i, X_i)\}))$ *is weakly limited on* $\bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\}$
- $\forall i \bullet P_i \sqsubseteq_f Q_i$
- $PAR(\{(Q_i, X_i)\})$ *is weakly prompt*

then $PAR(\{(P_i, X_i)\}) \sqsubseteq_f PAR(\{(Q_i, X_i)\})$.

Proof $(PAR(\{(P_i, X_i)\}))$ is weakly limited on $\bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\}$, so it follows from corollary 6.2.13 and lemma 6.2.11, and the fact that all *TCSP* operators preserve $\sqsubseteq_t$-refinement, that $PAR(\{(Q_i, X_i)\}))$ is weakly limited on $\bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\}$. The result follows immediately from lemma 6.3.32 □
Corollary A.1.5 states that

$$Z \cap Y = \emptyset \Rightarrow (P \setminus Z)\ {}_X\|_Y\ Q = (P\ {}_{X \cup Z}\|_Y\ Q) \setminus Z$$

The following consequences are useful:

Corollary 7.4.2

$$(P\ {}_X\|_Y\ Q) \setminus Z \quad = \quad (P \setminus ((X - Y) \cap Z)\ {}_X\|_Y\ Q) \setminus (Z - (X - Y))$$

Corollary 7.4.3

$$\begin{array}{l} (X \cap Y \cap Z) = \emptyset \wedge Z \subseteq (X \cup Y) \\ \quad \Rightarrow \quad (P\ {}_X\|_Y\ Q) \setminus Z = (P \setminus (Z \cap X))\ {}_X\|_Y\ (Q \setminus (Z \cap Y) \end{array}$$

This corollary generalises to a result enabling us to modularise networks:

Corollary 7.4.4 *If the sets I, Y_1 and Y_2 are defined by*

$$\begin{aligned} I &= \bigcup_{i=1}^{m}(\bigcup_{j=m+1}^{n} \{(X_i \cap X_j)\} \\ Y_1 &= I \cup \bigcup_{i=1}^{m}(X_i \setminus (\bigcup_{j \neq i} X_j)) \\ Y_2 &= I \cup \bigcup_{i=m+1}^{n} (X_i \setminus (\bigcup_{j \neq i} X_j)) \end{aligned}$$

then we have

$$\begin{aligned} & NET(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant n\}) \\ & \quad = \quad NET(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant m\}) \;{}_{Y_1}\|_{Y_2}\; NET(\{(Q_i, X_i) \mid m+1 \leqslant i \leqslant n\}) \end{aligned}$$

Theorem 7.4.5 *If $\{P_i\}_{i=1}^{n}$,$\{Q_i\}_{i=1}^{n}$, and $\{X_i\}_{i=1}^{n}$ are such that the set $\{X_i\}_{i=1}^{n}$ is triple disjoint and*

1. *Every Q_i is prompt*
2. *For every i,j, we have either Q_i or Q_j is non-retracting on $X_i \cap X_j$*
3. *for every i, $P_i \sqsubseteq_f Q_i$*
4. *$(PAR(\{(P_i, X_i)\}))$ is weakly limited on $\bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\}$*

then $NET(\{(P_i, X_i)\}) \sqsubseteq_f NET(\{(Q_i, X_i)\})$

Proof We first prove, by induction on r, that

$$PAR(\{(P_i, X_i) \mid 1 \leqslant i \leqslant r\}) \;\; \sqsubseteq_f \;\; PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant r\})$$

The base case ($r = 2$) follows immediately from theorem 6.3.27.

For the inductive step, assume

$$PAR(\{(P_i, X_i) \mid 1 \leqslant i \leqslant r\}) \;\; \sqsubseteq_f \;\; PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant r\})$$

Now define the set

$$I \quad = \quad \{i \mid Q_{r+1} \text{ is not non-retracting on } X_i \cap X_{r+1}\}$$

Then from our assumptions we have that for each $i \in I$ we have that Q_i is non-retracting on $X_i \cap X_r$, and so from theorem 5.1.13 we obtain that

$$PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant r\})$$

is non-retracting on $\bigcup_{i \in I}(X_i \cap X_{r+1})$. We also have that Q_{r+1} is non-retracting on $\bigcup_{i \notin I}(X_i \cap X_{r+1})$. Further, from the promptness conditions we have that both $PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant r\})$ and Q_{r+1} are prompt on $\bigcup_{1 \leqslant i \leqslant r}(X_i \cap X_{r+1})$, and so we are in a position to apply theorem 6.3.27, from which we conclude that

$$PAR(\{(P_i, X_i) \mid 1 \leqslant i \leqslant r + 1\}) \;\sqsubseteq_f\; PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant r + 1\})$$

Thus we may conclude by induction that

$$PAR(\{(P_i, X_i) \mid 1 \leqslant i \leqslant n\}) \;\sqsubseteq_f\; PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant n\})$$

Now, defining

$$A \;=\; \bigcup\{X_i \cap X_j \mid 1 \leqslant i < j \leqslant n\}$$

it follows from lemma 6.2.11 that $PAR(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant n\})$ is weakly limited on A, and from a generalisation of lemma 5.3.12 and theorem 5.3.8 it is also prompt, and hence weakly prompt, so from lemma 6.3.31 and lemma 6.3.32 it follows directly that

$$NET(\{(P_i, X_i) \mid 1 \leqslant i \leqslant n\}) \;\sqsubseteq_f\; NET(\{(Q_i, X_i) \mid 1 \leqslant i \leqslant n\})$$

□

8 Examples

In this chapter, we provide illustrations of the application of the specification and verification methods presented earlier. We first present three examples of specification: in each case, tbe desired behaviour of the system is captured by a combination of behavioural specifications and general properties presented in chapter 5. We then present candidate processes which we claim meet the specification in each case. Our verifications consist of applications of botb the proof system and some laws concerning the specifications. In the verifications of recursively defined processes we use the version of the proof rule that does not require a base case to be established.

As an example of the application of further rules from the proof system, we present a more detailed verification, that a stop and wait protocol has a particular liveness property. The same protocol is then shown to be a buffer by the method of timewise refinement from a previously verified untimed protocol. Timewise refinement is also used in the verification of an alternating bit protocol, and finally a sliding window protocol, by relating them in each case to untimed versions defined and verified in [PS88].

8.1 A Time Server

A time server $TIME$ provides times along a *time* channel. Our specification for it is the conjunction of the following conditions:

1. $TIME$ is responsive on *time*
2. $TIME$ **sat** $ACCURATE(s, \aleph)$, where

$$ACCURATE(s,\aleph) \;\widehat{=}\; \langle (t, time.n) \rangle \in s \Rightarrow 0 \leqslant t - n < 1$$

Our proposed $TCSP$ implementation of a timeserver is given as follows:

$$TIME \;\widehat{=}\; \mu\, X \bullet (\mu\, Y \bullet time.0 \rightarrow Y) \underset{1-3\delta}{\lightning} timesucc(X)$$

where the alphabet transformation *timesucc* is defined by

$$\begin{aligned} timesucc(time.n) \;&\widehat{=}\; time.(n+1) \\ timesucc(a) \;&\widehat{=}\; a \quad \text{if } a \neq time.n \text{ for all } n \end{aligned}$$

Now *timesucc* clearly preserves "initially $(1 - 3\delta, time)$-responsive", so by theorem 5.2.17 it immediately follows that $TIME$ is responsive on *time* (in fact it is $(1, time)$-responsive).

We will use the proof system to show that $TIME$ **sat** $ACCURATE(s, \aleph)$. Abbreviating $A(s, \aleph)$ for $ACCURATE(s, \aleph)$, it will be sufficient to show that the defining function for $TIME$ preserves $A(s, \aleph)$. Assuming X **sat** $A(s, \aleph)$ we require

$$(\mu\, Y \bullet time.0 \rightarrow Y) \underset{1-3\delta}{\lightning} timesucc(X) \quad \mathbf{sat} \quad A(s, \aleph)$$

It is immediate that $\checkmark \notin \sigma(\mu\, Y \bullet time.0 \rightarrow Y)$, so the special case rule for tireout can be applied. From that rule we see that we must find S_1 and S_2 such that

$$\begin{array}{rcl} \mu\, Y \bullet time.0 \rightarrow Y & \mathbf{sat} & S_1 \\ timesucc(WAIT\ \delta\, ; X) & \mathbf{sat} & S_2 \\ \left. \begin{array}{l} S_1(s \upharpoonright (1-3\delta), \aleph \upharpoonright (1-3\delta)) \wedge s \uparrow ((1-3\delta), 1-\delta) = \langle\rangle \\ \wedge\ S_2(s \mathbin{\dot{-}} (1-\delta), \aleph \mathbin{\dot{-}} (1-\delta)) \end{array} \right\} & \Rightarrow & A(s, \aleph) \end{array}$$

Now we have by assumption that X **sat** $A(s, \aleph)$, so we have

$$WAIT\ \delta\, ; X \quad \mathbf{sat} \quad A(s \mathbin{\dot{-}} \delta, \aleph \mathbin{\dot{-}} \delta) \wedge begin(s) \geqslant \delta$$

Hence it follows from the rule for alphabet transformation that

$$\begin{array}{rcl} timesucc(WAIT\ \delta\, ; X) & \mathbf{sat} & A(timesucc^{-1}(s \mathbin{\dot{-}} \delta), timesucc^{-1}(\aleph \mathbin{\dot{-}} \delta)) \\ & & \wedge\ begin(s) \geqslant \delta \wedge \sigma(s) \subseteq ran(timesucc) \end{array}$$

which provides us with a candidate for S_2:

$$\begin{array}{rcl} S_2(s, \aleph) & \widehat{=} & A(timesucc^{-1}(s \mathbin{\dot{-}} \delta), timesucc^{-1}(\aleph \mathbin{\dot{-}} \delta)) \\ & & \wedge\ begin(s) \geqslant \delta \wedge \sigma(s) \subseteq ran(timesucc) \end{array}$$

The relevant property of $\mu\, Y \bullet time.0 \rightarrow Y$ is that the value it is prepared to output on its *time* channel is always 0, so we define S_1 by

$$S_1(s, \aleph) \;\widehat{=}\; \langle(t, time.n)\rangle \text{ in } s \Rightarrow n = 0$$

which may be easily established by recursion induction. We only have to establish the logical condition to complete the proof. It reduces to

$$\left. \begin{array}{l} S_1(s \upharpoonright (1-3\delta), \aleph \upharpoonright (1-3\delta)) \wedge s \uparrow ((1-3\delta), 1) = \langle\rangle \\ \wedge\ A(timesucc^{-1}(s \mathbin{\dot{-}} 1), timesucc^{-1}(\aleph \mathbin{\dot{-}} 1)) \end{array} \right\} \Rightarrow A(s, \aleph)$$

This may be established by a case analysis on a typical timed event of the form $(t, time.n)$ which appears in the trace s. We assume the left hand side of the implication. We are hoping to establish that in each case we have $0 \leqslant t - n < 1$.

Case $t \leqslant 1 - 3\delta$

Then

$$(t, time.n) \textbf{ in } s \restriction (1 - 3\delta)$$

and we have

$$S_I(s \restriction (1 - 3\delta), \aleph \restriction (1 - 3\delta))$$

so we obtain $n = 0$, and so $0 \leqslant t - n < 1$ as required.

Case $t \in (1 - 3\delta, 1)$

This case cannot arise since it contradicts $s \uparrow ((1 - 3\delta), 1) = \langle\rangle$.

Case $t \geqslant 1$

Then we have

$$A(timesucc^{-1}(s \dot{-} 1), timesucc^{-1}(\aleph \dot{-} 1)) \wedge \sigma(s) \subseteq ran(timesucc)$$

and also

$$\langle (t - 1, time.(n - 1)) \rangle \in timesucc^{-1}(s \dot{-} 1)$$

so we obtain $0 \leqslant (t - 1) - (n - 1) < 1$, so $0 \leqslant t - n < 1$ as required.

The required result follows in all cases, so the proof is complete. □

8.2 Time Division Multiplexing

When many processes are sharing a resource, a method is required for allocating it. In *TCSP* we may represent this situation by modelling each process as a numbered process $i : P_i$, the resource by a process *RES* and the allocator as a process *CON* which controls access of the resource by the processes. The construction is given by

$$\mathop{|||}_{i=1}^{n} i : P_i \;{}_X\|_{I \cup R}\; (CON \;{}_I\|_R\; RES)$$

where

$$\begin{aligned} R &= \sigma(RES) \\ X &= \bigcup_{i=1}^{n} \sigma(i : P_i) \\ I &= X \cap R \end{aligned}$$

I represents the interface between the processes P_i and the resource *RES*; it is *CON* which restricts the occurrence of these events, and which thereby restricts the access to RES of the various processes. CON will therefore embody the algorithm by which access to RES is granted to the processes.

One such algorithm is time division multiplexing, where the resource is allocated to each process in turn for a fixed time period. The switch occurs every t, and there are n processes. The specification to be satisfied by a process implementing such a scheduling algorithm is the conjunction of the following requirements:

1. $\forall m \bullet CON$ is responsive and impartial on $m.\Sigma$
2. CON **sat** $\langle (t', i.c) \rangle$ **in** $s \Rightarrow i = \lfloor t'/t \rfloor mod\ n$

We define CON as follows:

$$\begin{aligned} CON &\mathrel{\widehat{=}} \mu X \bullet F(X) \\ F(X) &\mathrel{\widehat{=}} (\mu Y \bullet y : 0.\Sigma \rightarrow Y) \underset{t-3\delta}{↯} succ_n(X) \end{aligned}$$

where

$$\begin{aligned} succ_n(m.a) &\mathrel{\widehat{=}} ((m+1) mod\ \ n).a \\ succ_n(a) &\mathrel{\widehat{=}} a \quad \text{if } a \notin m.\Sigma \text{ for every } m \end{aligned}$$

We have $(succ_n)^n$ is the identity function on Σ, so by considering CON as the fixed point of the function $(F \circ WAIT\ \delta)^n$ we have that

$$\begin{aligned} P_0 &\mathrel{\widehat{=}} (\mu Y \bullet y : 0.\Sigma \rightarrow Y) \\ P_{m+1} &\mathrel{\widehat{=}} P_0 \underset{t-3\delta}{↯} succ_n(WAIT\ \delta\,; P_m) \\ CON &= \mu X \bullet P_{n-1} \underset{nt-3\delta}{↯} X \end{aligned}$$

It may be established by induction on m that

$$0 \leqslant r \leqslant m < n \quad \Rightarrow \quad P_m \text{ is immediately } ((m+1)t - 3\delta, r.\Sigma)\text{-responsive}$$

The base case of this induction is trivial, and the inductive step follows immediately from theorem 5.2.16. Hence for every m between 0 and $n-1$ we have that P_{n-1} is immediately $(nt - 3\delta, m.\Sigma)$-responsive. Hence from theorem 5.2.17 we obtain that CON is responsive on $m.\Sigma$ for each m between 0 and $n-1$.

Impartiality on $m.\Sigma$ is immediate from an application of theorem 5.4.2 on the syntax of CON.

We next prove that CON **sat** $A(s, \aleph)$, where

$$A(s, \aleph) \mathrel{\widehat{=}} \langle (t', i.c) \rangle \text{ in } s \Rightarrow i = \lfloor t'/t \rfloor mod\ n$$

Assuming X **sat** $A(s,\aleph)$ we must find S_1 and S_2 such that

$$\begin{array}{rcl}
\mu\, Y \bullet y : 0.\Sigma \rightarrow Y & \textbf{sat} & S_1 \\
succ_n(\mathit{WAIT}\ \delta\,;X) & \textbf{sat} & S_2 \\
\left.\begin{array}{l} S_1(s \upharpoonright (t-3\delta), \aleph \upharpoonright (t-3\delta)) \wedge s \uparrow (t-3\delta, t-\delta) = \langle\rangle \\ \wedge\ S_2(s \dot{-} (t-\delta), \aleph \dot{-} (t-\delta)) \end{array}\right\} & \Rightarrow & A(s,\aleph)
\end{array}$$

The relevant property that will suffice for S_1 is given by

$$S_1(s,\aleph) \;\widehat{=}\; \langle(t', i.c)\rangle \in s \Rightarrow i = 0$$

and we may use the same approach as in example 8.1 to obtain an expression for S_2, since we have that X **sat** $A(s,\aleph)$:

$$\begin{array}{rcl}
S_2(s,\aleph) & \widehat{=} & A(succ_n^{-1}(s \dot{-} \delta), succ_n^{-1}(\aleph \dot{-} \delta)) \\
& & \wedge\ begin(s) \geqslant \delta \wedge \sigma(s) \subseteq ran(succ_n)
\end{array}$$

The third proof obligation above can easily be established by a case analysis on the time t' of any given $\langle(t', i.c)\rangle$ in s, the cases being $t' \leqslant t - 3\delta$, $t' \in (t - 3\delta, t)$, and $t' \geqslant t$. □

8.3 A Watchdog Timer

The continuing correct operation of a process may be monitored by a process such as a watchdog timer. The timer is set up to expire within t time units, and when functioning correctly, the monitored process should reset tbe timer by means of a *reset* event. If it fails to do so by time t, then the timer withdraws the option of resetting, and raises the alarm within a further time T.

We formalise these requirements as follows:

1. *WTIM* **sat** $(s \upharpoonright reset) \uparrow [t_1, t_1 + t] = \langle\rangle \Rightarrow (s \upharpoonright reset) \upharpoonright t_1 = \langle\rangle$
2. *WTIM* **sat** $\#alarm = 0 \Rightarrow reset \notin \sigma(\aleph \uparrow [end(s) + 2\delta, end(s) + t))$
3. *WTIM* **sat** $s = s'^\frown\langle(t', alarm)\rangle \Rightarrow t' \geqslant end(s') + t$
4. *WTIM* **sat** $\#alarm = 0 \Rightarrow [end(s), end(s) + t + T) \times \{alarm\} \nsubseteq \aleph$
5. *WTIM* is strongly non-retracting on *alarm*

These specify that the timer will not allow a *reset* event if it has not performed one during the last t time units; that it must be prepared to engage in a *reset* before the timer expires; that the alarm cannot go off before the timer expires; that the alarm will always be prepared to go off within $t + T$ of the last reset, i.e. within T of the timer expiring; and that once tbe alarm is enabled, then it will remain enabled until it occurs. For convenience, we abbreviate the behavioural specifications in the first four obligations to W_1, W_2, W_3 and W_4 respectively.

The desired behaviour of the watchdog timer following the alarm will depend very much on the nature of the monitored process. It may be possible to restart the process following a malfunction, in which case we would wish to restart the timer as well. There will be some cases where the alarm triggers some emergency action from which recovery is not possihle, such as ejecting the pilot when the engine fails. In these cases, the timer can terminate after the alarm has been raised.

Our proposed implementation of *WTIM* will terminate after raising the alarm. We define

$$\begin{aligned} WTIM &\mathrel{\widehat=} WAIT\ 2\delta\,;\,TIM \\ TIM &\mathrel{\widehat=} (\mu X \bullet (reset \rightarrow X) \overset{t-2\delta}{\triangleright} alarm \rightarrow SKIP \end{aligned}$$

We first prove our fifth proof obligation, that *WTIM* is strongly non-retracting on *alarm*. From theorem 5.1.8 it is enough to prove that *TIM* is strongly non-retracting on *alarm*. Siuce "strongly non-retracting on *alarm*" is closed we need only prove that it is preserved by

$$F(X) \mathrel{\widehat=} (reset \rightarrow WAIT\ \delta\,;\,X) \overset{t-2\delta}{\triangleright} alarm \rightarrow SKIP$$

If X is strongly non-retracting on *alarm* then $reset \rightarrow WAIT\ \delta\,;\,X$ is also strongly non-retracting on *alarm* and $inits(reset \rightarrow WAIT\ \delta\,;\,X) \cap \{alarm\} = \emptyset$. We also have $alarm \rightarrow SKIP$ is strongly non-retracting on *alarm*, so by theorem 5.1.8 we conclude that $(reset \rightarrow WAIT\ \delta\,;X) \overset{t-2\delta}{\triangleright} alarm \rightarrow SKIP$ is strongly non-retracting on *alarm*, as required. □

In order to prove that $WAIT\ 2\delta\,;\,TIM$ **sat** $W_1(s, \aleph)$, the rule for delay yields that it will he sufficient to prove that

$$\begin{aligned} TIM \quad &\textbf{sat} \quad W_5 \\ W_5 \quad &= \quad W_1(s, \aleph) \wedge ((s \upharpoonright reset) \upharpoonright t - 2\delta = \langle\rangle \Rightarrow s \upharpoonright reset = \langle\rangle) \end{aligned}$$

We prove this by recursion induction. Assuming X **sat** $W_5(s, \aleph)$, we wish to prove that

$$(reset \rightarrow WAIT\ \delta\,;\,X) \overset{t-2\delta}{\triangleright} alarm \rightarrow SKIP \quad \textbf{sat} \quad W_5(s, \aleph)$$

We must find S_1 and S_2 such that

$$\begin{array}{rll} \mathit{reset} \rightarrow \mathit{WAIT}\ \delta\,;X & \textbf{sat} & S_1 \\ \mathit{alarm} \rightarrow \mathit{SKIP} & \textbf{sat} & S_2 \end{array}$$

$$\left.\begin{array}{l} \mathit{begin}(s) \leqslant t - 2\delta \wedge S_1(s,\aleph) \\ \vee \\ \mathit{begin}(s) \geqslant t - \delta \wedge S_1(\langle\rangle, \aleph \restriction t - 2\delta) \\ \wedge\ S_2(s \mathbin{\dot{-}} (t-\delta), \aleph \mathbin{\dot{-}} (t-\delta)) \end{array}\right\} \quad \Rightarrow \quad W_5(s,\aleph)$$

The property of *alarm* → *SKIP* contributing to the correctness of the construct is that it is unable to perform *reset*. Hence we may put

$$S_2(s,\aleph) \ \ \widehat{=}\ \ s \restriction \mathit{reset} = \langle\rangle$$

From the rules for prefixing and delay, we derive a suitable S_1:

$$\begin{array}{rcl} S_1\,(s,\aleph) & = & s = \langle\rangle \vee s = \langle(t', \mathit{reset})\rangle^\frown s' \\ & & \qquad\qquad \wedge\ W_5(s \mathbin{\dot{-}} (t' + 2\delta), \aleph \mathbin{\dot{-}} (t' + 2\delta)) \end{array}$$

Our third proof obligation

$$\left.\begin{array}{l} \mathit{begin}(s) \leqslant t - 2\delta \wedge S_1(s,\aleph) \\ \vee \\ \mathit{begin}(s) \geqslant t - \delta \wedge S_1(\langle\rangle, \aleph \restriction t - 2\delta) \\ \wedge\ S_2(s \mathbin{\dot{-}} (t-\delta), \aleph \mathbin{\dot{-}} (t-\delta)) \end{array}\right\} \quad \Rightarrow \quad W_5(s,\aleph)$$

is now straightforward to establish. □

We now prove that *WAIT 2δ*; *TIM* **sat** $W_2(s,\aleph)$, the rule for delay yields that it will be sufficient to prove that

$$\begin{array}{lcl} \mathit{TIM} & \textbf{sat} & W_6 \\ W_6 & = & s = \langle\rangle \Rightarrow \mathit{reset} \notin \sigma(\aleph \restriction (t - 2\delta)) \\ & & \wedge\ (s \neq \langle\rangle \wedge \#\mathit{alarm} = 0) \Rightarrow \mathit{reset} \notin \sigma(\aleph \uparrow [\mathit{end}(s) + 2\delta, \mathit{end}(s) + t)) \end{array}$$

We prove this by recursion induction. Assuming X **sat** $W_6(s,\aleph)$, we wish to prove that

$$(\mathit{reset} \rightarrow \mathit{WAIT}\ \delta\,;X) \overset{t-2\delta}{\rhd} \mathit{alarm} \rightarrow \mathit{SKIP} \quad \textbf{sat} \quad W_6(s,\aleph)$$

The structure of this proof is identical to that of the previous proof: The property of *alarm* → *SKIP* that will do for S_2 is that the first action it can do must be an alarm action. Hence we set

$$S_2 \ \ \widehat{=}\ \ s \neq \langle\rangle \Rightarrow \#\mathit{alarm} > 0$$

We can see that if W_6 is substituted for S_1 then the third proof obligation may be discharged. So we need only prove that

$$reset \rightarrow WAIT\,\delta\,;X \quad \mathbf{sat} \quad W_6$$

Using the law for prefixing, we need only find S_3 such that

$$\begin{array}{rcl} WAIT\,\delta\,;X & \mathbf{sat} & S_3(s,\aleph) \\ s=\langle\rangle \wedge reset \notin \sigma(\aleph) & \Rightarrow & W_6(s,\aleph) \\ \left.\begin{array}{l} s=\langle(t',reset)\rangle^\frown s' \wedge reset \notin \sigma(\aleph \upharpoonright t') \wedge begin(s') \geqslant t'+\delta \\ \wedge\, S_3(s' \dot{-} (t'+\delta),(\aleph \dot{-} (t'+\delta))) \end{array}\right\} & \Rightarrow & W_6(s,\aleph) \end{array}$$

The second of these obligations is immediately dischargeable. To find S_3, we have that X **sat** $W_6(s,\aleph)$, so from the delay rule we know that

$$WAIT\,\delta\,;X \quad \mathbf{sat} \quad W_6(s \dot{-} \delta, \aleph \dot{-} \delta) \wedge begin(s) \geqslant \delta$$

so we may define

$$S_3(s,\aleph) \;\;\hat{=}\;\; W_6(s \dot{-} \delta, \aleph \dot{-} \delta) \wedge begin(s) \geqslant \delta$$

Hence we have discharged our first proof obligation. The third and final proof obligation is simple to discharge, using elementary set arithmetic. □

The proofs that $WTIM$ **sat** $W_3(s,\aleph)$ and $WTIM$ **sat** $W_4(s,\aleph)$ are identical in structure to the previous two proofs.

To prove that $WTIM$ **sat** $W_3(s,\aleph)$, we reduce the proof obligation, using the rule for delay, to

$$\begin{array}{lcl} TIM & \mathbf{sat} & W_7 \\ W_7 & = & \#s \geqslant 2 \Rightarrow (s = s'^\frown\langle(t',alarm)\rangle \Rightarrow t' \geqslant end(s')+t \\ & & \wedge\, s=\langle(t',alarm)\rangle \Rightarrow t' \geqslant t - 2\delta \end{array}$$

In this case, the instantiations for S_1 and S_2 which are sufficient to establish that the defining equation for TIM preserves W_7 are given by

$$\begin{array}{rcl} S_1(s,\aleph) & = & W_3(s,\aleph) \\ S_2(s,\aleph) & = & \exists\, t,t' \bullet s \leqslant \langle(t,alarm),(t',\checkmark)\rangle \end{array}$$

Under the assumption that X **sat** W_7, it is straightforward to establish that

$$\begin{array}{rcl} reset \rightarrow WAIT\,\delta\,;X & \mathbf{sat} & S_1 \\ alarm \rightarrow SKIP & \mathbf{sat} & S_2 \end{array}$$

$$\left.\begin{array}{l} begin(s) \leqslant t - 2\delta \wedge S_1(s,\aleph) \\ \vee \\ begin(s) \geqslant t - \delta \wedge S_1(\langle\rangle, \aleph \upharpoonright t - 2\delta) \\ \wedge\, S_2(s \dot{-} (t-\delta), \aleph \dot{-} (t-\delta)) \end{array}\right\} \quad \Rightarrow \quad W_7(s,\aleph)$$

and hence that TIM **sat** W_7.

We finally prove that $WTIM$ **sat** $W_4(s, \aleph)$. The proof obligation is first reduced via the rule for delay to

$$\begin{array}{lll} TIM & \textbf{sat} & W_8 \\ W_8 & = & s = \langle\rangle \Rightarrow [end(s), end(s) + t + T - 2\delta) \times \{alarm\} \not\subseteq \aleph \\ & & (s \neq \langle\rangle \wedge \#alarm = 0) \Rightarrow [end(s), end(s) + t + T) \times \{alarm\} \not\subseteq \aleph \end{array}$$

Providing $T > \delta$, it is sufficient to instantiate S_1 and S_2 as follows:

$$\begin{array}{lll} S_1(s, \aleph) & = & W_4(s, \aleph) \\ S_2(s, \aleph) & = & s = \langle\rangle \Rightarrow alarm \notin \sigma(\aleph) \\ & & \wedge\ s \neq \langle\rangle \Rightarrow \#alarm > 0 \end{array}$$

We are then able, under the assumption that X **sat** W_8, to establish that

$$\begin{array}{rll} reset \rightarrow WAIT\ \delta\, ; X & \textbf{sat} & S_1 \\ alarm \rightarrow SKIP & \textbf{sat} & S_2 \end{array}$$

$$\left. \begin{array}{l} begin(s) \leqslant t - 2\delta \wedge S_1(s, \aleph) \\ \vee \\ begin(s) \geqslant t - \delta \wedge S_1(\langle\rangle, \aleph \restriction t - 2\delta) \\ \wedge\ S_2(s \dot{-} (t - \delta), \aleph \dot{-} (t - \delta)) \end{array} \right\} \quad \Rightarrow \quad W_8(s, \aleph)$$

and hence that TIM **sat** W_8, yielding that $WTIM$ **sat** W_4.

8.4 Some Simple Protocols

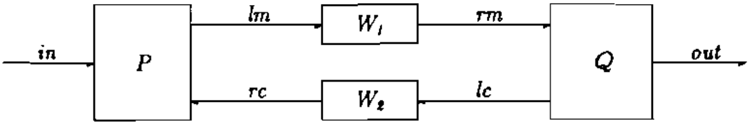
A protocol is a distributed algorithm for facilitating the communication of messages between processes. CSP is particularly suitable for the specification of protocols; the enhancements introduced in Timed CSP allow us to address the timing considerations that are often necessary for the correctness of the protocol.

8.4.1 The Stop and Wait Protocol I

As a more detailed illustration of the application of the proof system for TM_F, we will verify that a stop-and-wait protocol, similar to the one described in [PS88], meets a desirable requirement of communication protocols: that an output will always be available within two time units of an input occurring.

The proof will be performed in two stages. In the first stage we will place conditions upon the sender and receiver, and use the rules to verify that these conditions are sufficient to ensure correctness of the protocol. In the second stage, we will propose a *TCSP* implementation for each of the sender and the receiver, and the rules will be used to prove that these implementations meet the conditions placed upon them in the first stage.

The protocol consists of two processes, P and Q, communicating across two wires: W_1 and W_2. Together, they control the flow of data between two external processes. This may be represented pictorially as follows:



In general, protocols allow for unreliable channels, by duplicating data or requiring acknowledgements: such behaviour is easily modelled in Timed *CSP*. However, our purpose is to illustrate the use of the inference rules; we need not concern ourselves with these complications. Our protocol addresses only dataflow considerations, and we assume that the wires W_1 and W_2 are *reliable*: for every input, there is a corresponding output.

There are many requirements that we could place upon the protocol, but we will consider just one: that if a message is input, then output is ready within two time units. Formally, we wish our protocol *PROT* to meet the following timed failures specification:

$$SPEC(s, \aleph) \;\; \hat{=} \;\; last(s) = in \Rightarrow out \notin \sigma(\aleph \uparrow (end(s) + 2))$$

We give conditions on the components of the protocol, and verify that they are sufficient to ensure that the protocol exhibits this behaviour.

The sending process P should meet the following specification: it should perform the three events in, lm, rc in strict rotation; after performing an event, it should be prepared to perform the next within a certain time; initially, it should be ready to receive an input. We capture these requirements in the timed failures specification $SPEC_P$:

$$\begin{aligned} SPEC_P(s, \aleph) \;\; \hat{=} \;\; & tstrip(s) \leqslant \langle in, lm, rc\rangle^* \wedge \\ & last(s) = in \Rightarrow lm \notin \sigma(\aleph \uparrow (end(s) + 2\delta)) \wedge \\ & last(s) = lm \Rightarrow rc \notin \sigma(\aleph \uparrow (end(s) + 2\delta)) \wedge \\ & last(s) = rc \Rightarrow in \notin \sigma(\aleph \uparrow (end(s) + 2\delta)) \wedge \\ & s = \langle\rangle \Rightarrow in \notin \sigma(\aleph) \end{aligned}$$

After accepting and transmitting a message, the sending process must await confirmation from the receiving process before accepting another. The receiving process will send a confirmation signal once the previous message has been output. Initially, the system is empty. Hence we wish the receiving process Q to satisfy $SPEC_Q$:

$$\begin{aligned} SPEC_Q(s,\aleph) \;\hat{=}\;& tstrip(s) \leqslant \langle rm, out, lc\rangle^* \wedge \\ & last(s) = rm \Rightarrow out \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\ & last(s) = out \Rightarrow lc \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\ & last(s) = lc \Rightarrow rm \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\ & s = \langle\rangle \Rightarrow rm \notin \sigma(\aleph) \end{aligned}$$

The wires W_1 and W_2 have a propagation delay of 1 time unit, and will not be required to transmit more than one message at a time. However, each must be ready to accept another input almost immediately after output. They satisfy the specifications $SPEC_{W_1}$ and $SPEC_{W_2}$ respectively, where

$$\begin{aligned} SPEC_{W_1}(s,\aleph) \;\hat{=}\;& tstrip(s) \leqslant \langle lm, rm\rangle^* \wedge \\ & last(s) = lm \Rightarrow rm \notin \sigma(\aleph \upharpoonleft (end(s) + 1)) \wedge \\ & last(s) = rm \Rightarrow lm \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\ & s = \langle\rangle \Rightarrow lm \notin \sigma(\aleph) \end{aligned}$$

$$\begin{aligned} SPEC_{W_2}(s,\aleph) \;\hat{=}\;& tstrip(s) \leqslant \langle lc, rc\rangle^* \wedge \\ & last(s) = lc \Rightarrow rc \notin \sigma(\aleph \upharpoonleft (end(s) + 1)) \wedge \\ & last(s) = rc \Rightarrow lc \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\ & s = \langle\rangle \Rightarrow lc \notin \sigma(\aleph) \end{aligned}$$

The protocol is a combination of the sending process, the receiving process, and the wires. It can be defined by means of the NET operator described is section 7.4. We write it here in full in order to be able to apply the inference rules. If we define the sets

$$\begin{aligned} X \;&\hat{=}\; \{in, lm, rc\} \\ Y \;&\hat{=}\; \{out, rm, lc\} \\ C \;&\hat{=}\; \{lc, rc\} \\ M \;&\hat{=}\; \{lm, rm\} \\ A \;&\hat{=}\; M \cup C \end{aligned}$$

then the protocol may be defined:

$$PROT \;\hat{=}\; ((P\ {}_X\|_Y\ Q)\ {}_{X\cup Y}\|_{M\cup C}\ (W_1\ {}_M\|_C\ W_2)) \setminus A$$

Having formalised our requirements, we can now use the inference rules to demonstrate that the protocol *PROT* will meet the specification *SPEC*. We wish to establish that:

$$PROT \quad \mathbf{sat} \quad SPEC(s, \aleph)$$

The definition of *PROT* involves the hiding operator at the outermost level, so we must first apply the hiding rule. This reduces the proof requirement to:

$$\begin{array}{lll}(P \,{}_X\|_Y\, Q)\,{}_{X \cup Y}\|_{M \cup C}\,(W_1 \,{}_M\|_C\, W_2) & \mathbf{sat} & \sigma(\aleph') \subseteq A \wedge ([0, end(s, \aleph)) \times A) \subseteq \aleph \\ & & \Rightarrow SPEC(s \setminus A, \aleph - \aleph')\end{array}$$

This is a proof requirement on a parallel combination, so we apply the rule for the parallel operator. We have then to find specifications S_1 and S_2 such that:

$$\begin{array}{lll}P \,{}_X\|_Y\, Q & \mathbf{sat} & S_1(s, \aleph) \\ W_1 \,{}_M\|_C\, W_2 & \mathbf{sat} & S_2(s, \aleph)\end{array}$$

$$\left.\begin{array}{l}\sigma(s_1, \aleph_1) \subseteq (X \cup Y) \wedge \sigma(s_2, \aleph_2) \subseteq (M \cup C) \\ \sigma(\aleph_3) \subseteq \Sigma - (X \cup Y \cup M \cup C) \\ S_1(s_1, \aleph_1) \wedge S_2(s_2, \aleph_2) \wedge s_3 \in s_1 \,{}_{X \cup Y}\|_{M \cup C}\, s_2 \\ \aleph = \aleph_1 \cup \aleph_2 \cup \aleph_3 \\ \sigma(\aleph') \subseteq A \wedge ([0, end(s_3, \aleph)) \times A) \subseteq \aleph\end{array}\right\} \Rightarrow SPEC(s_3 \setminus A, \aleph - \aleph')$$

Before we continue, we note that the specification *SPEC* is independent of the hidden set of events A, for consider the definition:

$$SPEC \;\triangleq\; last(s) = in \Rightarrow out \notin \sigma(\aleph \upharpoonleft (end(s) + 2))$$

Formally, we can show that

$$SPEC(s, \aleph \upharpoonright (\Sigma - A)) \;\Rightarrow\; SPEC(s, \aleph)$$

This concurs with our intuition: the correctness of the protocol may be dependent upon hidden interactions, but our formal description of the service provided (the specification *SPEC*) should abstract away from internal detail.

Taking this in conjunction with the alphabet conditions upon the failure sets, we may reduce the third proof obligation to

$$\left.\begin{array}{l}\sigma(s_1, \aleph_1) \subseteq (X \cup Y) \wedge \sigma(s_2, \aleph_2) \subseteq (M \cup C) \\ \sigma(\aleph_3) \subseteq \Sigma - (X \cup Y) \\ S_1(s_1, \aleph_1) \wedge S_2(s_2, \aleph_2) \wedge s_3 \in s_1 \,{}_{X \cup Y}\|_{M \cup C}\, s_2 \\ ([0, end(s_3, \aleph_1 \cup \aleph_2 \cup \aleph_3)) \times A) \subseteq \aleph_1 \cup \aleph_2\end{array}\right\} \Rightarrow SPEC(s_3 \setminus A, \aleph_1)$$

To identify S_1 we apply the parallel rule once again. We are then required to find S_4 and S_5 such that:

$$\begin{array}{l}
P \quad \mathbf{sat} \quad S_4(s,\aleph) \\
Q \quad \mathbf{sat} \quad S_5(s,\aleph)
\end{array}$$

$$\left.\begin{array}{l}
\sigma(s_4,\aleph_4) \subseteq X \wedge \sigma(s_5,\aleph_5) \subseteq Y \\
\sigma(\aleph_6) \subseteq \Sigma - (X \cup Y) \\
S_4(s_4,\aleph_4) \wedge S_5(s_5,\aleph_5) \wedge s_6 \in s_4 \; {}_X\|_Y \; s_5
\end{array}\right\} \Rightarrow S_1(s_6, \aleph_4 \cup \aleph_5 \cup \aleph_6)$$

We already have specifications for the components P and Q. Substituting these for S_4 and S_5, and using the alphabet conditions upon the traces and refusals, we can reduce this proof obligation to:

$$\left.\begin{array}{l}
SPEC_P(s \upharpoonright X, \aleph \upharpoonright X) \\
SPEC_Q(s \upharpoonright Y, \aleph \upharpoonright Y) \\
\sigma(s) \subseteq (X \cup Y)
\end{array}\right\} \Rightarrow S_1(s,\aleph)$$

This yields a suitable instantiation for S_1: the antecedent of the above expression. In a similar fashion, we arrive at the following instantiation for S_2:

$$\begin{array}{l}
SPEC_{W_1}(s \upharpoonright M, \aleph \upharpoonright M) \wedge \\
SPEC_{W_2}(s \upharpoonright C, \aleph \upharpoonright C) \wedge \\
\sigma(s) \subseteq (M \cup C)
\end{array}$$

Our proof requirement can then be written as follows:

$$\left.\begin{array}{l}
\sigma(s_1,\aleph_1) \subseteq (X \cup Y) \wedge \sigma(s_2,\aleph_2) \subseteq (M \cup C) \\
\sigma(\aleph_3) \subseteq \Sigma - (X \cup Y) \\
SPEC_P(s_1 \upharpoonright X, \aleph_1 \upharpoonright X) \wedge SPEC_Q(s_1 \upharpoonright Y, \aleph_1 \upharpoonright Y) \\
SPEC_{W_1}(s_2 \upharpoonright M, \aleph_2 \upharpoonright M) \wedge SPEC_{W_2}(s_2 \upharpoonright C, \aleph_2 \upharpoonright C) \\
([0, end(s_3, \aleph_1 \cup \aleph_2 \cup \aleph_3)) \times A) \subseteq \aleph_1 \cup \aleph_2 \\
s_3 \in s_1 \; {}_{X \cup Y}\|_{M \cup C} \; s_2
\end{array}\right\} \Rightarrow SPEC(s_3 \setminus A, \aleph_1)$$

The alphabet conditions in S_1 and S_2 are subsumed in the first two conditions above.

We have reduced the proof obligation to a predicate on traces and refusal sets: the verification may be completed using simple properties of sets and sequences: assuming the conjuncts in the above antecedeut, we are trying to establish that

$$last(s_3 \setminus A) = in \Rightarrow out \notin \sigma(\aleph_1 \uparrow (end(s_3 \setminus A) + 2))$$

From $SPEC_P$, $SPEC_Q$, $SPEC_{W_1}$, $SPEC_{W_2}$, and the properties of sequences, we can deduce that

$$s_3 \leqslant \langle in, lm, rm, out, lc, rc\rangle^*$$

We then proceed by case analysis on the identity of the last event in s_3, given that $last(s_3 \setminus A) = in$, there are three possibilities. **Case:** $last(s_3) = in$

By $SPEC_P$,	$lm \notin \sigma((\aleph_1 \upharpoonright X) \upharpoonleft (end(s_1 \upharpoonright X) + 2\delta))$
In this case	$end(s_1) = end(s_1 \upharpoonright X)$
and we know that	$lm \notin Y$
Hence	$lm \notin \sigma(\aleph_1 \upharpoonleft (end(s_3) + 2\delta))$
Similarly, as	$s_3 \in s_1 \; {}_{X \cup Y}\|_{M \cup C} \; s_2$,
$SPEC_{W_1}$ implies that	$lm \notin \sigma(\aleph_2 \upharpoonleft (end(s_3) + 2\delta))$
Hence	$lm \notin \sigma((\aleph_1 \cup \aleph_2 \cup \aleph_3) \upharpoonleft (end(s_3) + 2\delta))$
However,	$([0, end(s, \aleph_1 \cup \aleph_2 \cup \aleph_3)) \times A) \subseteq (\aleph_1 \cup \aleph_2 \cup \aleph_3)$
and	$lm \in A$
So	$end(\aleph_1 \cup \aleph_2 \cup \aleph_3) \leqslant end(s_3) + 2\delta$
But $\delta \ll 1$, so	$(\aleph_1 \cup \aleph_2 \cup \aleph_3) \upharpoonleft (end(s_3) + 2)) = \{\}$
We conclude that	$out \notin \sigma((\aleph_1 \cup \aleph_2 \cup \aleph_3) \upharpoonleft end(s_3 \setminus A + 2))$

Case: $last(s_3) = lm$

We establish that $end(s_3) \leqslant end(s_3 \setminus A) + 2\delta$: that the lm event occurred within time 2δ of the last input.

Assume otherwise:	$end(s_3) > end(s_3 \setminus A) + 2\delta$
If we let t be the time	$(end(s_3 \setminus A) + end(s_3) + 2\delta)/2$
Then we know that	$last(s_3 \upharpoonright t) = in$
By the previous **case**	$lm \notin \sigma(((\aleph_1 \cup \aleph_2 \cup \aleph_3) \upharpoonright t) \upharpoonleft (end(s_3 \upharpoonright t) + 2\delta))$
From our assumptions	$([0, end(s_3, \aleph_1 \cup \aleph_2 \cup \aleph_3)) \times A) \subseteq \aleph_1 \cup \aleph_2$
And	$end(s_3 \upharpoonright t) + 2\delta = end(s_3 \setminus A) + 2\delta < t$
Hence	$lm \in \sigma(((\aleph_1 \cup \aleph_2 \cup \aleph_3) \upharpoonright t) \upharpoonleft (end(s_3 \upharpoonright t) + 2\delta))$

Forcing a contradiction.

We can show, with a similar argument to the first **case**, in which the event rm replaces lm, that $end(\aleph_1 \cup \aleph_2 \cup \aleph_3) \leqslant end(s_3) + 1$. From above, $end(s_3) \leqslant end(s_3 \setminus A) + 2\delta$: the result follows.

Case: $last(s_3) = rm$

By a similar argument, we can establish that the event rm must occur no later than $1 + 2\delta$ after the last input. We then appeal to the specification of Q, and the result follows immediately. □

The treatment of hiding in Timed *CSP* is central to the construction of the above proof; the hidden events lm and rm must occur as soon as possible. Our method of proof allowed us to include these events in our reasoning, by eliminating the hiding operator from our proof obligation.

Only at the final stage of the proof did we identify the protocol requirement *SPEC*. To establish that another property holds of the above protocol, it would not be necessary to perform the whole proof again. We have characterised the behaviour of the protocol in terms of the known properties of its components. To prove that the protocol satisfies an arbitrary specification S, we have only to show that the following predicate is true:

$$\left.\begin{array}{l} \sigma(s_1, \aleph_1) \subseteq (X \cup Y) \wedge \sigma(s_2, \aleph_2) \subseteq (M \cup C) \\ \sigma(\aleph_3) \subsetneq \Sigma - (X \cup Y) \\ SPEC_P(s_1 \upharpoonright X, \aleph_1 \upharpoonright X) \wedge SPEC_Q(s_1 \upharpoonright Y, \aleph_1 \upharpoonright Y) \\ SPEC_{W_1}(s_2 \upharpoonright M, \aleph_2 \upharpoonright M) \wedge SPEC_{W_2}(s_2 \upharpoonright C, \aleph_2 \upharpoonright C) \\ \aleph = \aleph_1 \cup \aleph_2 \cup \aleph_3 \wedge s_3 \in s_1 \; {}_{X \cup Y}\|_{M \cup C} \; s_2 \\ \sigma(\aleph') \subseteq A \wedge ([0, end(s_3, \aleph)) \times A) \subsetneq \aleph \end{array}\right\} \Rightarrow S(s_3 \setminus A, \aleph - \aleph')$$

For a particular specification S, we will be able to discard most of the conditions in the antecedent: the residual proof requirement is often easy to discharge.

We now move on to the second stage in our verification of the protocol. We propose *TCSP* implementations of tbe components, and use the inference rules to demonstrate that they meet the appropriate specifications.

The protocol consists of two components, transmitter P and receiver Q, communicating across two wires W_1 and W_2. The transmitter process should accept an input on channel in, and he prepared to transmit it along W_1, via channel lm. After this transmission has occurred, P waits for a confirmation event from wire W_2, on channel rc, before repeating this behaviour. Our intuition suggests the following as an implementation:

$$P \;\widehat{=}\; \mu X \bullet in \rightarrow lm \rightarrow rc \rightarrow X$$

We have yet to establish that this implements our requirements: that it meets the formal specification $SPEC_P$.

A similar set of conditions applies to the receiving process Q. It should be prepared to receive a signal from wire W_1, on channel rm, before offering output on channel *out*. It should then send a confirmation signal along wire W_2, on channel *lc*, before returning to its initial state. Our proposed solution:

$$Q \;\widehat{=}\; \mu\, Y \bullet rm \rightarrow out \rightarrow lc \rightarrow Y$$

Again, we will have to verify that this is an implementation of the specification $SPEC_Q$.

We wish to show that the transmitting process P meets the specification placed upon it:

$$\mu\, X \bullet in \rightarrow lm \rightarrow rc \rightarrow X \quad \mathbf{sat} \quad SPEC_P(s, \aleph)$$

This is a recursive process; the second recursion rule requires us to find a specification $S(s, \aleph)$ such that:

$$\begin{aligned} X \ \mathbf{sat}\ S(s,\aleph) \;&\Rightarrow\; in \rightarrow lm \rightarrow rc \rightarrow (\mathit{WAIT}\,\delta\,; X)\ \mathbf{sat}\ S(s,\aleph) \\ S(s,\aleph) \;&\Rightarrow\; SPEC_P(s,\aleph) \end{aligned}$$

We will show that the specification $SPEC_P$ is strong enough to be preserved by the recursion. We have to show that:

$$X\ \mathrm{sat}\ SPEC_P(s,\aleph) \;\Rightarrow\; in \rightarrow lm \rightarrow rc \rightarrow (\mathit{WAIT}\,\delta\,; X)\ \mathbf{sat}\ SPEC_P(s,\aleph)$$

Assume that X **sat** $SPEC_P(s, \aleph)$. We wish to establish that:

$$in \rightarrow lm \rightarrow rc \rightarrow (\mathit{WAIT}\,\delta\,; X) \quad \mathbf{sat} \quad SPEC_P(s,\aleph)$$

Applying the prefix rule three times transforms this proof obligation to the following requirement: we must find a specification $U(s, \aleph)$ such that:

$$\mathit{WAIT}\,\delta\,; X\ \mathbf{sat}\ U(s,\aleph)$$

$$\left.\begin{array}{l}
s = \langle\rangle \wedge in \notin \sigma(\aleph) \\
\vee \\
s = \langle(t, in)\rangle^\frown s' \wedge in \notin \sigma(\aleph \upharpoonright t_1) \wedge \\
\quad s' \dot{-} (t_1 + \delta) = \langle\rangle \wedge lm \notin \sigma(\aleph \dot{-} (t_1 + \delta)) \\
\quad \vee \\
\quad s' \dot{-} (t_1 + \delta) = \langle(t_2, lm)\rangle^\frown s'' \wedge lm \notin \sigma(\aleph \dot{-} (t_1 + \delta) \upharpoonright t_2) \wedge \\
\qquad s'' \dot{-} (t_2 + \delta) = \langle\rangle \wedge rc \notin \sigma(\aleph \dot{-} (t_1 + t_2 + 2\delta)) \\
\qquad \vee \\
\qquad s'' \dot{-} (t_2 + \delta) = \langle(t_3, rc)\rangle^\frown s''' \wedge \\
\qquad\quad rc \notin \sigma(\aleph \dot{-} (t_1 + t_2 + 2\delta) \upharpoonright t_3) \wedge \\
\qquad\quad U(s''' \dot{-} (t_1 + t_2 + t_3 + 3\delta), \aleph \dot{-} (t_1 + t_2 + t_3 + 3\delta))
\end{array}\right\} \Rightarrow SPEC_P(s,\aleph)$$

With a suitable choice of τ_1, τ_2, τ_3, this can be transformed to:

$$WAIT\ \delta\,;X\ \mathbf{sat}\ \ U(s,\aleph)$$

$$\left.\begin{array}{l}
s = \langle\rangle \wedge in \notin \sigma(\aleph) \\
\vee \\
s = \langle(\tau_1, in)\rangle \wedge \in \notin \sigma(\aleph \upharpoonright \tau_1) \wedge lm \notin \sigma(\aleph \upharpoonleft \tau_1 + \delta) \\
\vee \\
s = \langle(\tau_1, in), (\tau_2, lm)\rangle \wedge in \notin \sigma(\aleph \upharpoonright \tau_1) \\
\qquad\qquad\qquad\qquad\quad \wedge\ lm \notin \sigma(\aleph \uparrow [\tau_1 + \delta, \tau_2)) \\
\qquad\qquad\qquad\qquad\quad \wedge\ rc \notin \sigma(\aleph \upharpoonleft \tau_2 + \delta) \\
\vee \\
s = \langle(\tau_1, in), (\tau_2, lm), (\tau_3, rc)\rangle ^\frown u \wedge in \notin \sigma(\aleph \upharpoonright \tau_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ lm \notin \sigma(\aleph \uparrow [\tau_1 + \delta, \tau_2) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ rc \notin \sigma(\aleph \uparrow [\tau_2 + \delta, \tau_3)) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ U(u \mathbin{\dot{-}} (\tau_3 + \delta), \aleph \mathbin{\dot{-}} (\tau_3 + \delta))
\end{array}\right\} \Rightarrow SPEC_P(s, \aleph)$$

Applying the second form of tbe delay rule, we can instantiate U as follows:

$$U(s,\aleph) \ \ \equiv\ \ SPEC_P(s \mathbin{\dot{-}} \delta, \aleph \mathbin{\dot{-}} \delta) \wedge begin(s) \geqslant \delta$$

Having discharged the first proof obligation, the proof can be completed with a simple case analysis on trace s. This becomes clear when we recall the form of specification $SPEC_P$:

$$\begin{array}{rcl}
SPEC_P & \widehat{=} & tstrip(s) \leqslant \langle in, lm, rc\rangle^* \wedge \\
& & last(s) = in \Rightarrow lm \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\
& & last(s) = lm \Rightarrow rc \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\
& & last(s) = rc \Rightarrow in \notin \sigma(\aleph \upharpoonleft (end(s) + 2\delta)) \wedge \\
& & s = \langle\rangle \Rightarrow in \notin \sigma(\aleph)
\end{array}$$

The only non-trivial case corresponds to $s = \langle(\tau_1, in), (\tau_2, lm), (\tau_3, rc)\rangle^\frown u$. Here we require two arguments, one for each of the cases: $u = \langle\rangle$, $u \neq \langle\rangle$. Expanding the specification $SPEC_P$ makes the solution obvious.

This completes the verification of our transmitter process P. It will not be necessary to perform a similar proof for the receiver Q; we can exploit tbe symmetry present in our descriptions.

The operator f in $TCSP$ allows us to relabel the events performed by a process. In the case of injective functions, this allows us to re-use a process description. By renaming events, we can transform processes while retaining their structure. The relationships between different events are maintained: given that a particular result holds for all the behaviours of a process, we can infer a corresponding result about the behaviours of the image of that process under such a transformation:

$$\frac{P\ \mathbf{sat}\ S_1(s,\aleph) \qquad S_1(s,\aleph) \Rightarrow S(f(s),f(\aleph))}{f(P)\ \mathbf{sat}\ S(s,\aleph)}$$

For example, we can use the result of the previous section to establish that Q **sat** $SPEC_Q$, by defining injective function f such that:

$$\begin{aligned} f(in) &\mathrel{\widehat{=}} rm \\ f(lm) &\mathrel{\widehat{=}} out \\ f(rc) &\mathrel{\widehat{=}} lc \end{aligned}$$

We then observe that:

$$\begin{aligned} SPEC_P(s,\aleph) &= SPEC_Q(f(s),f(\aleph)) \\ Q &= f(P) \end{aligned}$$

The inference rule allows us to conclude that:

$$Q \quad \mathbf{sat} \quad SPEC_Q(s,\aleph)$$

Which completes our verification of the protocol.

8.4.2 The Stop and Wait Protocol II

A shorter verification of the stop and wait protocol can be achieved by the application of the general specifications introduced in chapter 5 used in conjunction with timewise refinement. We have the following definitions:

$$\begin{aligned} S &\mathrel{\widehat{=}} \mu X \bullet (in?x \rightarrow lm!x \rightarrow rc?y \rightarrow X) \\ A_S &\mathrel{\widehat{=}} in \cup lm \cup rc \\ R &\mathrel{\widehat{=}} \mu X \bullet (rm?x \rightarrow out!x \rightarrow lc!x \rightarrow X) \\ A_R &\mathrel{\widehat{=}} rm \cup out \cup lc \end{aligned}$$

We also have specifications on the media M_1 and M_2: M_1 is prompt and responsive on lm and on rm, and M_1 **sat** $rm \leqslant_1 lm$; and M_2 is prompt and responsive on lc and on rc, and M_2 **sat** $rc \leqslant_1 lc$. We further define $A_{M_1} \mathrel{\widehat{=}} \{lm, rm\}$, and $A_{M_2} \mathrel{\widehat{=}} \{lc, rc\}$. Then setting $I \mathrel{\widehat{=}} \{S, R, M_1, M_2\}$ we may define $SAWP$ as follows:

$$SAWP \mathrel{\widehat{=}} NET(\{(P, A_P) \mid P \in I\})$$

Neither of the process definitions for S and R involve either hiding or indexed non-deterministic choice, so by corollary 6.3.24 we have that $\Theta(S) \sqsubseteq_f S$, and $\Theta(R) \sqsubseteq_f R$. It follows from the specifications of M_1 and M_2 that $COPY \sqsubseteq_t M_1$, $COPY \sqsubseteq_t M_2$, $COPY \sqsubseteq_f M_1$, and $COPY \sqsubseteq_f M_2$. Further, the set $\{A_i\}$ is triple disjoint, every $P \in I$ is prompt, S is non-retracting on both *lm* and *rc*, and R is non-retracting on both *rm* and *lc*.

We define $R_S \mathrel{\hat=} \Theta(S)$, $R_R \mathrel{\hat=} \Theta(R)$, $R_{M_1} \mathrel{\hat=} COPY[lm, rm/in, out]$ and $R_{M_2} \mathrel{\hat=} COPY[lc, rc/in, out]$. It is proved in [PS88] that any trace of $PAR(\{(R_i, A_i)\})$ has

$$rc \leqslant lc \leqslant_1 out \leqslant_1 rm \leqslant lm \leqslant_1 in \quad \text{and} \quad rc \leqslant_1 in$$

so the channels in the trace appear in the cyclic sequence $\langle in, lm, rm, out, lc, rc\rangle^*$. Hence $PAR(\{(R_i, A_i)\})$ is limited on $lm \cup rm \cup lc \cup rc$, which is the same as $\bigcup\{A_i \cap A_j \mid i, j \in I, i \neq j\}$.

Thus we apply theorem 7.4.5 to obtain

$$NET(\{(R_i, A_i)\}) \;\; \sqsubseteq_f \;\; NET(\{(P, A_P)\})$$

It was proved in [PS88] that

$$\Theta(NET(\{(P, A_P)\})) \quad \mathbf{sat} \quad out \leqslant_1 in$$

so from theorem 6.2.12 it follows that

$$NET(\{(P, A_P)\}) \quad \mathbf{sat} \quad out \leqslant_1 in$$

Standard algebraic techniques may be used to establish that $\Theta(NET(\{(R_i, A_i)\}))$ is deadlock-free, so $NET(\{(P, A_P)\})$ is deadlock-free; it is also a pipe, impartial on *in* (immediate from the syntax), and satisfies $out \leqslant_1 in$, and so from theorem 7.2.5 we conclude that it is a one-place buffer, thus verifying the protocol.

8.4.3 The Alternating Bit Protocol

An untimed alternating bit protocol is presented and verified in [PS88]. In the absence of timing information, the timeout required by the sender of the alternating bit protocol is modelled as a timeout *event*, '$\bigcirc$', which may non-deterministically occur. Using Timed CSP, we may refine the original description to model the timeout explicitly while retaining the safety property proved of the original description.

The untimed description is as follows:

$$\begin{aligned}
S &\triangleq S_0 \\
S_b &\triangleq in?x \rightarrow lm!x.b \rightarrow S_{x.b} \quad \text{where } b \in \{0,1\} \\
S_{x.b} &\triangleq ((rc?a \rightarrow (\textbf{ if } a = b \textbf{ then } S_{1-b} \textbf{ else } S_{x.b})) \\
&\quad \Box \bigcirc \rightarrow lm!x.b \rightarrow S_{x.b}) \setminus \{\bigcirc\}
\end{aligned}$$

$$\begin{aligned}
R &\triangleq R_1 \\
R_b &\triangleq rm?x.c \rightarrow (\textbf{ if } c \neq b \textbf{ then } out!x \rightarrow lc!c \rightarrow R_c \\
&\qquad\qquad\qquad\qquad \textbf{else } lc!b \rightarrow R_b) \quad \text{where } b \in \{0,1\}
\end{aligned}$$

The conditions assumed to hold of the wires *M1* and *M2* are the following:

$$\begin{aligned}
M1 &\quad \textbf{sat} \quad lm \trianglelefteq rm \\
M2 &\quad \textbf{sat} \quad lc \trianglelefteq rc
\end{aligned}$$

where $s_1 \trianglelefteq s_2$ means that s_1 is a (not necessarily contiguous) subsequence of s_2. These requirements allow the media *M1* and *M2* to drop messages, but not to duplicate or reorder them. The associated alphabets of the processes are the following:

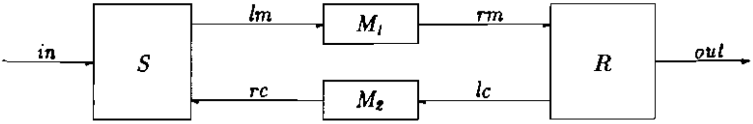
$$\begin{aligned}
A_S &\triangleq \{in, lm, rc\} \\
A_R &\triangleq \{out, rm, lc\} \\
A_{M_1} &\triangleq \{lm, rm\} \\
A_{M_2} &\triangleq \{lc, rc\}
\end{aligned}$$

Then setting $I = \{S, R, M_1, M_2\}$ we may define *ABP* as follows:

$$ABP \triangleq NET(\{(P, A_P) \mid P \in I\})$$

The result proved in M_T is that *ABP* **sat** $out \leqslant_1 in$.

Pictorially, the *ABP* is entirely similar to the *SAWP*:



We refine the sender by making the timeout explicit. Our timed sender *TS* will timeout at time t. The timed receiver will have the same description as the original receiver.

$$\begin{array}{rcl}
TS & \mathrel{\widehat{=}} & TS_0 \\
TS_b & \mathrel{\widehat{=}} & in?x \rightarrow lm!x.b \rightarrow TS_{x.b} \quad \text{where } b \in \{0, 1\} \\
TS_{x.b} & \mathrel{\widehat{=}} & ((rc?a \rightarrow (\ \mathbf{if}\ a = b\ \mathbf{then}\ TS_{1-b}\ \mathbf{else}\ TS_{x.b})) \\
 & & \overset{t}{\triangleright} lm!x.b \rightarrow TS_{x.b})
\end{array}$$

$$\begin{array}{rcl}
TR & \mathrel{\widehat{=}} & TR_1 \\
TR_b & \mathrel{\widehat{=}} & rm?x.c \rightarrow (\ \mathbf{if}\ c \neq b\ \mathbf{then}\ out!x \rightarrow lc!c \rightarrow R_c \\
 & & \qquad\qquad\qquad\qquad \mathbf{else}\ lc!b \rightarrow TR_b) \quad \text{where } b \in \{0, 1\}
\end{array}$$

Recalling from [Hoa85] the law (for the untimed models)

$$((a \rightarrow P) \Box b \rightarrow Q) \setminus b \;=\; ((a \rightarrow (P \setminus b)) \Box Q \setminus b) \sqcap Q \setminus b$$

we obtain that $\Theta(TS) = S$. We also have that $\Theta(TR) = R$. Hence we obtain from corollary 6.2.8 that $S \sqsubseteq_t TS$ and $R \sqsubseteq_t TR$. Then for timed media *TM1* and *TM2* we may define the timed alternating bit protocol as follows:

$$TABP \;\mathrel{\widehat{=}}\; NET(\{(P, A_P) \mid P \in I\})$$

where $I \mathrel{\widehat{=}} \{TS, TR, TM1, TM2\}$, and the A_P are the same sets as for the corresponding untimed processes.

Now if $sr(X\ \mathbf{sat}\ lm \trianglelefteq rm)(TM1)$ and $sr(X\ \mathbf{sat}\ lc \trianglelefteq rc)(TM2)$, then

$$\exists\, M1 \sqsubseteq_t TM1 \bullet M1\ \mathbf{sat}\ lm \trianglelefteq rm \;\wedge\; \exists\, M2 \sqsubseteq_t TM2 \bullet M2\ \mathbf{sat}\ lc \trianglelefteq rc$$

Hence there is an ABP such that $ABP \sqsubseteq_t TABP$ and $ABP\ \mathbf{sat}\ out \leqslant_1 in$, so we obtain that $sr(X\ \mathbf{sat}\ out \leqslant_1 in)(TABP)$, which is written

$$TABP\ \mathbf{sat}\ out \leqslant_1 in$$

Hence we have a verified alternating bit protocol over wires *TM1* and *TM2* for which the above conditions hold. Written another way, the requirements of the wires are

$$\begin{array}{lll}
TM1 & \mathbf{sat} & lm \trianglelefteq rm \\
TM2 & \mathbf{sat} & lc \trianglelefteq rc
\end{array}$$

and we have a verification that the alternating bit protocol satisfies the safety property that the output stream is a prefix of the input stream, of length at least one less that the length of the input stream.

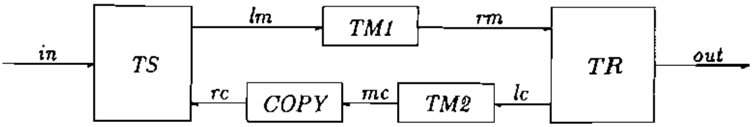
In order to obtain that the protocol is live, in the sense that any input will eventually be offered as output, we require more stringent conditions on the system. We first require that the media be live on their iuput channels: that *TM1* is responsive on *lm*, and *TM2* is respousive on *lc*. We will also require a progress property of the media, that they will not lose messages for ever. Since we are presently unable adequately to model fairness withiu the Timed *CSP* framework, we place the requirement on *TM1* and *TM2* that they cannot input more than N messages without offering one for output. These can be captured as behavioural specifications:

$$\begin{aligned} \mathit{TM1} \quad \mathbf{sat} \quad & (s = u^\frown w \wedge w \restriction rm = \langle\rangle \wedge \# w \restriction lm \geqslant N) \\ & \qquad \Rightarrow [\mathit{begin}(w), \mathit{end}(w)) \times rm \not\subseteq \aleph \\ \mathit{TM2} \quad \mathbf{sat} \quad & (s = u^\frown w \wedge w \restriction rc = \langle\rangle \wedge \# w \restriction lc \geqslant N) \\ & \qquad \Rightarrow [\mathit{begin}(w), \mathit{end}(w)) \times rc \not\subseteq \aleph \end{aligned}$$

A further modification to the system is necessary. Since *TS* is not non-retracting on *rc*, and since we do not insist that *TM2* is non-retracting on *rc*, a generalisation of corollary 7.3.12 indicates that we will need to insert *COPY* along *rc* in order to ensure eventual synchronisation. We will call the modified system *TABP2*.

Liveness of the *ABP* is not guaranteed in M_F, since the timeout event may always occur (non-deterministically), and no message acknowledgement received as a result. Liveness here requires a fairness assumption, that eventually an acknowledgement will be received by the sender. In Timed *CSP*, our explicit modelling of the timeout removes the need for this assumption. However, the timewise refinement approach to prove liveness is not open to us, since there is no corresponding untimed result. We therefore use the proof system to establish mechanically and laboriously that *TABP2* is respousive.

The alternating bit protocol *TABP2* may be described pictorially as follows:



We make the following new definitions, and redefine A_{M_2}

$$C \;\widehat{=}\; \mathit{COPY}[mc, rc/in, out]$$

$$
\begin{aligned}
A_C &\mathrel{\widehat{=}} \{mc, rc\} \\
A_{M2} &\mathrel{\widehat{=}} \{lc, mc\}
\end{aligned}
$$

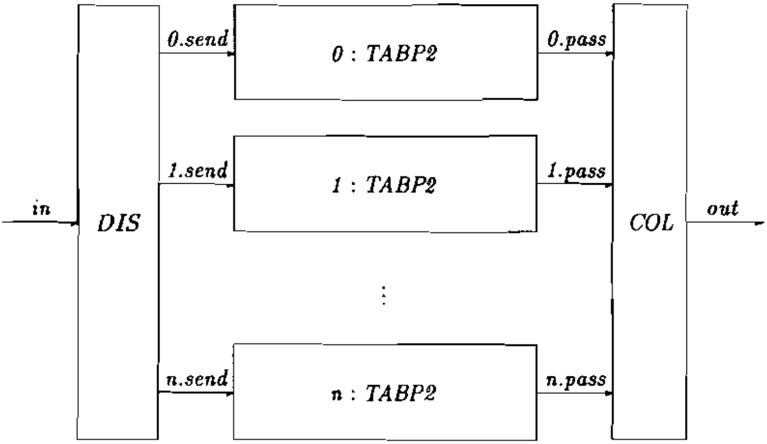
We then define

$$TABP2 \mathrel{\widehat{=}} NET(\{(P, A_P) \mid P \in \{S, R, M1, M2, C\}\})$$

TABP2 is clearly impartial on *in*, and since it is responsive on $in \cup out$, and we have established that it satisfies $out \leqslant_l in$, it follows from theorem 7.2.4 that it is a buffer, verifying the protocol. It follows as a refinement of *COPY*, state that $COPY \sqsubseteq_f TABP2$. It also follows from theorem 5.1.13 that *TABP2* is non-retracting on both *in* and *out*.

8.4.4 The Sliding Window Protocol

In [PS88] it is shown how a sliding window protocol may be considered as *w* alternating bit protocols working in parallel, controlled by a message distributer and a message collator. Diagrammatically, the structure of the sliding window protocol is as follows:



This is defined in the process algebra as follows:

$$\begin{aligned}
Q_i &\mathrel{\widehat{=}} i : (\mathit{TABP2}[in, out/i.send, i.pass]) \\
A_{Q_i} &\mathrel{\widehat{=}} i.send \cup i.pass \\[1ex]
\mathit{DIS} &\mathrel{\widehat{=}} D_0 \\
D_h &\mathrel{\widehat{=}} in?x \rightarrow h.send!x \rightarrow D_{h \oplus_\omega 1} \\
A_{\mathit{DIS}} &\mathrel{\widehat{=}} in \cup \bigcup_{i=0}^{n} i.send \\[1ex]
\mathit{COL} &\mathrel{\widehat{=}} \mathit{COL}_0 \\
C_l &\mathrel{\widehat{=}} l.pass?x \rightarrow out!x \rightarrow C_{l \oplus_\omega 1} \\
A_{\mathit{COL}} &\mathrel{\widehat{=}} out \cup \bigcup_{i=0}^{n} i.pass \\[1ex]
\mathit{SWP} &\mathrel{\widehat{=}} \mathit{NET}\{(Q, A_Q) \mid Q \in \{Q_i \mid 0 \leqslant i \leqslant n\} \cup \{\mathit{COL}, \mathit{DIS}\}\}
\end{aligned}$$

We have from the previous section tbat $\mathit{COPY} \sqsubseteq_f \mathit{TABP2}$, and that $\mathit{TABP2}$ is non-retracting on both *in* and *out*, so

$$i : (\mathit{COPY}[in, out/send, pass]) \;\sqsubseteq_f\; i : (\mathit{TABP2}[in, out/send, pass])$$

for any i, and that $i : (\mathit{TABP2}[in, out/seud, pass])$ is non-retracting on both $i.send$ and $i.receive$. It follows from corollary 6.3.24 that

$$\begin{aligned}
\Theta(\mathit{COL}) &\sqsubseteq_f \mathit{COL} \\
\Theta(\mathit{DIS}) &\sqsubseteq_f \mathit{DIS}
\end{aligned}$$

and we also have from theorem 5.3.6 that *COL* and *DIS* are both prompt. Defining

$$\begin{aligned}
P_i &\mathrel{\widehat{=}} i : (\mathit{COPY}[in, out/send, pass]) \\
A_{P_i} &\mathrel{\widehat{=}} A_{Q_i}
\end{aligned}$$

we define an untimed sliding window protocol *SWP* by

$$\mathit{SWP} \mathrel{\widehat{=}} \mathit{NET}\{(P, A_P) \mid P \in \{P_i \mid 0 \leqslant i \leqslant n\} \cup \{\Theta(\mathit{COL}), \Theta(\mathit{DIS})\}\}$$

The processes *DIS* and *COL* between them participate in every action of the parallel combination, and they both alternate between performing an internal event and an external one. Hence, no more than two internal events can occur between external events. Therefore

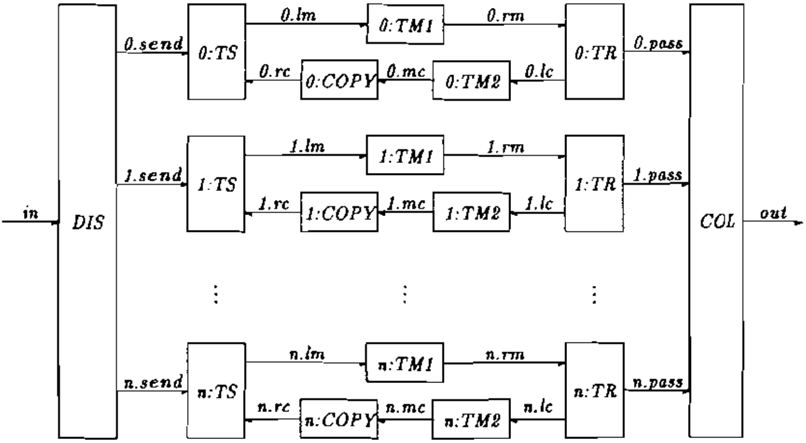
$PAR(\{(P, A_P) \mid P \in \{P_i \mid 0 \leqslant i \leqslant n\}\} \cup \{(\Theta(COL), \sigma(COL)), (\Theta(DIS), \sigma(DIS))\})$

is limited on $\bigcup_{i=0}^{n}(i.send \cup i.pass)$. We therefore obtain that

$$SWP \sqsubseteq_f TSWP$$

It is proved in [PS88] tbat SWP **sat** $out \leqslant_{w+2} in$, and it is straightforward to prove that SWP is a buffer in the failures model. Hence we obtain that $TSWP$ is a buffer, and that the protocol is correct.

The full structure of the sliding window protocol is as follows:



This is obtained hy instantiating each of the i : *TABP2* with its component processes, from the diagram on page 158.

9 Conclusions, Comparisons, and Future Work

9.1 Conclusions

In this thesis, we have presented a variety of methods applicable to the verification of real-time concurrent systems. We have exhibited a sound and complete proof system, defined a nnmber of high-level specifications and produced laws concerning their interaction, and produced a method for translating specifications between the different models of the hierarchy. We have shown how operators for introducing time-critical behaviour, snch as timeout and interrupt, and communication constructs such as channels, input and output, and chaining, can be added to the syntax of *TCSP*. Finally, we have demonstrated the application of these verification techniques to processes involving these constructs.

The restriction of our class of specifications to behavioural specifications has been crucial for both the development of the proof system and the application of timewise refinement to specifications. Behavioural specifications in the various models are sufficient to capture safety, liveness, and real-time specifications, and to distinguish deadlock, divergence, and possibility of divergence. Further, many useful results can be obtained for the class of behavioural specifications: for example, any behavioural specification is continuous. However, the definition of non-retraction serves as a reminder that not all desirable specifications can be written as behavioural specifications.

In a specification-oriented semantics [OH83], each process is identified with its strongest (behavioural) specification. Snch a semantics will be compositional when the denotational semantics are *directly* compositional, in the sense described in section 4.5. As we have seen, this is the case for TM_F but not for TM_{FS}. We employ a congruent semantic domain TM_{FI} which yields a complete proof system and a corresponding mapping from processes to strongest specifications. Tbis may also be considered as a complete proof system for TM_{FS} for a restricted class of specifications: those specifications on TM_{FS} corresponding to behavioural specifications on TM_{FI}. Thus a compositional specification-oriented semantics may be given for *TCSP* into specifications on TM_{FS}, but the domain of specifications will be a subset of the behavioural specifications on TM_{FS}.

The proof system for TM_F has already been used in a case study [Jac89]. In this stndy, aircraft engine control software was specified using behavioural specifications, and the proof rules were used to verify a proposed *TCSP* implementation with respect to the requirements specified; it was demonstrated that the rules are usable in addition to being sound and complete. It is also apparent that the rules can be applied in a mechanical fashion, indicating that a mechanical proof assistant would be both possible and useful.

The general specifications presented in chapter 5 capture a few aspects of good behaviour that we would wish to establish of systems. In particular, non-retraction, responsiveness, and promptness are useful properties for establishing correctness. We have seen that each can be shown to hold of a process by an examination of its syntax. This enables us to build processes which are guaranteed to meet specifications of this sort, by employing only those process coustructors which preserve the desired property.

The chapter on timewise refinement showed how different models may be used in the verification of *TCSP* systems, by enabling the translation of specifications between models. The simpler model may then be used to provide a verification for an untimed version of the system, which serves as a verification of the timed system. This approach allows us to use the simplest model that permits a verification of the property uuder consideration. Our verification that the sliding window protocol described in chapter 8 met tbe specification $out \leqslant in$ was much easier in M_T than it would have been in TM_{FS}, where any proof would have involved the manipulation of (*trace, stability, refusal*) triples. This approach also provides the basis for a timewise refinement approach to development, since a real-time specification may be broken down into the conjunction of time-dependent and time-independent specifications. A *CSP* process may then be developed and verified in M_T (or M_F) relative to the translation of the time-independent component of the original specification. Any timewise refinement of that process will then meet the time-independent specification, so our development task reduces to finding a timewise refinement of the untimed process which also meets the time-dependent constraints.

We have seen how process constructors can be built from the syntax of standard *TCSP*; one advantage of this approach is that the constructors are automatically continuous, fulfilling an essential proof obligation. Explicit time-critical constructs such as timeout and interrupt have been defined, and algebraic laws concerning their interaction with other process constructors have been formulated. Communication constructs have also been defined: channels, input, output, chaining, and the more general network construct, were defined in a fashion analogous to that of untimed *CSP*, although the timed behaviour of the hiding operator ensures that chaining operator does not obey the same laws as its untimed counterpart. The concept of a huffer was characterised, and it was seen that the specification 'timed buffer' is a timewise refinement of the standard untimed specification of a buffer. We have also seen that to guarantee communication, we require of one of the two communicating processes that it be non-retracting on their mutual channel. Non-retraction and promptness were seen to be useful in supporting timewise refinement of networks. Chapter 8 illustrated how the specification and proof techniques presented in this thesis could be applied, with reference to the communication constructs and protocols as well as the time critical operators, and demonstrated that *TCSP* is becoming a powerful tool for specification and verification of real-time systems.

9.2 Comparisons

Approaches to the specification and verification of real-time systems fall into three broad groups. One method is to develop a general specification language whicb enables reasoning about system requirements, and which is applicable to many process description formalisms via corresponding verification methods. Another approach is that of a process algebra with an associated operational semantics or set of axioms for establishing equivalence between processes. Specifications may be predicates on a graph generated by the operational semantics (as in the case of Petri nets). More often, a specification is a process, and a verification of a candidate process is a demonstration that it exhibits the same behaviour as the specifying process: that there is a *bisimulation* between them. There are different kinds of bisimilarity relations, and the particular relation used in a verification will reflect the aspects of behaviour we wish to verify. The third approach, taken in this thesis, is to provide a process description language with a denotational semantics. A specification will then be a predicate on tbe semantic domain, so the nature of the specification language will be dependent on the semantics given.

There is a degree of overlap between these approaches. A general specification language may be applicable to process algebras with an operational semantics, and also to programming languages with denotational semantics. In many cases, a process algebra may be provided with both a denotational and an operational semantics, allowing results from the second and third approaches to be combined.

Temporal Logic

Standard temporal logic is particularly useful as a specification language, since it can succinctly express botb safety and liveness properties; there uow exists a large amount of work in the literature, which is beyond the scope of this section — a survey of work up to 1986 may be found in [Pnu86]. Temporal logic has been successfully applied to the verification of systems (see e.g. [MP82], [HO83], [SL87], among many others), and compositional proof techniques have been developed (see e.g. [BKP84] for discrete time, [BKP85] for continuous time). We will contrast temporal logic specifications with our specifications on *TCSP* processes, which are predicates on TM_{FS} or TM_F.

Temporal logics consist of propositional atoms which are assigned truth values at every instant, together with temporal operators such as 'eventually' ($\Diamond$), 'henceforth ($\Box$), 'unless' ($\mathcal{U}$) and 'since' (S). The semantics rests on a time domain, consisting of all instants together with an ordering on them; it is usually taken to be a total order, although other models (such as branching time, and more general partial orders) are also possible. The meaning of a temporal logic statement thus depends on the time domain under consideration.

The propositional atoms will often be taken to be predicates on the state of a system. For example, if r bolds of a state in which the system is ready to perform an a action, tben tbe assertion tbat $\Diamond r$ holds at a certain time of a system means that it will eventually be tbe case that the system will be ready to perform a. Access to state information allows the easy formulation of operational concepts that were not so natural to formulate as predicates on observable behaviours. Non-retraction may be succinctly captured in this way: if s holds exactly when a occurs, and r holds exactly when a is ready to occur, then

$$\Box(r \Rightarrow r\mathcal{U}s) \text{ holds of } P$$

specifies that once P is ready to perform a, then it will remain ready at least until it is actually performed. This form of non-retraction is termed 'persistence' in [Pnu86]. The definition contrasts with that of non-retraction on *TCSP* processes in terms of observahle behaviours, because the denotational semantics of *TCSP* processes does not characterise state information. For example, the two processes

$$(a \rightarrow STOP) \sqcap \bot$$

and

$$(a \rightarrow STOP(\sqcap((a \rightarrow STOP) \overset{2}{\triangleright} \bot) \sqcap \bot$$

have the same semantics, but they have different possible states. Using the temporal logic definition we would conclude that tbe first process is persistent on a, whereas the second is not. Hence the access to state information allows some distinctions to be made that the denotational semantics of *TCSP* does not make.

Temporal logic may also be used as an assertion language on computational models which distinguisb systems only by their observable events, by taking the propositional atoms of the temporal logic to he predicates on (finite) past histories. This allows a direct translation between behavioural specifications and temporal logic specifications. A behavioural specification of the form X **sat** $S(s)$ translates to a temporal logic specification of the form $\Box S(s)$. This says that if s is the record of what P has performed up to a time, then S must hold of s.

The work on temporal logic discussed so far is concerned only with qualitative temporal reasoning. However, the analysis of time-dependent and time-critical systems often requires knowledge about the actual times at which events occur or become available. To say that a process is (t, a)-responsive is to say that it must make available event a within time t. This is expressible within temporal logic using the next operator ($\bigcirc$), since it is equivalent to saying

$$\bigvee_{r=0}^{t} \bigcirc^{r}(a)$$

where the time between successive instants is one time uuit. However, this is only possible when time is discrete, since a dense time domain will not support a next operator; at most we can insist tbat a will eventually be available.

Koymans argues this case in [Koy89], and proposes an extension to the lauguage of temporal logic to include temporal operators which have explicit times associated with them. To do this, he requires that there be a metric fuuction on the time domain which yields the temporal distance between two instants. It is then possible to define the required operators. For example, the operator $\mathbf{F}_{\delta}(\phi)$ asserts of its argument that there is a point exactly δ in the future at which ϕ will be true. $\mathbf{F}_{<\delta}(\phi)$ asserts that ϕ will hold within δ. Its dual, $\mathbf{G}_{<\delta}(\phi)$ asserts that ϕ will hold at all instants within δ.

The atomic propositions may again access state information, allowing the formulation of more operatioual style definitions. Respousiveness of P may be defined as follows:

$$\Box(\mathbf{F}_{<t}(r)) \text{ holds of } P$$

where r holds of a state iu which a is available. It says that a will be available within time t of any instant. Promptness may also be expressed by this specification language: that if a is not available by time t, then it will not become available before the next observable event. Taking s to be true precisely when an event other than a occurs, we may specify the promptness of P as follows:

$$\Box(\mathbf{G}_{<t}\neg r \Rightarrow (\neg r)\mathcal{U}s \text{ holds of } P$$

The metric temporal logic language presented by Koymans provides the quantitative element we require for the analysis by temporal logic of systems whose correctness rests on time-critical considerations. This specification language is still fairly recent, and there are not compositional rules as yet. However, it is claimed in [Koy89] that the development of a hierarchical method would not seem to be significantly more difficult than that developed for standard temporal logic.

Temporal logic is not associated with any particular programming notation. This yields the advantage that the specification and proof techniques apply to a wide range of system descriptions. The corresponding disadvantage is that for any particular description of a program we must first show that it meets the temporal logic specification claimed of it. There are methods for establishing this wheu the specification language is staudard temporal logic (see e.g. [BKP85], [Pnu86]), but as yet there are few such methods for metric temporal logic ([HW89] is a notable exception, discussed later in this chapter). The descriptions of time critical systems will in general be complex, and it will not be immediately apparent whether or not they meet particular specifications. Methods and tools must therefore be developed which allow us to relate specifications to process description languages (such as *TCSP*).

Operational Semantics

A different approach to system design and verification is to provide an operational semantics for the system under consideration. This may be done either by modelling the states of the system as a (nsually finite) graph and specifying the circumstances under which transitions may occur from one node (representing a state) to another, as in Petri net theory; or by using a process algebra with an operational semantics, such as *CCS*, *ATP*, *ACP*, or *LOTOS*, to represent systems.

Graphs

We will focus on the Petri net, since it is the most common example of the graph-theoretic approach to analysis of concurrent systems, though there are alternatives (e.g. HMS machines [FG89]). A safety property of a system may be expressed as the condition that a given set of nodes is unreachable from the initial state. Liveness properties are concerned with the availability of transitions. (see e.g. [Mer87]). For example, deadlock freedom is expressed as the requirement that every (reachable) state enables at least one transition.

A Petri net (see e.g. [Pet77]) consists of a set of places, a set of transitions, and a set of directed arcs from places to transitions and from transitions to places. It also has a marking, consisting of an assignment of tokens to places. A transition t is enabled under a marking if all the places which have an arc to t have some token assigned to them. When the transition fires, the marking of the net alters: the tokens enabling t are removed, and a token is assigned to each place to which there is an arc from the transition.

There are several approaches to the introduction of time into Petri nets. For example, delays may be associated with places (see e.g. [Sif80],[CR85]) or with transitions. In these respective cases, a token must remain on a place for at least the length of time associated with that place, or a transition has a duration. It is claimed in [LS87] that these two approaches are equivalent. Alternatively, a master timing mechanism may be introduced (see e.g. [CR83]). We shall consider a different approach, described in [LS87], originally presented in [MF76]. The formation of a timed Petri net here involves associating witb each transition a minimum and maximum length of time that the transition may be continuonsly enabled without firing. Hence a transition t may fire at time τ if it has been continuously enabled since $\tau - min(t)$; it must fire if it has been continuously enabled since $\tau - max(t)$. Petri nets may be considered as a special case of Time Petri nets, with each transition having minimum enabling time 0 and maximum enabling time ∞.

A safety property will state that particular undesirable markings cannot be reached from the initial marking by a sequence of enabled transitions. For example, a mutual exclusion property will be captured by the requirement that the place p representing the critical section will never be assigned more than one token by any reachable marking. Safety properties are preserved by an assignment of minimum and maximum enabling times to the transitions, which we may consider as a 'timewise refinement' of a net. It was remarked in [LS87] that such an assignment can only reduce the set of reachable markings, in a fashion analogous to that of timewise refinement in *TCSP*, where the set of possible traces is reduced by a $\sqsubseteq_t$-refinement.

Liveness properties concern tbe eventnal enabling of transitions. For example, we may say that a transition is live if there is a sequence of transitions from any reachable marking after which the transition is enabled (see e.g. [Mer87]). An analogue of (t, A)-responsive for Petri nets may be that at least one of the transitions in the set A must be enabled within t time units of any moment. Deadlock freedom is expressed as the requirement that any reachable marking enables at least one transition. Some liveness properties, such as deadlock-freedom, are preserved by timewise refinement. Others, such as liveness of an entire net (defined in [Mer87] as the condition that every transition is live), are not necessarily preserved. There does not appear to be a systematic way of transforming untimed liveness properties into timed ones, in the way that *CSP* liveness specifications may be transformed into *TCSP* ones by means of the strongest refinement operation.

Algebras

Process algebras with operational semantics provide another way of designing and analysing systems. A syntax for the algebra is given together with rules describing the valid transitions: these rules constitnte the operational semantics of the language, and will generate a tree of possible transition sequences for any process written in the algebra. This approach also considers a process algebra to be specification language, in that a process written in a process algebra is a description of our requirements. For example, the ideal one-place buffer in *CCS* is captured as follows:

$$Buff \triangleq in(x).\overline{out}(x).Buff$$

We may think of a verification that a process is a one place buffer as a demonstration that it is *bisimilar* to *Buff*. Two processes are bisimilar if they exhibit the same behaviour. (There are a number of bisimilarity relations depending on what aspect of behaviour we are interested in: see e.g. [Mil89].) Each process algebra has laws consistent with its operational semantics for rewriting process descriptions while preserving bisimilarity. Verifications of systems therefore take the form of algebraic manipulations, which establish a bisimilarity relation (see e.g. [LM86] for an example of a protocol verification).

There have been a number of different approaches to the inclusion of time in a process algebra. Most of these are extensions or adaptations of untimed algebras. For example, current research at Snssex [Reg89] is investigating the consequences of the addition of a single *WAIT* statement to *EPL* [Hen88]. The treatment of time is discrete, and actions are instantaneous.

Synchronised *CCS* [Mil89] considers the duration of a transition to be one time unit. All concurrent components of a system proceed at the same rate, in lockstep; time is thus considered to be discrete. This yields an elegant and simple calculus. A form of 'timewise refinement' is possible, since *CCS* processes may be 'implemented' by *SCCS* ones. It also seems possible, for a given bisimilarity relation between *CCS* processes, that there may be a weak bisimilarity relation (that ignores the passage of time) that holds between *SCCS* processes precisely when the corresponding untimed *CCS* processes are bisimilar. However, a complete set of axioms must also be found for such a relation if it is to be useful. I am presently unaware of any results of this form.

The algebra for timed processes *ATP* [NRSV89] treats time rather differently. Actions are considered to take no time, except for an explicit synchronisation action χ which is not present in the syntax but which appears in the semantics of the timeout operator and the terminated process (which allows time to pass but which can perform no other action). Concurrent processes are completely asynchronous, (although they may communicate) apart from the requirement that they must all synchronise on a χ action. Sneccessive instants are identified with sneccessive occurrences of the χ action, so time is in some sense discrete. However, processes may perform arbitrarily many actions between two successive instants. An operational semantics is provided for the *ATP* language, and a verification is a demonstration of a bisimilarity relationship between the system under consideration and a system we know to be correct (see [NRSV89] for a verification of an alternating bit protocol).

Timed *LOTOS* [QF87] and Timed *ACP* [BB89] are botb process algebras which allow continuous time. The syntax of Timed *LOTOS* allows the hidden action τ to be treated on the same footing as any visible action. There is also a delay operator *WAIT t* (non-negative real t), which postpones the commencement of the process following it. Thus constructs similar to the timeout construct for *TCSP* may be built from the basic operators of Timed *LOTOS*. The existence of an operational semantics once again allows specifications to he written as Timed *LOTOS* processes, and verifications again cousist of demonstrating a bisimilarity between a candidate process and a process we take to be the specification. Timewise refinement of basic *LOTOS* processes into Timed *LOTOS* processes appears straightforward, but as in the case for *SCCS* verifications of untimed processes will not be preserved by timewise refiuement, unless a weak bisimulation relation is used.

Timed *ACP* [BB89] is an extension of *ACP* [BKl84] to inclnde time, which is considered to be the non-negative reals. Each event a is associated with a time t, so atomic actions are of the form $a(t)$. An operational semantics is given for the language, which provides the basis for a number of bisimulation preserving algebraic laws, as we have by now come to expect. A process is again regarded as a specification, and a verification consists of a demonstration that two processes are bisimilar. It is pointed out in [BB89] tbat an implementation of an *ACP* process can be imagined as a real time version of the specification. However, it is also pointed out that two equivalent processes will not in general have equivalent real time implementations, and the question is left open as to what predictions can be made concerning the behaviour of a timed version of an untimed *ACP* process from an untimed verification.

The contrast is sharp between the specification and verification techniques of the process algebras discussed ahove, and the approach of this thesis. The *TCSP* approach, in common with other approaches via denotational semantics (as we shall see), considers a specification to be a property, and a verification to be that the semantics of a candidate process bas that property. Any other properties the process may have are irrelevant to the verification. On the other hand, a proof that two processes are hisimilar will establish that they share *every* property identified by the bisimilarity relation. Thus a verification often rednces to algehraic manipulations of the process description we wish to verify, until a process is obtained that is obviously correct (in the sense that it has the property we require). The proofs of the alternating bit protocol in [NRSV89] and again in [BB89] are examples of this approach.

The consideration of processes as specifications will not in general allow verifications to be translated into timed verifications by timewise refinement, since the properties we would hope to preserve are not made explicit. It also seems likely that the addition of time into these process algebras will mean that the method of proving processes correct by transforming their descriptions to ones which are obviously correct will predominate over the approach whereby a process is presented as a specification and a bisimulation then demonstrated. It appears that there will be too much information contained in any process description to make it useful as a specification; for example, the canonical one place buffer *Buff* given above must perform $in(x)$ on its first step, which may not he a property we wish to retain. If the bisimilarity relation is weakened to ignore the passage of time, then we may also lose a property we wish to retain; for example, that it is always ready to output the instant after input occurs.

A less common form of specification on process algebras is that of the specifying equation: this is of the form $C[P] = D[P]$, placing the requirement on P that when it is placed in context C and in context D the results are indistinguishable (i.e. bisimilar). The timed behaviour of $C[\,]$ and $D[\,]$ need not be the same, so there will in general be fewer processes P satisfying the equation than in the untimed version. For example, we may attempt to capture a class of process by requiring of a candidate process B that it satisfies

$$B \gg \mathit{Buff} \approx \mathit{Buff} \gg B$$

However, the timed behaviour of *Buff* requires that it perform $in(x)$ at the first instant, so any process B that does not allow an input at the first instant will not meet the specification. If *Buff* is replaced by a more relaxed buffer which does not insist upon input at the first instant, then a process B which does insist on such input will not meet the specification. On the other hand, if the bisimilarity relation is relaxed to allow both possibilities, then we may lose more timing information than we wish. It seems that the inclusion of timing information often leads to overspecification when this technique is applied.

Hennessy-Milner logic [HM85], called process logic in [Mil89], provides the approach to specification for *CCS* that is closest to the denotational semantics approach. It would seem that the logic could equally well be applied to other process algebras, although I am not aware of any attempts to do this. The logic is used to express safety properties of systems using two modal operators (which are dual), $\langle\rangle$ and $[]$. The specification $\langle\alpha\rangle G$ asserts of a process P that it may perform α and reach a state of which G holds. The specification $[\alpha]G$ asserts of P that whenever P performs α it reaches a state of which G holds.

Little work seems to have been done with process logic concerning the specification and verification of systems. In particular, I am not aware of any case studies in which a process is specified using the logic, and then proved correct; such a study would be useful. It also seems likely that a form of timewise refinement will be possible for such specifications. This is a topic for further research.

Denotational Semantics

The provision of a denotational semantics for a programming language or process algebra involves the association of each program with a mathematical object considered to be the meaning of the program. The possible meanings of programs are members of a semantic domain. A denotational semantics is compositional, in that the meaning of a compound program may be deduced from the meanings of its constituents, without any reference to their syntax. Specifications on processes are expressible as predicates on the semantic domain. In the most general case, any mapping from the semantic domain to $\{TRUE, FALSE\}$ will be a specification, but it may be considered desirable to restrict the class of specifications to make reasoning easier. For example, the set of proof rules presented in this thesis is complete for the class of behavioural specifications.

The semantic domain for a process algebra is reflected in the nature of the possible specifications. As we have seen, the semantics provided for *TCSP* processes do not refer to state information. This is because we do not wish to distinguish processes whose external behaviour is identical, even if their internal states may be different. It follows that specifications on *TCSP* processes may not be concerned with the possible states of a process. Such specifications are permitted by semantic domains which contain state information as a component of the possible denotation of a process. The choice of a semantic domain for a process algebra is concerned with the distinctions between processes we wish to make, which in turn is closely associated with the kind of specifications we hope to express. Of course, a given process algebra may be supplied with different semantic domains: these will permit different sorts of specification. For example, *CSP* has the traces model M_T and the failures model M_F. The traces model allows only safety specifications, and cannot be used to verify liveness properties, since it is nnable to distinguish live processes from possibly deadlocking ones. The failures model permits such distinctions.

Methods of verification using a denotational semantics or an operational semantics may overlap. The denotational semantics may give rise to a complete set of algebraic laws which allow the semantics-preserving transformation of any syntactic process to one in normal form. Thus an algebraic proof system, of the sort associated with operational semantics, is possible. Such proof systems may also permit the verification that one process is more deterministic than another. (see [Bro83] for an example of this.)

This approach is taken in [GLZ88], where a denotational semantics based on discrete time is given for a *CSP*-based language which has a timed prefix operator: the semantics of a process is the set of possible behaviours of that process. A complete proof system is presented for finite processes (i.e. those defined without recursion); further laws permit reasoning about recursive processes, but these laws are not complete.

An example specification and verification is presented in [LZ88]. The specification on the system is that it is more deterministic than the process $SAFE$, which is taken to be the most non-deterministic process that captures the safety requirement. A verification that $SAFE \sqsubseteq NUCLEAR$ is presented, from which it is concluded that $NUCLEAR$ meets tbe safety requirement. Specifications expressible in the form $P \sqsubseteq X$ (for a fixed P) are similar to behavioural specifications, since X will be more deterministic than P precisely when its behaviours are a subset of those of P. The requirement is therefore captured by the behavioural specification X **sat** $(b \in \mathcal{E}[\![P]\!])$, where b ranges over behaviours, and $\mathcal{E}[\![P]\!]$ denotes the semantics of P.

An alternative approacb is to provide an explicit specification language together with a denotational semantics for a given process algebra. For example, in [HW89] a denotational semantics is provided for an occam-like programming language: this allows either discrete or continuous time. Each possible behaviour is a mapping from time to a quadruple representing the channels on which a communication is taking place, those on wbich tbe process is waiting to send, those on which it is waiting to receive, and a boolean value representing whether the process has terminated. The semantics of a program is the set of its possible behaviours. The specification language is based on real-time temporal logic similar to that presented in [KR83], and compositional proof rules are presented which relate programs to specifications. For example, we have the rule

$$send(c) \ \mathbf{sat} \ c! \, \mathcal{U} \ (c \ U_{=1} \ done)$$

for the program $send(c)$. This states that it is ready to send on c until (weak until) the transmission occurs; this is followed one time-unit later by termination (strong until indexed by 'exactly one time-unit'). The rules form a sound proof system which is also complete relative to provability of valid formulae in the specification language.

This proof system, like the one presented in this thesis, exploits the compositional nature of the semantics of the language. There have been denotational semantics offered for a variety of real-time programming languages (e.g. [KSRGA88],[BG87],[SN89], in addition to those already discussed). The semantic definitions should in each case give rise to a set of sound proof rules (this is done in [HR89]). Completeness, however, will need to be established separately, since it does not follow in general; this was illustrated by the rules available for TM_{FS}, where our inability to establish completeness motivated the move to a different (though congruent) semantics.

Although many of the semantics for real-time systems are given for extensions of established programming languages, I am unaware of any attempts to refine specifications and verifications of programs written in the original language, in the way that timewise refinement for *TCSP* allows the transformation of untimed specifications into timed ones. It is clear tbat any untimed program may also be considered within the framework of time, and so properties such as deadlock freedom will still hold of it; but it is often desirable to prove that such specifications hold of timed programs which contain explicit timing constructs, and if such properties do not follow from timing considerations, then a timewise refinement relation may be useful. The hierarchy of models for *CSP* and *TCSP*, together with tbe projection mappings between them, provide the fonndation for timewise (and other) refinement relations. It wonld be interesting to see how similar results could be established for other languages.

9.3 Future Work

The previous section has highlighted important areas where further research into specification and verification methods would be useful. The provision of an operational semantics for *TCSP*, either via timed Petri nets or in terms of potentially infinite trees, would allow the approaches to establishing correctness that we have just discussed. A set of algebraic laws which preserve some form of eqnivalence (such as bisimilarity) may be obtainable, and wonld make some verifications easier. Perhaps the most important immediate task is the development of a real-time temporal logic based specification language for *TCSP*, similar (or even identical) to the metric temporal logic presented in [Koy89]. This would provide an alternative means of specifying systems, compatible with the body of research on temporal logic that already exists, and may lead to a complete proof system similar to that presented in [HW89]. D. Jackson, a doctoral student at Oxford, is currently working on such a specification language.

An infinite failures model is needed to support the distinction between arbitrary delay and infinite delay, and the distinction between arbitrarily fast and infinitely fast. As we pointed out in the section on infinite behaviours, *any* model consisting purely of finite behaviours will not be able to make this distinction. An infinite failures model would have as its semantic domain sets of infinite (*trace,refusal*) pairs, with each pair representing a possible record of the complete execution of a process. We would expect the restrictions of these pairs to any finite time to yield the finite behaviours of TM_F, yet the (infinite) semantics of a process will not necessarily be the set of all limits of the finite behaviours. In particular, if $\mathcal{I}_T[\![P]\!]$ denotes the infinite failure semantics of P, then we would expect the semantic equations for indexed non-deterministic choice, hiding, and renaming to be as follows:

$$\begin{aligned}
\mathcal{I}_T[\![\bigsqcap_{i\in I} P(i)]\!] &= \bigcup_{i\in I} \mathcal{I}_T[\![P(i)]\!] \\
\mathcal{I}_T[\![P \setminus A]\!] &= \{(s \setminus A, \aleph \mid (s, \aleph \cup [0,\infty) \times A) \in \mathcal{I}_T[\![P]\!]\} \\
\mathcal{I}_T[\![f(P)]\!] &= \{(f(s), \aleph) \mid (s, f^{-1}(\aleph) \in \mathcal{I}_T[\![P]\!]\}
\end{aligned}$$

We would expect to retain all of the axioms for TM_F as axioms for the finite approximations of behaviours in the infinite failures model, with the exception of axiom 5. This axiom states that a process cannot perform an infinite number of events in a finite time, but its formulation for TM_F also prevents a process from performing arbitrarily many events, since the two notions are indistinguishable in TM_F. The axiom states for any set S of behaviours in the semantic domain that

$$t \in [0,\infty) \;\Rightarrow\; \exists\, n(t) \in \mathbb{N}, \forall\, s \in \mathit{traces}(S) \bullet \mathit{end}(s) \leqslant t \Rightarrow \#s \leqslant n(t)$$

This axiom would be too strong for an infinite failures model. We only wish to disallow infinitely many actions in a finite time, so we may weaken the axiom to:

$$\forall\, t \in [0,\infty), s \in \mathit{traces}(S) \bullet \#(s \upharpoonright t) < \infty$$

Recall that for an indexed choice or a prefix choice to be well defined, it was required that the set of component processes were uniformly bounded. The weakening of axiom 5 allows this restriction to be lifted, and so we gain the well-definedness of both forms of infinite choice for all arguments.

The 'eventually' operator in temporal logic enables the distinction between arbitrary delays and infinite ones (see e.g. [BKP85]). Consider the pair of processes

$$\begin{aligned}
P &\mathrel{\hat=} \bigsqcap_{n\in\mathbb{N}} \mathit{WAIT}\; n \;;\; a \rightarrow \mathit{STOP} \\
Q &\mathrel{\hat=} P \sqcap \mathit{STOP}
\end{aligned}$$

We would expect that P satisfies the temporal logic specification 'X will eventually offer a', since any resolution of the choice will eventually offer a; but that Q does not satisfy the specification, since it may resolve the choice in favour of $STOP$, in which case a will never be offered. However, as has previously been remarked, both P and Q have the same semantics in TM_F (and indeed in TM_{FS}). Hence any temporal logic specification language which wishes to distinguish between the two processes must be based on a model that makes such distinctions. Clearly the infinite failures model will make precisely the distinctions required, since the temporal logic assertions are concerned with infinite behaviour. Process P will not exhibit the behaviour $(\langle\rangle, [0,\infty) \times \{a\})$, so it will not be able to refuse a for all time. However, that behaviour is possible for Q, indicating that Q does not meet the specification required. Hence an infinite failnres model will be useful for the construction of a temporal logic based specification langnage.

The possibilities for timewise refinement are also enhanced by an infinite failures model. The reason that timewise refinement is not preserved by hiding and infinite non-determinism in general is that infinite non-determinism is not adequately modelled in TM_F. The equations above for hiding and indexed non-deterministic choice mean that both operators preserve timewise refinement (see lemmas 6.3.31 and 6.3.36), and hence that the chaining operator, whose definition involves the hiding operator, will also preserve timewise refinement. This will greatly ease verification of pipes via timewise refinement, and more generally of communicating networks of processes. I conjecture that an infinite failures model will yield the result $\Theta(Q) \sqsubseteq_f Q$ for *all TCSP* processes Q. However, at the time of writing I do not know what the structure of the semantic domain will be, nor which extra axioms will be required, so I am unsure as to how recursion will be defined and whether $P \sqsubseteq_f X$ will be a continuous specification. This is a topic for further research.

A refinement calculus for exploiting the timewise refinement relation would be invaluable. It is presently difficult to generate a refinement of an untimed *CSP* process. The only systematic way of doing this at present is either to insert delays at suitable points in the process description, or to resolve a non-deterministic choice by replacing it with a timeout construct. We reqnire more rnles for algebraically refining process descriptions, to enable us to work towards the timing reqnirements of a specification.

A form of refinement not mentioned in this thesis is *speedwise* refinement, through which processes become faster. A *faster than* relation between processes could be defined, which would enable us to compare speeds of various processes. This would provide us with the machinery to address problems associated with the speeds of processes. For example, we would wish to know when the speed-up of a component of a network will maintain correctness of the network, and whether it will provide a speedwise refinement of the entire network; this will not be the case in general: in the alternating bit protocol example, a significant speed np of the sender process could result in a timeout occnrring before an acknowledgement could possibly arrive; this could result in a less efficient protocol, since a second copy of the message will always be sent, even over ideal wires. The 'faster than' relation would also enahle us to examine which specifications are preserved by speedwise refinement (e.g. 'the process will not respond for 5s' will not be preserved, but 'the process will respond within 5s' will be).

In addition to these directions, there are other areas of research that have already heen proposed. The major extension to the hierarchy of models will be the addition of probability to the models; this will allow the specification and verification of fault-tolerant systems and knowledge-based systems, and will allow the modelling of fair processes. A strategy for pursuing this line of research appears in [Ree88]. Other lines of research into the use of *TCSP* for specification and verification are currently being pursued at Oxford. The results of this research will make easier the task of estahlishing correctness of real-time systems.

A Mathematical Proofs

A.1 The hiding lemma

The main result we will establish is the following hiding lemma:

$$Z \subseteq (X \setminus Y) \Rightarrow (P \setminus Z)\ {}_X\|_Y\ Q = (P\ {}_X\|_Y\ Q) \setminus Z$$

In order to prove this, we need to establish a number of subsidiary lemmas.

We first of all need a technical sublemma. Let P be an element of TM_{FS}. Let $\alpha, \delta, t, X, \aleph, \gamma$ be such that

$$(\alpha \geqslant \delta \geqslant end(t) \wedge (t, \alpha, \aleph \cup ([0, max\{\delta, end(\aleph)\}) \times X)) \in \mathcal{E}_T[\![P]\!]$$

Then define the following:

$$\begin{aligned}
w_0 &= t \\
w_{n+1} &= \begin{cases} w_n & \textbf{if } B_n = \emptyset \\ w_n{}^\frown\langle(t_{n+1}, a_{n+1})\rangle & \textbf{if } B_n \neq \emptyset \\ \text{where } (t_{n+1}, a_{n+1}) \in B_n & \end{cases} \\
t_0 &= max\{\delta, end(\aleph)\} \\
t_{n+1} &= \begin{cases} \text{some } \beta \text{ where } \exists\, a \bullet (\beta, a) \in B_n & \textbf{if } B_n \neq \emptyset \\ \gamma & \textbf{if } B_n = \emptyset \end{cases} \\
B_n &= \{(\beta, a) \mid t_n \leqslant \beta \leqslant \gamma \wedge \\
&\qquad a \in X \wedge (w_n{}^\frown\langle(\beta, a)\rangle, \aleph \cup ([0, \beta) \times X)) \in fail(\mathcal{E}_T[\![P]\!])\}
\end{aligned}$$

Sublemma A.1.1 $\forall\, n \bullet (w_n, \aleph \cup [0, t_n) \times X) \in fail(\mathcal{E}_T[\![P]\!])$,

We prove the sublemma inductively as follows:

Base Case: $n = 0$, the result follows from the definitions.

Inductive Step: Assume $(w_n, \aleph \cup ([0, t_n) \times X)) \in fail(\mathcal{E}_T[\![P]\!])$.

Case $B_n = \emptyset$

$$\begin{array}{lll}
\vdash & (w_n, \aleph \cup ([0, \gamma) \times X)) \in fail(\mathcal{E}_T[\![P]\!]) & a4 \\
\vdash & (w_{n+1}, \aleph \cup ([0, t_{n+1}) \times X)) \in fail(\mathcal{E}_T[\![P]\!]) & \square
\end{array}$$

Case $B_n \neq \emptyset$

Trivial

which establishes the sublemma. □

We are now in a position to prove

Lemma A.1.2

$$
\begin{array}{ll}
\forall \alpha, \delta, t, a, \aleph, \gamma \bullet & \\
\quad (\alpha \geqslant \delta \geqslant end(t) \wedge (t, \alpha, \aleph \cup ([0, max\{\delta, end(\aleph)\}) \times X)) \in \mathcal{E}_T [\![P]\!] & \\
\Rightarrow \quad \exists\, w, \alpha' \bullet w \setminus X = t \setminus X \wedge \alpha' \geqslant \delta \wedge (w, \alpha', \aleph \cup ([0, \gamma) \times X)) \in \mathcal{E}_T [\![P]\!]) &
\end{array}
$$

Proof.

Case

$$
\begin{array}{lll}
& \gamma \leqslant max\{\delta, end(\aleph)\} & \\
\vdash & \exists\, \alpha' \geqslant \alpha \bullet (t, \alpha', \aleph \cup ([0, \gamma) \times X)) \in \mathcal{E}_T [\![P]\!] & a10
\end{array}
$$

Case

$$
\begin{array}{lll}
& \gamma > max\{\delta, end(\aleph)\} & \\
\vdash & \exists\, N \bullet \forall s \in traces(\mathcal{E}_T [\![P]\!]) \bullet (end(s) \leqslant \gamma \Rightarrow \#s \leqslant N) & \\
& \wedge (w_r = w_{r+1} \Rightarrow \forall k \geqslant r \bullet (w_k = w_{k+1}) & \\
\vdash & w_{N+1} \neq w_N \Rightarrow \forall i \leqslant N \bullet (w_i \neq w_{i+1}) & \\
& \wedge (w_i \neq w_{i+1} \Rightarrow \#w_{i+1} = \#w_i + 1) & \text{defn } w_i, t_i, B_i \\
\vdash & (w_{N+1} \neq w_N \Rightarrow \#w_{N+1} = \#w_0 + N + 1 \geqslant N + 1) &
\end{array}
$$

$$
\begin{array}{lll}
\text{But} & end(w_{N+1}) = t_{N+1} \leqslant \gamma & \\
\vdash & \#w_{N+1} \leqslant N & \text{defn} \\
\vdash & w_{N+1} = w_N & \\
\vdash & B_N = \emptyset & \\
\vdash & t_{N+1} = \gamma & \\
\vdash & \exists\, \alpha' \bullet (w_{N+1}, \alpha', \aleph \cup ([0, \gamma) \times X)) \in \mathcal{E}_T [\![P]\!] &
\end{array}
$$

To complete the proof, it is sufficient to show that $\alpha' \geqslant \delta$:

Case

$$
\begin{array}{lll}
& w_0 = w_1 & \\
\vdash & w_{N+1} = w_0 \wedge B_0 = \emptyset & \\
\vdash & \alpha' = \alpha & a11 \\
\vdash & \alpha' \geqslant \delta &
\end{array}
$$

Case

$$
\begin{array}{lll}
& w_0 < w_1 & \\
\vdash & w_1 = w_0 \bullet \langle (t_1, a) \rangle & \text{defn} \\
\vdash & end(w_1) \geqslant t_1 \geqslant t_0 & \\
\vdash & \alpha' \geqslant end(w_{N+1}) \geqslant end(w_1) \geqslant t_0 \geqslant \delta &
\end{array}
$$

Corollary A.1.3 $(s,\aleph) \in \mathcal{F}_T\llbracket P \setminus A\rrbracket \Rightarrow \forall\, t\, \exists\, w \bullet w\setminus A = s \wedge (w, \aleph\cup[0,t)\times A) \in \mathcal{F}_T\llbracket P\rrbracket$

Lemma A.1.4 $Z \subseteq (X \setminus Y) \Rightarrow (P \setminus Z)\, {}_X\|_Y\, Q = (P\, {}_X\|_Y\, Q) \setminus Z$

Recall the definitions of hiding and of the parallel operator in TM_{FS} given in [Ree88]:

$$\begin{aligned} P \setminus X \;\; &= SUP \;\; \{(s \setminus X, \beta, \aleph) \mid \exists\, \alpha \geqslant \beta \geqslant end(s) \bullet \\ &\qquad (s, \alpha, \aleph \cup ([0, max\{\beta, end(\aleph)\}) \times X)) \in \mathcal{E}_T\llbracket P\rrbracket\} \end{aligned}$$

$$\begin{aligned} P\, {}_X\|_Y\, Q \;\; &= SUP \;\; \{(s, max\{\alpha_P, \alpha_Q\}, \aleph_P \cup \aleph_Q \cup \aleph_Z) \mid \exists (s_P, \alpha_P, \aleph_P) \in \mathcal{E}_T\llbracket P\rrbracket, \\ &\qquad (s_Q, \alpha_Q, \aleph_Q) \in \mathcal{E}_T\llbracket Q\rrbracket \bullet \alpha(\aleph_P) \subseteq X \wedge \alpha(\aleph_Q) \subseteq Y \\ &\qquad \wedge s \in (s_P\, {}_X\|_Y\, s_Q) \wedge \alpha(\aleph_Z) \subseteq (\Sigma - (X \cup Y))\} \end{aligned}$$

where $v\, {}_X\|_Y\, w = \{s \in (T\Sigma)^*_\leq \mid (s \upharpoonright (X \cup Y) = s) \wedge (s \upharpoonright X = v) \wedge (s \upharpoonright Y = w)\}$
The following definitions will be useful:

$$\begin{aligned} S(\alpha_P, \alpha_Q) \;\; &\Leftrightarrow \;\; \exists \aleph_P, \aleph_Q \bullet \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y \\ &\qquad (t \upharpoonright X, \alpha_P, \aleph_P) \in \mathcal{E}_T\llbracket P\rrbracket \\ &\qquad \wedge (t \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T\llbracket Q\rrbracket \\ &\qquad \wedge (\aleph_P \cup \aleph_Q) = (\aleph \cup ([0, max\{\beta, end(\aleph)\}) \times Z)) \upharpoonright X \cup Y \end{aligned}$$

$$D_{P,Q}(t,\beta,\aleph) \;\; = \;\; \{max\{\alpha_P, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\}$$

$$\begin{aligned} C_{P,Q}(s,\aleph) \;\; &= \;\; \{\beta \mid \; \exists\, t \bullet \;\; t \setminus Z = s \\ &\qquad\qquad \wedge\, sup\; D_{P,Q}(t,\beta,\aleph) \geqslant \beta \geqslant end(t)\} \end{aligned}$$

$$\begin{aligned} S^Z_P(s,\aleph) \;\; &= \;\; \{\beta \mid \; \exists\, t, \alpha \bullet \;\; \alpha \geqslant \beta \geqslant end(t) \wedge t \setminus Z = s \\ &\qquad\qquad \wedge (t, \alpha, \aleph \cup ([0, max\{\beta, end(\aleph)\} \times Z)) \in \mathcal{E}_T\llbracket P\rrbracket\} \end{aligned}$$

$$\begin{aligned} T_{P,Q}(s,\aleph) \;\; = \;\; \{max\{\alpha_P, \alpha_Q\} \mid \; \exists \aleph_P, \aleph_Q \bullet \;\; &\alpha_P = sup\; S^Z_P(s \upharpoonright X, \aleph_P) \\ &\wedge \alpha_P \geqslant 0 \\ &\wedge (s \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T\llbracket Q\rrbracket \\ &\wedge \aleph_P \cup \aleph_Q = \aleph \upharpoonright X \cup Y \\ &\wedge \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y\} \end{aligned}$$

$$\begin{array}{rcll} U_{P,Q}(s,\aleph) & = & \{max\{\delta,\alpha_Q\} \mid \exists \aleph_P, \aleph_Q \bullet & \delta \in S_P^Z(s \upharpoonright X, \aleph_P) \\ & & & \wedge (s \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!] \\ & & & \wedge \aleph_P \cup \aleph_Q = \aleph \upharpoonright X \cup Y \\ & & & \wedge \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y\} \end{array}$$

The definitions of these sets allow us to obtain the following equivalences (adopting the convention that $sup\emptyset = -\infty$):

$$(s,\alpha,\aleph) \in \mathcal{E}_T[\![P \setminus Z]\!] \Leftrightarrow \alpha = sup\ S_P^Z(s,\aleph)$$

$$(s,\alpha,\aleph) \in \mathcal{E}_T[\![(P \setminus Z)\ {}_X\|_Y\ Q]\!] \quad \Leftrightarrow \quad \alpha = sup\ T_{P,Q}(s,\aleph)$$

$$(s,\alpha,\aleph) \in \mathcal{E}_T[\![(P\ {}_X\|_Y\ Q) \setminus Z]\!] \quad \Leftrightarrow \quad \alpha = sup\ C_{P,Q}(s,\aleph)$$

Hence it is sufficient to prove that, for arbitrary $s, \aleph$, and any processes P and Q, that $sup\ T_{P,Q}(s,\aleph) = sup\ C_{P,Q}(s,\aleph)$. This will be done by first showing that

$$sup\ T_{P,Q}(s,\aleph) \geqslant sup\ C_{P,Q}(s,\aleph)$$

and then by showing that

$$sup\ C_{P,Q}(s,\aleph) \geqslant sup\ T_{P,Q}(s,\aleph)$$

We first aim to show that $sup\ T_{P,Q}(s,\aleph) \geqslant sup\ C_{P,Q}(s,\aleph)$:

Assume $C_{P,Q}(s,\aleph) \neq \emptyset$. Then let $\beta \in C_{P,Q}(s,\aleph)$. Take $\delta \in D_{P,Q}(t,\beta,\aleph)$, where $t \setminus Z = s$. Then we have

$$\begin{array}{ll} & (t \upharpoonright X, \alpha_P, \aleph_P) \in \mathcal{E}_T[\![P]\!] \wedge \sigma(\aleph_P) \subseteq X \\ \wedge & (t \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!] \wedge \sigma(\aleph_Q) \subseteq Y \\ \wedge & (\aleph_P \cup \aleph_Q) = (\aleph \cup ([0, max\{\beta, end(\aleph)\}) \times Z)) \upharpoonright X \cup Y \end{array}$$

for some α_P, α_Q, $\aleph_P, \aleph_Q$ such that $max\{\alpha_P, \alpha_Q\} = \delta$. Let $\aleph_R = \aleph_P \setminus ([0,\infty) \times Z)$. Then

$$\begin{array}{ll} & (t \upharpoonright X) \setminus Z = s \upharpoonright X \\ \wedge & (t \upharpoonright X, \alpha_P, \aleph_R \cup ([0, max\{\beta, end(\aleph)\}) \times Z)) \in \mathcal{E}_T[\![P]\!] \\ \wedge & (t \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!] \\ \wedge & \aleph_R \cup \aleph_Q = \aleph \upharpoonright X \cup Y \\ \wedge & \alpha_P \geqslant end(t \upharpoonright X) \end{array}$$

Case

$$\begin{array}{ll} & \beta > \alpha_P \\ \vdash & (t \upharpoonright X, \alpha_P, \aleph_R \cup ([0, max\{\alpha_P, end(\aleph_P)\}) \times Z)) \in \mathcal{E}_T[\![P]\!] \\ \vdash & \alpha_P \in S_P^Z(s \upharpoonright X, \aleph_P) \end{array}$$

Case

$$\begin{array}{ll} & \beta \leqslant \alpha_P \\ \vdash & \beta \in S_P^Z(s \upharpoonright X, \aleph_P) \\ \text{since} & \beta \geqslant end(t) \geqslant end(t \upharpoonright X) \end{array}$$

So we deduce that $min\{\beta, \alpha_P\} \in S_P^Z(s \upharpoonright X, \aleph_P)$.

We have so far established

$$\begin{array}{ll} & sup\ D_{P,Q}(t, \beta, \aleph) \geqslant \beta \\ & \wedge\, S(\alpha_P, \alpha_Q) \Rightarrow (\exists\, \aleph_P \bullet min\{\beta, \alpha_P\} \in S_P^Z(s \upharpoonright X, \aleph_P) \\ \vdash & sup\ T_{P,Q}(s, \aleph) \geqslant sup\ \{max\{min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\} \end{array}$$

We again have two cases to consider.

Case

$$\begin{array}{ll} & \forall\, \alpha_P, \alpha_Q \bullet (S(\alpha_P, \alpha_Q) \Rightarrow \beta \geqslant \alpha_P) \\ \vdash & \{max\{min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\} = \{max\{\alpha_P, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\} \\ \vdash & sup\ \{max\{min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\} = sup\ D_{P,Q}(t, \beta, \aleph) \geqslant \beta \end{array}$$

Case

$$\begin{array}{ll} & \exists\, \alpha_P, \alpha_Q \bullet S(\alpha_P, \alpha_Q) \wedge \alpha_P \geqslant \beta \\ \vdash & sup\ \{max\{min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q)\} \geqslant \beta \end{array}$$

In either case we conclude that $sup\ T_{P,Q}(s, \aleph) \geqslant \beta$
(and that $T_{P,Q}(s, \aleph) \neq \emptyset$). □

We now aim to show that $sup\ C_{P,Q}(s, \aleph) \geqslant sup\ T_{P,Q}(s, \aleph)$:

Assume $T_{P,Q}(s, \aleph) \neq \emptyset$. Then $sup\ T_{P,Q}(s, \aleph) = sup\ U_{P,Q}(s, \aleph)$.

$$\begin{array}{ll} \text{Let} & \beta \in U_{P,Q}(s, \aleph) \\ \vdash & \beta = max\{\delta, \alpha_Q\} \\ & \wedge\, \alpha \geqslant \delta \geqslant end(t) \\ & \wedge\, t \setminus Z = s \upharpoonright X \\ & \wedge\, (t, \alpha, \aleph_P \cup ([0, max\{\delta, end(\aleph_P)\}) \times Z)) \in \mathcal{E}_T[\![P]\!] \\ & \wedge\, (s \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!] \\ & \wedge\, \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y \\ & \wedge\, \aleph_P \cup \aleph_Q = \aleph \upharpoonright X \cup Y \\ & \text{for some } \delta, \alpha, \alpha_Q, \aleph_P, \aleph_Q, t \\ \vdash & \beta = max\{\delta, \alpha_Q\} \\ & \wedge\, w \setminus X = t \setminus X \\ & \wedge\, \alpha' \geqslant \delta \\ & \wedge\, (w, \alpha', \aleph_P \cup [0, max\{\beta, end(\aleph)\}) \in \mathcal{E}_T[\![P]\!] \\ & \wedge\, t \setminus X = s \upharpoonright X \\ & \wedge\, (s \upharpoonright Y, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!] \\ & \wedge\, \aleph_P \cup \aleph_Q = \aleph \upharpoonright X \cup Y \text{ for some } w, \alpha' \qquad\qquad (\textit{Lemma A.1.2}) \end{array}$$

Let $u \in (w\ {}_X\|_Y\ (s \upharpoonright Y))$

$\vdash$ $(u \upharpoonright \alpha_P, \alpha', \aleph_R) \in \mathcal{E}_T[\![P]\!]$
$\wedge\, (u \upharpoonright \alpha_Q, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!]$
$\wedge\, \aleph_R \cup \aleph_Q = (\aleph \cup [0, max\{\beta, end(\aleph)\}) \times Z) \upharpoonright X \cup Y$
(defining $\aleph_R = \aleph_P \cup [0, max\{\beta, end(\aleph)\}) \times Z$)

Now $end(w) \leqslant max\{\beta, end(\aleph)\}$

$\vdash$ $end(u \upharpoonright X) \leqslant max\{\beta, end(\aleph)\}$

Also $end(u \upharpoonright Y) \leqslant \beta \leqslant max\{\beta, end(\aleph)\}$

There are two cases to consider here:

Case

$max\{\alpha', \alpha_Q\} \leqslant end(\aleph)$

$\vdash$ $D_{P,Q}(t, \beta, \aleph) = D_{P,Q}(t, max\{\alpha', \alpha_Q\}, \aleph)$

$\vdash$ $max\{\alpha', \alpha_Q\} \in C_{P,Q}(s, \aleph)$
(since $end(u) \leqslant max\{\alpha', \alpha_Q\}$)

Case

$max\{\alpha', \alpha_Q\} > end(\aleph)$

subcase $\beta \geqslant end(u)$
$\vdash \beta \in C_{P,Q}(s, \aleph)$

subcase $\beta < end(u)$

$\vdash$ $max\{\beta, end(\aleph)\} = end(\aleph) \geqslant end(u)$
(since $end(u \upharpoonright X) \leqslant max\{\beta, end(\aleph)\}, end(u \upharpoonright Y) \leqslant \beta$,
so $\beta < end(u) \leqslant max\{\beta, end(\aleph)\}$)

$\vdash$ $\alpha' \geqslant end(\aleph) \geqslant \delta$
$\wedge\, \alpha' \geqslant end(\aleph) \geqslant \alpha_Q$

$\vdash$ $max\{\alpha', \alpha_Q\} \in D_{P,Q}(u, end(\aleph), \aleph)$
$\wedge\, max\{\alpha', \alpha_Q\} \geqslant end(u)$

$\vdash$ $max\{\alpha', \alpha_Q\} \in C_{P,Q}(s, \aleph)$

Also $max\{\alpha', \alpha_Q\} \geqslant end(u) > \beta$.

Therefore $sup\ C_{P,Q}(s, \aleph) \geqslant sup\ U_{P,Q}(s, \aleph) = sup\ T_{P,Q}(s, \aleph)$ □

Corollary A.1.5 $Z \cap Y = \emptyset \Rightarrow (P \setminus Z)\ {}_X\|_Y\ Q = (P\ {}_{X \cup Z}\|_Y\ Q) \setminus Z$

B Semantic Models and Mappings

B.1 Reed's Hierarchy

The semantic models are arranged in a hierarchical structure as follows:

$$\begin{tikzcd}
 & & \underline{TM_{FS}} \arrow[d] \arrow[dr] \arrow[ddll] & & \\
 & & M_{FS} \arrow[dl] \arrow[dr] & \underline{TM^*_{FS}} \arrow[l] \arrow[d] \arrow[dr] & \\
\underline{TM_F} \arrow[r] \arrow[drr] & M_F \arrow[dr] & & M_S \arrow[dl] & TM_S \arrow[l] \arrow[dll] \\
 & & M_T & & \\
 & & TM_T \arrow[u] & &
\end{tikzcd}$$

Some Projection Mappings

$\Pi : TM_{FS} \rightarrow TM_F$ is defined on processes $Q \in TM_{FS}$ by

$$\Pi(Q) = fail(Q)$$

The projection mapping $\Pi : TM_{FS} \rightarrow TM^*_{FS}$ may be expressed using our notion of s reflected in $\aleph$. It is defined as follows:

Let $P \in TM_{FS}$. For each $s \in T\hat{\Sigma}^*_{\leqslant}$, let

$$P[s] = \{(\tilde{s}, \beta, \aleph) \in P \mid s \text{ is reflected in } \aleph\}$$

$$\begin{aligned}\Pi(P) = \{(s, \alpha, X) \in T\hat{\Sigma}^*_{\leqslant} \times TSTAB \times \mathbf{P}(\Sigma) \mid \\ & P[s] \neq \emptyset \\ & \wedge \\ & \alpha = sup\{\beta \mid (\tilde{s}, \beta, \aleph) \in P[s]\} \\ & \wedge \\ & (\alpha < \infty \wedge \exists (\tilde{s}, \beta, \aleph) \in P[s] \bullet X = \sigma(\aleph \upharpoonleft \alpha) \\ & \vee\ \alpha = \infty \wedge X \in \mathbf{P}(\Sigma))\end{aligned}$$

$\Pi : TM_{FS} \rightarrow M_F$ is defined on processes $Q \in TM_{FS}$ by

$$\begin{aligned}\Pi(P) = \{(tstrip(s), X) \mid \quad & (s, \alpha, \aleph) \in P \wedge \exists (s, \aleph) \in fail(P) \bullet \\ & \alpha \leqslant begin(\aleph) < \infty \wedge \sigma(\aleph) = X \\ & \vee \\ & \alpha = \infty \wedge X \in \mathbf{P}(\Sigma)\}\end{aligned}$$

B.2 Semantic Models and Mappings

We reproduce the definitions for [Ree88] of the (untimed) trace model, the (untimed) failures model, the untimed-failures timed-stability model, the timed failures model, and the timed failures stability model.

The traces model

The Evaluation Domain M_T.

We formally define M_T to be the set of all those subsets S of Σ^* satisfying:

1. $\langle\rangle \in S$
2. $s^\frown w \in S \Rightarrow s \in S$.

The Complete Partial Order $\sqsubseteq$ on M_T.

For $S_1, S_2 \in M_T$, let $S_1 \sqsubseteq S_2$ if and only if $S_1 \subseteq S_2$.

The Complete Metric d on M_T.

For $S \in M_T$, we define

$$S(n) \mathrel{\widehat{=}} \{s \in S \mid \#s \leq n\}.$$

The complete metric on M_T is defined:

$$d(S_1, S_2) \mathrel{\widehat{=}} \mathit{inf}\{2^{-n} \mid S_1(n) = S_2(n)\}.$$

The Semantic Function $\mathcal{T}$.

We now define a semantic function $\mathcal{T} : \mathit{CSP} \rightarrow M_T$.

$$\begin{aligned}
\mathcal{T}\llbracket \bot \rrbracket &\mathrel{\widehat{=}} \{\langle\rangle\} \\
\mathcal{T}\llbracket \mathit{STOP} \rrbracket &\mathrel{\widehat{=}} \{\langle\rangle\} \\
\mathcal{T}\llbracket \mathit{SKIP} \rrbracket &\mathrel{\widehat{=}} \{\langle\rangle, \langle\checkmark\rangle\} \\
\mathcal{T}\llbracket a \rightarrow P \rrbracket &\mathrel{\widehat{=}} \{\langle\rangle\} \cup \{\langle a\rangle^\frown s \mid s \in \mathcal{T}\llbracket P \rrbracket\} \\
\mathcal{T}\llbracket P \Box Q \rrbracket &\mathrel{\widehat{=}} \mathcal{T}\llbracket P \rrbracket \cup \mathcal{T}\llbracket Q \rrbracket \\
\mathcal{T}\llbracket P \sqcap Q \rrbracket &\mathrel{\widehat{=}} \mathcal{T}\llbracket P \rrbracket \cup \mathcal{T}\llbracket Q \rrbracket \\
\mathcal{T}\llbracket P \parallel Q \rrbracket &\mathrel{\widehat{=}} \mathcal{T}\llbracket P \rrbracket \cap \mathcal{T}\llbracket Q \rrbracket \\
\mathcal{T}\llbracket P \;{}_X\!\parallel_Y Q \rrbracket &\mathrel{\widehat{=}} \{s \mid s \upharpoonright X \in \mathcal{T}\llbracket P \rrbracket \;\wedge\; s \upharpoonright Y \in \mathcal{T}\llbracket Q \rrbracket \\
&\qquad\qquad \wedge\; s \upharpoonright (X \cup Y) = s\} \\
\mathcal{T}\llbracket P \mid\mid\mid Q \rrbracket &\mathrel{\widehat{=}} \{s \mid \exists\, u \in \mathcal{T}\llbracket P \rrbracket,\; v \in \mathcal{T}\llbracket Q \rrbracket \bullet \\
&\qquad\quad s \in \mathit{Merge}(u, v)\} \\
\mathcal{T}\llbracket P \,;\, Q \rrbracket &\mathrel{\widehat{=}} \{s \mid s \in \mathcal{T}\llbracket P \rrbracket \wedge \checkmark \notin s\} \\
&\qquad \cup \{s^\frown t \mid s^\frown\langle\checkmark\rangle \in \mathcal{T}\llbracket P \rrbracket \;\wedge\; t \in \mathcal{T}\llbracket Q \rrbracket\} \\
\mathcal{T}\llbracket P \setminus X \rrbracket &\mathrel{\widehat{=}} \{s \setminus X \mid s \in \mathcal{T}\llbracket P \rrbracket\} \\
\mathcal{T}\llbracket f^{-1}(P) \rrbracket &\mathrel{\widehat{=}} \{s \mid f(s) \in \mathcal{T}\llbracket P \rrbracket\} \\
\mathcal{T}\llbracket \mu\, P.F(P) \rrbracket &\mathrel{\widehat{=}} \text{The least fixed point of the continuous mapping with re-} \\
&\qquad \text{spect to the complete partial order } \sqsubseteq \text{ on } M_T \text{ represented} \\
&\qquad \text{by } F. \\
&\mathit{or} \\
\mathcal{T}\llbracket \mu\, P.F(P) \rrbracket &\mathrel{\widehat{=}} \text{The unique fixed point of the contraction mapping with} \\
&\qquad \text{respect to the complete metric } d \text{ on } M_T \text{ represented by} \\
&\qquad F.
\end{aligned}$$

We may also add the equations

$$\begin{aligned}\mathcal{T}[\![a : A \rightarrow P(a)]\!] &\triangleq \{\langle\rangle\} \cup \{\langle a\rangle^\frown s \mid a \in A \wedge s \in \mathcal{T}[\![P(a)]\!]\}\\ \mathcal{T}[\![\bigsqcap P(a)]\!] &\triangleq \bigcup \mathcal{T}[\![P(a)]\!]\end{aligned}$$

The failures model

The Evaluation Domain M_F.

We formally define M_F to be those subsets S of $\Sigma^* \times \mathsf{P}(\Sigma)$, satisfying:

1. $\langle\rangle \in \mathit{Traces}(S)$
2. $s^\frown w \in \mathit{Traces}(S) \Rightarrow s \in \mathit{Traces}(S)$
3. $(s, X) \in S \ \wedge \ Y \subseteq X \ \Rightarrow \ (s, Y) \in S$
4. $(s, X) \in S \ \wedge \ (\forall a \in Y, \ ((s.\langle a\rangle, \emptyset) \notin S)) \Rightarrow (s, X \cup Y) \in S$

The Complete Metric on M_F.

If $S \in M_F$, we define

$$\begin{aligned}S(n) \ &\triangleq \ \{(s,X) \in S \mid \#s < n\}\\ &\quad\cup\{(s,X) \mid \#s = n \wedge s \in \mathit{traces}(S)\}.\end{aligned}$$

The complete metric on M_F is defined:

$$d(S_1, S_2) \triangleq \mathit{inf}\{2^{-n} \mid S_1(n) = S_2(n)\}$$

The Semantic Function $\mathcal{F}$.

We now define the semantic function $\mathcal{F} : \mathit{CSP} \rightarrow M_F$.

$$\begin{aligned}\mathcal{F}[\![\perp]\!] &\triangleq \{(\langle\rangle, X) \mid X \in \mathsf{P}(\Sigma)\}\\ \mathcal{F}[\![\mathit{STOP}]\!] &\triangleq \{(\langle\rangle, X) \mid X \in \mathsf{P}(\Sigma)\}\\ \mathcal{F}[\![\mathit{SKIP}]\!] &\triangleq \{(\langle\rangle, X) \mid \checkmark \notin X\} \cup \{(\langle\checkmark\rangle, X) \mid X \in \mathsf{P}(\Sigma)\}\end{aligned}$$

$$
\begin{aligned}
\mathcal{F}\llbracket a \rightarrow P\rrbracket &\mathrel{\widehat{=}} \{(\langle\rangle, X) \mid a \notin X\} \\
&\quad \cup\{(\langle a\rangle^\frown s, X) \mid (s, X) \in \mathcal{F}\llbracket P\rrbracket\} \\
\mathcal{F}\llbracket P \Box Q\rrbracket &\mathrel{\widehat{=}} \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}\llbracket P\rrbracket \cap \mathcal{F}\llbracket Q\rrbracket\} \\
&\quad \cup\{(s, X) \in \mathcal{F}\llbracket P\rrbracket \cup \mathcal{F}\llbracket Q\rrbracket \mid s \neq \langle\rangle\} \\
\mathcal{F}\llbracket P \sqcap Q\rrbracket &\mathrel{\widehat{=}} \mathcal{F}\llbracket P\rrbracket \cup \mathcal{F}\llbracket Q\rrbracket \\
\mathcal{F}\llbracket P \parallel Q\rrbracket &\mathrel{\widehat{=}} \{(s, X \cup Y) \mid (s, X) \in \mathcal{F}\llbracket P\rrbracket \\
&\qquad\qquad\qquad\qquad \wedge (s, Y) \in \mathcal{F}\llbracket Q\rrbracket\} \\
\mathcal{F}\llbracket P\, {}_X\!\parallel_Y Q\rrbracket &\mathrel{\widehat{=}} \{(s, Z_P \cup Z_Q \cup Z) \mid s \restriction (X \cup Y) = s \wedge \\
&\quad Z_P \subseteq X \wedge Z_Q \subseteq Y \wedge Z \subseteq (\Sigma - (X \cup Y)) \\
&\quad \wedge (s \restriction X, Z_P) \in \mathcal{F}\llbracket P\rrbracket \\
&\quad \wedge (s \restriction Y, Z_Q) \in \mathcal{F}\llbracket Q\rrbracket\} \\
\mathcal{F}\llbracket P \mid\mid\mid Q\rrbracket &\mathrel{\widehat{=}} \{(s, X) \mid \exists\, u, v \in \Sigma^* \bullet (u, X) \in \mathcal{F}\llbracket P\rrbracket \\
&\quad \wedge (v, X) \in \mathcal{F}\llbracket Q\rrbracket \wedge s \in \mathit{Merge}(u, v)\} \\
\mathcal{F}\llbracket P \,;\, Q\rrbracket &\mathrel{\widehat{=}} \{(s, X) \mid \checkmark \notin s \wedge (s, X \cup \{\checkmark\}) \in \mathcal{F}\llbracket P\rrbracket\} \\
&\quad \cup\{(s^\frown w, X) \mid \checkmark \notin s \wedge (s^\frown\langle\checkmark\rangle, \emptyset) \in \mathcal{F}\llbracket P\rrbracket \\
&\qquad\qquad\qquad\qquad \wedge (w, X) \in \mathcal{F}\llbracket Q\rrbracket\} \\
\mathcal{F}\llbracket P \setminus X\rrbracket &\mathrel{\widehat{=}} \{s \setminus X, Y) \mid (s, X \cup Y) \in \mathcal{F}\llbracket P\rrbracket\} \\
&\quad \cup\{(s, Y) \mid \forall\, n \geq \#s,\ \exists\, w_n \in \mathit{Traces}(\mathcal{F}\llbracket P\rrbracket) \bullet \\
&\qquad\qquad\qquad w_n < w_{n+1} \wedge s = w_n \setminus X\} \\
\mathcal{F}\llbracket f^{-1}(P)\rrbracket &\mathrel{\widehat{=}} \{(s, X) \mid (f(s), f(X)) \in \mathcal{F}\llbracket P\rrbracket\} \\
\mathcal{F}\llbracket f(P)\rrbracket &\mathrel{\widehat{=}} \{(f(s), X) \mid (s, f^{-1}(X)) \in \mathcal{F}\llbracket P\rrbracket\} \\
\mathcal{F}\llbracket \mu P.F(P)\rrbracket &\mathrel{\widehat{=}} \text{The unique fixed point of the contraction mapping} \\
&\quad \widehat{C}(Q) = C(\mathit{WAIT}\,\delta\,;\,Q),\ \text{where } C \text{ is the mapping on} \\
&\quad M_F \text{ represented by } F. \ ?
\end{aligned}
$$

We may also add the equations

$$
\begin{aligned}
\mathcal{F}\llbracket a : A \rightarrow P(a)\rrbracket &\mathrel{\widehat{=}} \{(\langle\rangle, X) \mid A \cap X = \emptyset\} \\
&\quad \cup\{(\langle a\rangle^\frown s, X) \mid a \in A \wedge (s, X) \in \mathcal{F}\llbracket P\rrbracket\} \\
\mathcal{F}\llbracket \textstyle\bigsqcap P(a)\rrbracket &\mathrel{\widehat{=}} \textstyle\bigcup \mathcal{F}\llbracket P(a)\rrbracket
\end{aligned}
$$

The (Untimed Failures)-(Timed Stability) Model

The Evaluation Domain TM_{FS}^{*}

We formally define TM_{FS}^{*} to be those subsets S of $T\hat{\Sigma}_{\leqslant}^{*} \times [0,\infty] \times \Sigma$ satisfying:

1. $\langle\rangle \in traces(S)$
2. $s^\frown w \in traces(S) \;\Rightarrow\; s \in traces(S)$
3. $(s,\alpha,X) \in S \;\Rightarrow\; (\tilde{s},\alpha,X) \in S$
4. $(s,\alpha,X) \in S \;\wedge\; \underline{s \cong w} \;\Rightarrow\; (w,\alpha,X) \in S$
5. $s^\frown\langle(t,a)\rangle \in traces(S) \;\Rightarrow\; \exists\, t' \leqslant t \bullet (s \upharpoonright t').\langle(t',\hat{a})\rangle \in traces(S) \;\wedge$
   $(t' \leqslant t'' < t \Rightarrow (s \upharpoonright t'').\langle(t'',a)\rangle \in traces(S)\,)$
6. $\forall\, t \in [0,\infty),\ \exists\, n(t) \in \mathbb{N}$ *such that* $\forall\, s \in traces(S), (end(s) \leqslant t \Rightarrow \#s \leqslant n(t))$
7. $(s,\alpha),(s,\alpha') \in stab(S) \;\Rightarrow\; \alpha = \alpha'$
8. $(s,\alpha) \in stab(S) \;\Rightarrow\; end(s) \leqslant \alpha$
9. $(s,\alpha) \in stab(S) \wedge s^\frown\langle(t,\hat{a})\rangle \in traces(S) \;\Rightarrow\; t \leqslant \alpha$
10. $(s,\alpha) \in stab(S) \;\Rightarrow$ if $t > \alpha$, $t' \geqslant \alpha$, $a \in \Sigma$ and
    $w \in T\hat{\Sigma}_{\leqslant}^{*}$ is such that $w = \langle(t,a)\rangle^\frown w'$,
    then $(s^\frown w,\alpha',X) \in S \Rightarrow (\underline{s^\frown(w + (t'-t)}),\gamma,X) \in S$,
    where$\gamma \geqslant \alpha' + (t'-t)$
11. $(s,\alpha,X) \in S \;\wedge\; Y \subseteq X \;\Rightarrow\; (s,\alpha,Y) \in S$
12. $(s,\alpha,X) \in S \wedge \exists\, Y \in \mathsf{P}(\Sigma) \bullet (\forall\, a \in Y,$
    $\exists\, t \geqslant \alpha \bullet (s^\frown\langle(t,a)\rangle,\emptyset) \notin fail(S)) \;\Rightarrow\; (s,\alpha,X \cup Y) \in S$
13. $(s,\infty) \in stab(S) \wedge X \in \mathsf{P}(\Sigma) \Rightarrow (s,\infty,X) \in S$

The Complete Metric on TM_{FS}^{*}

If $S \in TM_{FS}^{*}$ and $t \in [0,\infty)$, we define

$$\begin{aligned} S(t) \;\widehat{=}\; & \{(s,\alpha,X) \in S \mid \alpha < t\} \\ & \cup\{(s,\infty,X) \mid end(s) < t \\ & \quad \wedge\ \exists\, \alpha \geqslant t \text{ such that } (s,\alpha) \in stab(S) \wedge X \in \mathsf{P}(\Sigma)\}. \end{aligned}$$

The complete metric on TM^*_{FS} is defined:

$$d(S_1, S_2) \ \widehat{=}\ inf\{2^{-t} \mid S_1(t) = S_2(t)\}$$

The Semantic Function $\mathcal{E}^*_T$

We now define the semantic function $\mathcal{E}^*_T : TCSP \rightarrow TM^*_{FS}$.

$$\begin{array}{rcl}
\mathcal{E}^*_T[\![\bot]\!] & \widehat{=} & \{(\langle\rangle, \infty, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}^*_T[\![STOP]\!] & \widehat{=} & \{(\langle\rangle, 0, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}^*_T[\![SKIP]\!] & \widehat{=} & \{(\langle\rangle, 0, X) \mid \checkmark \notin X\} \\
 & & \cup\{(\langle(0, \hat{\checkmark})\rangle, 0, X) \mid X \in \mathbf{P}(\Sigma)\} \\
 & & \cup\{(\langle(t, \checkmark)\rangle, t, X) \mid t \geq 0 \ \wedge \ \checkmark \notin X\} \\
\mathcal{E}^*_T[\![WAIT\ t]\!] & \widehat{=} & \{(\langle\rangle, t, X) \mid \checkmark \notin X\} \\
 & & \cup\{(\langle(t, \hat{\checkmark})\rangle, t, X) \mid X \in \mathbf{P}(\Sigma)\} \\
 & & \cup\{(\langle(t', \checkmark)\rangle, t', X) \mid t' \geqslant t \ \wedge \ X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}^*_T[\![a \rightarrow P]\!] & \widehat{=} & \{(\langle\rangle, 0, X) \mid a \notin X\} \\
 & & \cup\{(\langle(0, \hat{a})\rangle.(s + \delta), \alpha + \delta, X) \mid (s, \alpha, X) \in \mathcal{E}^*_T[\![P]\!]\} \\
 & & \cup\{(\langle(t, a)\rangle^\frown(s + (t + \delta)), \alpha + t + \delta, X) \mid t \geq 0 \\
 & & \ \wedge (s, \alpha, X) \in \mathcal{E}^*_T[\![P]\!]\} \\
\mathcal{E}^*_T[\![P \Box Q]\!] & \widehat{=} & SUP(\{(\langle\rangle, \alpha, X) \mid (\langle\rangle, \alpha) \in stab(\mathcal{E}^*_T[\![P]\!] \cup \mathcal{E}^*_T[\![Q]\!]) \\
 & & \qquad \wedge (\langle\rangle, X) \in fail(\mathcal{E}^*_T[\![P]\!]) \cap fail(\mathcal{E}^*_T[\![Q]\!])\} \\
 & & \qquad \cup\{(s, \alpha, X) \in \mathcal{E}^*_T[\![P]\!] \cup \mathcal{E}^*_T[\![Q]\!] \mid s \neq \langle\rangle\}) \\
\mathcal{E}^*_T[\![P \sqcap Q]\!] & \widehat{=} & SUP(\mathcal{E}^*_T[\![P]\!] \cup \mathcal{E}^*_T[\![Q]\!]) \\
\mathcal{E}^*_T[\![P \parallel Q]\!] & \widehat{=} & SUP(\{((s_P \vee s_Q), max\{\alpha_P, \alpha_Q\}, X_P \cup X_Q) \mid (s_P, \alpha_P, X_P) \in \mathcal{E}^*_T[\![P]\!] \\
 & & \qquad \wedge(s_Q, \alpha_Q, X_Q) \in \mathcal{E}^*_T[\![Q]\!] \wedge \bar{s}_P = \bar{s}_Q\}) \\
\mathcal{E}^*_T[\![P\ {}_X\!\parallel_Y Q]\!] & \widehat{=} & \{(s, max\{\alpha_P, \alpha_Q\}, Z_P \cup Z_Q \cup Z) \mid \exists(s_P, \alpha_P, Z_P) \in \mathcal{E}^*_T[\![P]\!] \\
 & & \wedge (s_Q, \alpha_Q, Z_Q) \in \mathcal{E}^*_T[\![Q]\!] \ with\ Z_P \subseteq X \wedge Z_Q \subseteq Y \bullet \\
 & & s \in (s_P\ {}_X\!\parallel_Y s_Q) \wedge Z \subseteq (\Sigma - (X \cup Y))\} \\
\mathcal{E}^*_T[\![P \mid\mid\mid Q]\!] & \widehat{=} & SUP(\{(s, max\{\alpha_P, \alpha_Q\}, X) \mid \exists(u, \alpha_P, X) \in \mathcal{E}^*_T[\![P]\!] \\
 & & \qquad \wedge (v, \alpha_Q, X) \in \mathcal{E}^*_T[\![Q]\!] \bullet s \in Tmerge(u, v)\}) \\
\mathcal{E}^*_T[\![P\,; Q]\!] & \widehat{=} & CL_{\cong}(SUP(\{(s, \alpha, X) \mid (s, \alpha, X \cup \{\checkmark\}) \in \mathcal{E}^*_T[\![P]\!] \ \wedge \ \checkmark \notin \sigma(s)\} \\
 & & \qquad \cup\{(s^\frown(w + t), \alpha + t, X) \mid s^\frown\langle(t, \hat{\checkmark})\rangle \in traces(\mathcal{E}^*_T[\![P]\!]) \\
 & & \qquad \ \wedge \checkmark \notin \sigma(s) \ \wedge \ (w, \alpha, X) \in \mathcal{E}^*_T[\![Q]\!]\}))
\end{array}$$

$$
\begin{aligned}
\mathcal{E}_T^*[\![P \setminus X]\!] \quad &\hat{=} \quad \{(s \setminus X, \alpha, Y) \mid s \text{ is } X\text{-active in } \mathcal{E}_T^*[\![P]\!] \\
&\qquad \land\, \alpha = sup\{\beta \mid \exists (w, \beta) \in \mathcal{E}_T^*[\![P]\!] \bullet \\
&\qquad w \text{ is } X\text{-active} \,\land\, w \setminus X = s \setminus X\} \\
&\qquad \land\, (\alpha < \infty \,\land\, (s, X \cup Y) \in fail(\mathcal{E}_T^*[\![P]\!]) \\
&\qquad \lor\, (\alpha = \infty \,\land\, Y \in \mathbf{P}(\Sigma)))\} \\
&\qquad \text{where } s \text{ is } X\text{-active provided} \\
&\qquad s \text{ contains no element of the form } (t, a) \text{ for } a \in X \\
&\qquad \text{(all communications in } X \text{ are in the form } \hat{a}).
\end{aligned}
$$

$$
\mathcal{E}_T^*[\![f^{-1}(P)]\!] \quad \hat{=} \quad \{(s, \alpha, X) \mid (f(s), \alpha, f(X)) \in \mathcal{E}_T^*[\![P]\!]\}
$$

$$
\mathcal{E}_T^*[\![f(P)]\!] \quad \hat{=} \quad SUP(\{(f(s), \alpha, X) \mid (s, \alpha, f^{-1}(X)) \in \mathcal{E}_T^*[\![P]\!]\})
$$

$$
\begin{aligned}
\mathcal{E}_T^*[\![\mu P.F(P)]\!] \quad &\hat{=} \quad \text{The unique fixed point of the contraction mapping} \\
&\qquad \hat{C}(Q) = C(\mathit{WAIT}\ \delta\, ; Q), \text{ where } C \text{ is the mapping} \\
&\qquad \text{on } TM_{FS}^* \text{ represented by } F.
\end{aligned}
$$

We may add the following equations

$$
\mathcal{E}_T^*[\![\bigsqcap_{a \in A} P(a)]\!] \quad \hat{=} \quad SUP(\bigcup_{a \in A} \mathcal{E}_T^*[\![P(a)]\!])
$$

$$
\begin{aligned}
\mathcal{E}_T^*[\![a : A \longrightarrow P(a)]\!] \quad &\hat{=} \quad \{(\langle\rangle, 0, X) \mid A \cap X = \emptyset\} \\
&\qquad \cup \{(\langle(0, \hat{a})\rangle^\frown(s + \delta), \alpha + \delta, X) \mid \\
&\qquad\qquad a \in A \land (s, \alpha, X) \in \mathcal{E}_T^*[\![P(a)]\!]\} \\
&\qquad \cup \{(\langle(t, a)\rangle^\frown(s + t + \delta), \alpha + t + \delta, X) \mid \\
&\qquad\qquad a \in A \land t \geqslant 0 \land (s, \alpha, X) \in \mathcal{E}_T^*[\![P(a)]\!]\}
\end{aligned}
$$

whenever the set $\{P(a) \mid a \in A\}$ is uniformly bounded.

The timed failures model

The Evaluation Domain TM_F.

We formally define TM_F to be those subsets S of $T\Sigma^*_{\leqslant} \times RSET$ satisfying:

1. $\langle\rangle \in Traces(S)$

2. $(s^\frown w, \aleph) \in S \;\Rightarrow\; (s, \aleph \upharpoonright begin(w)) \in S$

3. $(s, \aleph) \in S \;\wedge\; s \cong w \;\Rightarrow\; (w, \aleph) \in S$

4. $$\begin{array}{lll} (s,\aleph) \in fail(S) & \Rightarrow & \exists \aleph' \in RSET \bullet \\ \wedge\, t \geqslant 0 & & \aleph \subseteq \aleph' \wedge (s, \aleph') \in fail(S) \wedge \\ & & (t' \leqslant t \wedge (t', a) \notin \aleph') \\ & & \Rightarrow (s \upharpoonright t'^\frown\langle(t', a)\rangle, \aleph' \upharpoonright t') \in fail(S) \end{array}$$

5. $\forall t \in [0, \infty),\ \exists\, n(t) \in \mathbf{N} \bullet \forall s \in traces(S), (end(s) \leqslant t \Rightarrow \#s \leqslant n(t))$

6. $(s, \aleph) \in S \wedge \aleph' \in RSET \wedge \aleph' \subseteq \aleph \;\Rightarrow\; (s, \aleph') \in S$

7. $(s^\frown w, \aleph) \in S \wedge \aleph' \in RSET \bullet end(s) \leqslant begin(\aleph') \;\wedge\; end(\aleph') \leqslant begin(w) \wedge$
   $(\forall (t, a) \in \aleph', (s.\langle(t, a)\rangle, \aleph \upharpoonright t) \notin S)$
   $\Rightarrow (s, \aleph \cup \aleph') \in S$

The Complete Metric on TM_F.

If $S \in TM_F$ and $t \in [0, \infty)$, we define

$$S(t) \;\;\widehat{=}\;\; \{(s, \aleph) \in S \mid end(s) < t \wedge end(\aleph) < t\}.$$

The complete metric on TM_F is defined:

$$d(S_1, S_2) \;\widehat{=}\; inf\{2^{-1} \mid S_1(t) = S_2(t)\}$$

The Semantic Function $\mathcal{F}_T$.

We now define the semantic function $\mathcal{F}_T : TCSP \rightarrow TM_F$.

$$
\begin{aligned}
\mathcal{F}_T\llbracket\bot\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid \aleph \in RSET\} \\
\mathcal{F}_T\llbracket STOP\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid \aleph \in RSET\} \\
\mathcal{F}_T\llbracket SKIP\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid \checkmark \notin \sigma(\aleph)\} \\
&\quad \cup\{(\langle(t,\checkmark)\rangle,\aleph_1 \cup \aleph_2) \mid t \geq 0 \wedge (I(\aleph_1) \subseteq [0,t) \wedge \checkmark \notin \sigma(\aleph_1)) \\
&\qquad \wedge\ I(\aleph_2) \subseteq [t,\infty)\} \\
\mathcal{F}_T\llbracket WAIT\ t\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid \aleph \cap ([t,\infty) \times \{\checkmark\}) = \emptyset\} \\
&\quad \cup\{(\langle(t',\checkmark)\rangle,\aleph_1 \cup \aleph_2 \cup \aleph_3) \mid t' \geqslant t \wedge I(\aleph_1) \subseteq [0,t) \\
&\qquad \wedge(I(\aleph_2) \subseteq [t,t') \wedge \checkmark \notin \sigma(\aleph_2)) \wedge I(\aleph_3) \subseteq [t',\infty)\} \\
\mathcal{F}_T\llbracket a \rightarrow P\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid a \notin \sigma(\aleph)\} \\
&\quad \cup\{(\langle(t,a)\rangle^\frown(s + (t+\delta)),\aleph_1 \cup \aleph_2 \cup (\aleph_3 + (t+\delta))) \mid t \geq 0 \\
&\qquad \wedge(I(\aleph_1) \subseteq [0,t) \wedge a \notin \sigma(\aleph_1)) \wedge I(\aleph_2) \subseteq [t,t+\delta) \\
&\qquad \wedge(s,\aleph_3) \in \mathcal{F}_T\llbracket P\rrbracket\} \\
\mathcal{F}_T\llbracket P \Box Q\rrbracket &\;\widehat{=}\; \{(\langle\rangle,\aleph) \mid (\langle\rangle,\aleph) \in \mathcal{F}_T\llbracket P\rrbracket \cap \mathcal{F}_T\llbracket Q\rrbracket\} \\
&\quad \cup\{(s,\aleph) \mid s \neq \langle\rangle\ \wedge\ (s,\aleph) \in \mathcal{F}_T\llbracket P\rrbracket \cup \mathcal{F}_T\llbracket Q\rrbracket \\
&\qquad\quad \wedge\ (\langle\rangle,\aleph \upharpoonright begin(s)) \in \mathcal{F}_T\llbracket P\rrbracket \cap \mathcal{F}_T\llbracket Q\rrbracket\} \\
\mathcal{F}_T\llbracket P \sqcap Q\rrbracket &\;\widehat{=}\; \mathcal{F}_T\llbracket P\rrbracket \cup \mathcal{F}_T\llbracket Q\rrbracket \\
\mathcal{F}_T\llbracket P \parallel Q\rrbracket &\;\widehat{=}\; \{(s,\aleph_P \cup \aleph_Q) \mid (s,\aleph_P) \in \mathcal{F}_T\llbracket P\rrbracket\ \wedge\ (s,\aleph_Q) \in \mathcal{F}_T\llbracket Q\rrbracket\} \\
\mathcal{F}_T\llbracket P\ {}_X\|_Y\ Q\rrbracket &\;\widehat{=}\; \{(s,\aleph_P \cup \aleph_Q \cup \aleph_Z) \mid \exists (s_P,\aleph_P) \in \mathcal{F}_T\llbracket P\rrbracket \\
&\quad \wedge\ (s_Q,\aleph_Q) \in \mathcal{F}_T\llbracket Q\rrbracket\ \mathit{with}\ \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y \bullet \\
&\quad s \in (s_P\ {}_X\|_Y\ s_Q) \wedge \sigma(\aleph_Z) \subseteq (\Sigma - (X \cup Y))\} \\
\mathcal{F}_T\llbracket P \,|||\, Q\rrbracket &\;\widehat{=}\; \{(s,\aleph) \mid \exists (u,\aleph) \in \mathcal{F}_T\llbracket P\rrbracket,\ (v,\aleph) \in \mathcal{F}_T\llbracket Q\rrbracket \\
&\quad \text{such that } s \in Tmerge(u,v)\}) \\
\mathcal{F}_T\llbracket P\,;Q\rrbracket &\;\widehat{=}\; CL_{\cong}(\{(s,\aleph) \mid \checkmark \notin \sigma(s) \\
&\qquad \wedge\ (\forall I \in TINT,\ (s,\aleph \cup (I \times \{\checkmark\})) \in \mathcal{F}_T\llbracket P\rrbracket)\} \\
&\quad \cup\{(s^\frown(w+t),\aleph_1 \cup (\aleph_2 + t)) \mid \checkmark \notin \sigma(s) \\
&\qquad \wedge\ (s^\frown\langle(t,\checkmark)\rangle,\aleph_1 \cup ([0,t) \times \{\checkmark\})) \in \mathcal{F}_T\llbracket P\rrbracket \\
&\qquad \wedge\ end(\aleph_1) \leqslant t)\ \wedge\ (w,\aleph_2) \in \mathcal{F}_T\llbracket Q\rrbracket\}) \\
\mathcal{F}_T\llbracket P \setminus X\rrbracket &\;\widehat{=}\; \{(s \setminus X,\aleph) \mid (s,\aleph \cup ([0,max\{end(s),end(\aleph)\}) \times X) \in \mathcal{F}_T\llbracket P\rrbracket\}
\end{aligned}
$$

$$\begin{aligned}
\mathcal{F}_T[\![f^{-1}(P)]\!] &\mathrel{\hat{=}} \{(s,\aleph) \mid (f(s), f(\aleph)) \in \mathcal{F}_T[\![P]\!]\}\\
\mathcal{F}_T[\![f(P)]\!] &\mathrel{\hat{=}} \{(f(s),\aleph) \mid (s, f^{-1}(\aleph)) \in \mathcal{F}_T[\![P]\!]\}\\
\mathcal{F}_T[\![\mu\, X \bullet F(X)]\!] &\mathrel{\hat{=}} \text{The unique fixed point of the contraction mapping}\\
&\qquad \hat{C}(Q) = C(\mathit{WAIT}\;\delta\,; Q)\text{, where } C \text{ is the mapping}\\
&\qquad \text{on } TM_F \text{ represented by } F.
\end{aligned}$$

We may also add the following equations:

$$\begin{aligned}
\mathcal{F}_T[\![\bigsqcap_{a\in A} P(a)]\!] &\mathrel{\hat{=}} \bigcup_{a\in A} \mathcal{F}_T[\![P(a)]\!]\\
\mathcal{F}_T[\![a : A \to P_a]\!] &\mathrel{\hat{=}} \{(\langle\rangle,\aleph) \mid A \cap \sigma(\aleph) = \emptyset\}\\
&\quad \cup\{(\langle(t,a)\rangle^\frown(s + (t+\delta)),\aleph) \mid\\
&\qquad\qquad a \in A \wedge t \geqslant 0 \wedge A \cap \sigma(\aleph \upharpoonright t) = \emptyset\\
&\qquad\qquad \wedge (s, \aleph \dot{-} (t+\delta)) \in \mathcal{F}_T[\![P(a)]\!]\}
\end{aligned}$$

provided the set of processes $\{P(a) \mid a \in A\}$ is uniformly bounded.

The timed failures stability model

The Evaluation Domain TM_{FS}.

We formally define TM_{FS} to be those subsets S of $(T\Sigma)^*_\leqslant \times [0,\infty] \times RSET$ satisfying the following 12 axioms:

1. $\langle\rangle \in \mathit{Traces}(S)$

2. $(s^\frown w, \aleph) \in \mathit{fail}(S) \Rightarrow (s, \aleph \upharpoonright \mathit{begin}(w)) \in \mathit{fail}(S)$

3. $(s, \alpha, \aleph) \in S \;\wedge\; s \cong w \Rightarrow (w, \alpha, \aleph) \in S$

4. $$\begin{aligned}
(s,\aleph) \in \mathit{fail}(S) \quad &\Rightarrow \quad \exists \aleph' \in RSET \bullet\\
\wedge\; t \geq 0 \quad &\qquad \aleph \subseteq \aleph' \wedge (s,\aleph') \in \mathit{fail}(S) \wedge\\
&\qquad (t' \leqslant t \wedge (t',a) \notin \aleph') \Rightarrow (s \upharpoonright t'^\frown\langle(t',a)\rangle, \aleph' \upharpoonright t') \in \mathit{fail}(S)
\end{aligned}$$
5. $t \in [0,\infty) \Rightarrow \;\exists\, n(t) \in \mathbf{N}$ such that $\forall s \in \mathit{Traces}(S)$,
   $\mathit{end}(s) \leqslant t \Rightarrow \#s \leqslant n(t)$

6. $(s, \alpha, \aleph), (s, \beta, \aleph) \in S \Rightarrow \;\alpha = \beta$

7. $(s, \alpha, \aleph) \in S \;\Rightarrow\; end(s) \leqslant \alpha$

8. $(s, \alpha, \aleph) \in S \;\wedge\; (s.\langle(t, a)\rangle, \aleph) \in fail(S) \;\wedge\; t > t' \geq \alpha \;\wedge\; t \geqslant end(\aleph) \;\Rightarrow (t', a) \notin \aleph$

9. $(s, \alpha, \aleph) \in S \;\Rightarrow$ if $t > \alpha$, $t' \geqslant \alpha$, $a \in \Sigma$ and
$w \in (T\Sigma)^*_\leq$ is such that $w = \langle(t, a)\rangle.w'$, then
$(s.w, \alpha', \aleph') \in S \;\wedge\; \aleph \subseteq \aleph' \upharpoonright t \;\Rightarrow$
$\exists\, \gamma \geqslant \alpha' + (t' - t) \bullet$
$(s.(w + (t' - t)), \gamma, \aleph_1 \cup \aleph_2 \cup (\aleph_3 + (t' - t))) \in S$,
where $\aleph_1 = \aleph' \upharpoonright \alpha$, $\aleph_2 = [\alpha, t') \times \Sigma(\aleph' \cap ([\alpha, t) \times \Sigma))$,
and $\aleph_3 = \aleph' \cap ([t, \infty) \times \Sigma)$.

10. $(s, \alpha, \aleph) \in S \wedge \aleph' \in RSET$ such that $\aleph' \subseteq \aleph$
$\Rightarrow\; \exists\, \alpha' \geqslant \alpha$ such that $(s, \alpha', \aleph') \in S$

11. $(s.w, \alpha, \aleph) \in S \wedge \aleph' \in RSET$ is such that $end(s) \leqslant begin(\aleph') \wedge$
$end(\aleph') \leqslant begin(w) \;\wedge\; (\forall (t, a) \in \aleph', (s.\langle(t, a)\rangle, \aleph \upharpoonright t) \notin fail(S))$
$\Rightarrow\; (s.w, \alpha, \aleph \cup \aleph') \in S$

12. $(s, \alpha, \aleph) \in S \Rightarrow$
$(\forall\, I \in TINT,\;\; I \subseteq [\alpha, \infty) \Rightarrow (s, \alpha, \aleph \cup (I \times \Sigma(\aleph \cap ([\alpha, \infty) \times \Sigma)))) \in S)$

The Complete Metric on TM_{FS}.

If $S \in TM_{FS}$ an $t \in [0, \infty)$, we define

$$\begin{aligned} S(t) \;\;\widehat{=}\;\; & \{(s, \alpha, \aleph) \in S \mid \alpha < t \wedge end(\aleph) < t\} \\ & \cup\{(s, \infty, \aleph) \mid end(s) < t \wedge end(\aleph) < t \;\wedge\; \exists\, \alpha \geq t \bullet (s, \alpha, \aleph) \in S\}. \end{aligned}$$

The complete metric on TM_{FS} is defined:

$$d(S_1, S_2) \;\widehat{=}\; inf\{2^{-t} \mid S_1(t) = S_2(t)\}$$

The Semantic Function $\mathcal{E}_T$.

We now define the semantic function $\mathcal{E}_T : TCSP \to TM_{FS}$.

$$\mathcal{E}_T[\![\bot]\!] \;\;\widehat{=}\;\; \{(\langle\rangle, \infty, \aleph) \mid \aleph \in RSET\}$$

$$\mathcal{E}_T[\![STOP]\!] \;\;\widehat{=}\;\; \{(\langle\rangle, 0, \aleph) \mid \aleph \in RSET\}$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket SKIP \rrbracket \;\widehat{=}\;& \{(\langle\rangle, 0, \aleph) \mid \checkmark \notin \Sigma(\aleph)\} \\
& \cup\{(\langle(t,\checkmark)\rangle, t, \aleph_1 \cup \aleph_2) \mid t \geqslant 0 \wedge (I(\aleph_1) \subseteq [0,t) \wedge \checkmark \notin \Sigma(\aleph_1)) \\
& \qquad\qquad\qquad\qquad \wedge\; I(\aleph_2) \subseteq [t,\infty)\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket WAIT\ t \rrbracket \;\widehat{=}\;& \{(\langle\rangle, t, \aleph) \mid \aleph \cap ([t,\infty) \times \{\checkmark\}) = \emptyset\} \\
& \cup\{(\langle(t',\checkmark)\rangle, t', \aleph_1 \cup \aleph_2 \cup \aleph_3) \mid t' \geqslant t \wedge I(\aleph_1) \subseteq [0,t) \\
& \quad \wedge(I(\aleph_2) \subseteq [t,t') \wedge \checkmark \notin \Sigma(\aleph_2)) \wedge I(\aleph_3) \subseteq [t',\infty)\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket a \rightarrow P \rrbracket \;\widehat{=}\;& \{(\langle\rangle, 0, \aleph) \mid a \notin \Sigma(\aleph)\} \\
& \cup\{(\langle(t,a)\rangle^\frown(s + (t+\delta)), \alpha + t + \delta, \aleph) \mid \\
& t \geqslant 0 \;\wedge a \notin \Sigma(\aleph \upharpoonright t)) \wedge (s, \alpha, \aleph \mathbin{\dot{-}} (t+\delta)) \in \mathcal{E}_T\llbracket P \rrbracket\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket P \,\Box\, Q \rrbracket \;\widehat{=}\;& SUP(\{(\langle\rangle, \mathit{max}\{\alpha_P, \alpha_Q\}, \aleph) \mid (\langle\rangle, \alpha_P, \aleph) \in \mathcal{E}_T\llbracket P \rrbracket \\
& \qquad\qquad\qquad\qquad\qquad\qquad \wedge\; (\langle\rangle, \alpha_Q, \aleph) \in \mathcal{E}_T\llbracket Q \rrbracket\} \\
& \qquad \cup\{(s, \alpha, \aleph) \mid s \neq \langle\rangle \;\wedge\; (s,\alpha,\aleph) \in \mathcal{E}_T\llbracket P \rrbracket \cup \mathcal{E}_T\llbracket Q \rrbracket \\
& \qquad\; \wedge (\langle\rangle, \aleph \upharpoonright \mathit{begin}(s)) \in \mathit{fail}(\mathcal{E}_T\llbracket P \rrbracket) \cap \mathit{fail}(\mathcal{E}_T\llbracket Q \rrbracket)\})
\end{aligned}
$$

$$
\mathcal{E}_T\llbracket P \sqcap Q \rrbracket \;\widehat{=}\; SUP(\mathcal{E}_T\llbracket P \rrbracket \cup \mathcal{E}_T\llbracket Q \rrbracket)
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket P \parallel Q \rrbracket \;\widehat{=}\;& SUP(\{(s, \mathit{max}\{\alpha_P, \alpha_Q\}, \aleph_P \cup \aleph_Q) \mid \\
& \qquad (s, \alpha_P, \aleph_P) \in \mathcal{E}_T\llbracket P \rrbracket \wedge (s, \alpha_Q, \aleph_Q) \in \mathcal{E}_T\llbracket Q \rrbracket\})
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket P \;{}_X\!\parallel_Y Q \rrbracket \;\widehat{=}\;& SUP\{(s, \mathit{max}\{\alpha_P, \alpha_Q\}, \aleph_P \cup \aleph_Q \cup \aleph_Z) \mid \\
& \exists (s_P, \alpha_P, \aleph_P) \in \mathcal{E}_T\llbracket P \rrbracket, (s_Q, \alpha_Q, \aleph_Q) \in \mathcal{E}_T\llbracket Q \rrbracket \bullet \\
& \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y \wedge \\
& s \in (s_P \;{}_X\!\parallel_Y s_Q) \wedge \sigma(\aleph_Z) \subseteq (\Sigma - (X \cup Y))\} \\
& \text{where} \\
& v \;{}_X\!\parallel_Y w = \{s \in (T\Sigma)^*_{\leqslant} \mid s \in (X \cup Y)^* \wedge \\
& \qquad\qquad\qquad\qquad s \upharpoonright X = v \wedge s \upharpoonright Y = w\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T\llbracket P \mathbin{|||} Q \rrbracket \;\widehat{=}\;& SUP\;(\{(s, \mathit{max}\{\alpha_P, \alpha_Q\}, \aleph) \mid \exists (u, \alpha_P, \aleph) \in \mathcal{E}_T\llbracket P \rrbracket \\
& \qquad (v, \alpha_Q, \aleph) \in \mathcal{E}_T\llbracket Q \rrbracket \bullet s \in \mathit{Tmerge}(u, v)\})
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T[\![P\,;Q]\!] \quad &\mathrel{\widehat{=}} \quad CL_{\cong}(SUP(\{(s,\alpha,\aleph) \mid \checkmark \notin \sigma(s) \;\wedge\; \forall I \in TINT \bullet \\
&\qquad\qquad\qquad (s,\alpha,\aleph \cup (I \times \{\checkmark\})) \in \mathcal{E}_T[\![P]\!]\} \\
&\qquad \cup \{(s^\frown(w+t), \alpha+t, \aleph_1 \cup (\aleph_2+t)) \mid \\
&\qquad\quad \checkmark \notin \sigma(s) \;\wedge\; end(\aleph_1) \leqslant t \wedge\; (w,\alpha,\aleph_2) \in \mathcal{E}_T[\![Q]\!] \\
&\qquad\quad \wedge\, (s^\frown\langle(t,\checkmark)\rangle, \aleph_1 \cup ([0,t) \times \{\checkmark\})) \in fail(\mathcal{E}_T[\![P]\!])\}))
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T[\![P \setminus X]\!] \quad &\mathrel{\widehat{=}} \quad SUP(\{s \setminus X, \beta, \aleph) \mid \;\exists\, \alpha \geqslant \beta \geqslant end(s) \bullet \\
&\qquad\qquad (s,\alpha,\aleph \cup ([0, max\{\beta, end(\aleph)\}) \times X)) \in \mathcal{E}_T[\![P]\!]\})
\end{aligned}
$$

$$
\mathcal{E}_T[\![f^{-1}(P)]\!] \quad \mathrel{\widehat{=}} \quad \{(s,\alpha,\aleph) \mid (f(s),\alpha,f(\aleph)) \in \mathcal{E}_T[\![P]\!]\}
$$

$$
\mathcal{E}_T[\![f(P)]\!] \quad \mathrel{\widehat{=}} \quad SUP(\{(f(s),\alpha,\aleph) \mid (s,\alpha,f^{-1}(\aleph)) \in \mathcal{E}_T[\![P]\!]\})
$$

$\mathcal{E}_T[\![\mu\, p.F(p)]\!] \quad \mathrel{\widehat{=}} \quad$ The unique fixed point of the contraction mapping $\hat{C}(Q) = C(WAIT\,\delta\,;Q)$, where C is the mapping on TM_{FS} represented by F.

We may also add:

$$
\begin{aligned}
\mathcal{E}_T[\![a : A \rightarrow P(a)]\!] \quad &\mathrel{\widehat{=}} \quad \{(\langle\rangle, 0, \aleph) \mid A \cap \Sigma(\aleph) = \emptyset\} \\
&\qquad \cup\{(\langle(t,a)\rangle^\frown(s+(t+\delta)), \alpha+t+\delta, \aleph) \mid \\
&\qquad t \geqslant 0 \;\wedge A \cap \Sigma(\aleph \upharpoonright t) = \emptyset) \wedge (s,\alpha,\aleph \mathbin{\dot{-}} (t+\delta)) \in \mathcal{E}_T[\![P]\!]\} \\
&\qquad \text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}_T[\![\textstyle\bigsqcap P(a)]\!] \quad &\mathrel{\widehat{=}} \quad SUP(\textstyle\bigcup \mathcal{E}_T[\![P(a)]\!]) \\
&\qquad \text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}
\end{aligned}
$$

B.3 Semantics for *TCSP* with process variables

We give the semantic equations which map *CSP* with variables and *TCSP* with variables to the semantic domains defined in [Ree88].

CSP with variables

$$\begin{array}{lll} P & ::= & \bot \mid STOP \mid SKIP \mid a \rightarrow P \mid a : A \rightarrow P(a) \\ & & P \sqcap Q \mid \bigsqcap P(a) \mid P \,\Box\, Q \mid P \parallel Q \mid P\ {}_X\!\|_Y\ Q \\ & & P \,|||\, Q \mid P\,;Q \mid P \setminus A \mid f(P) \mid f^{-1}(P) \\ & & X \mid \mu X \bullet P \end{array}$$

TCSP with variables

$$\begin{array}{lll} P & ::= & \bot \mid STOP \mid SKIP \mid WAIT\,t \mid WAIT\,t\,;P \\ & & a \rightarrow P \mid a : A \rightarrow P(a) \mid P \sqcap Q \mid \bigsqcap P(a) \mid P \,\Box\, Q \\ & & P \parallel Q \mid P\ {}_X\!\|_Y\ Q \mid P \,|||\, Q \mid P\,;Q \mid P \setminus A \\ & & f(P) \mid f^{-1}(P) \mid X \mid \mu X \bullet P \end{array}$$

The Semantic Function $\mathcal{T}$.

The environment σ is of type $var \rightarrow M_T$.

We now define a semantic function $\mathcal{T} : CSP \rightarrow M_T$.

$$
\begin{array}{rcl}
\mathcal{T}\llbracket \bot \rrbracket \sigma & \widehat{=} & \{\langle\rangle\} \\
\mathcal{T}\llbracket STOP \rrbracket \sigma & \widehat{=} & \{\langle\rangle\} \\
\mathcal{T}\llbracket SKIP \rrbracket \sigma & \widehat{=} & \{\langle\rangle, \langle\checkmark\rangle\} \\
\mathcal{T}\llbracket a \rightarrow P \rrbracket \sigma & \widehat{=} & \{\langle\rangle\} \cup \{\langle a\rangle^\frown s \mid s \in \mathcal{T}\llbracket P \rrbracket \sigma\} \\
\mathcal{T}\llbracket a : A \longrightarrow P(a) \rrbracket \sigma & \widehat{=} & \{\langle\rangle\} \cup \{\langle a\rangle^\frown s \mid a \in A \wedge s \in \mathcal{T}\llbracket P(a) \rrbracket \sigma\} \\
\mathcal{T}\llbracket P \Box Q \rrbracket \sigma & \widehat{=} & \mathcal{T}\llbracket P \rrbracket \sigma \cup \mathcal{T}\llbracket Q \rrbracket \sigma \\
\mathcal{T}\llbracket P \sqcap Q \rrbracket \sigma & \widehat{=} & \mathcal{T}\llbracket P \rrbracket \sigma \cup \mathcal{T}\llbracket Q \rrbracket \sigma \\
\mathcal{T}\llbracket \bigsqcap P(a) \rrbracket \sigma & \widehat{=} & \bigcup \mathcal{T}\llbracket P(a) \rrbracket \sigma \\
\mathcal{T}\llbracket P \parallel Q \rrbracket \sigma & \widehat{=} & \mathcal{T}\llbracket P \rrbracket \sigma \cap \mathcal{T}\llbracket Q \rrbracket \sigma \\
\mathcal{T}\llbracket P \;{}_X\!\parallel_Y Q \rrbracket \sigma & \widehat{=} & \{s \mid s \upharpoonright X \in \mathcal{T}\llbracket P \rrbracket \sigma \;\wedge\; s \upharpoonright Y \in \mathcal{T}\llbracket Q \rrbracket \sigma \\
 & & \qquad \wedge\; s \upharpoonright (X \cup Y) = s\} \\
\mathcal{T}\llbracket P \mid\mid\mid Q \rrbracket \sigma & \widehat{=} & \{s \mid \exists\, u \in \mathcal{T}\llbracket P \rrbracket \sigma,\; v \in \mathcal{T}\llbracket Q \rrbracket \sigma \bullet \\
 & & \qquad s \in Merge(u, v)\} \\
\mathcal{T}\llbracket P ; Q \rrbracket \sigma & \widehat{=} & \{s \mid s \in \mathcal{T}\llbracket P \rrbracket \sigma \wedge \checkmark \notin s\} \\
 & & \cup\, \{s^\frown t \mid \mathrm{s}^\frown\langle\checkmark\rangle \in \mathcal{T}\llbracket P \rrbracket \sigma \;\wedge\; t \in \mathcal{T}\llbracket Q \rrbracket \sigma\} \\
\mathcal{T}\llbracket P \setminus X \rrbracket \sigma & \widehat{=} & \{s \setminus X \mid s \in \mathcal{T}\llbracket P \rrbracket \sigma\} \\
\mathcal{T}\llbracket f^{-1}(P) \rrbracket \sigma & \widehat{=} & \{s \mid f(s) \in \mathcal{T}\llbracket P \rrbracket \sigma\} \\
\mathcal{T}\llbracket X \rrbracket \sigma & \widehat{=} & \sigma(X) \\
\mathcal{T}\llbracket \mu\, X \bullet P \rrbracket \sigma & \widehat{=} & \text{The least fixed point of the continuous mapping } \lambda\, Y \bullet \mathcal{T}\llbracket P \rrbracket(\sigma[Y/X]) \text{ with respect to} \\
 & & \text{the complete partial order } \sqsubseteq \text{ on } M_{\mathcal{T}}. \\
 & \textit{or} & \\
\mathcal{T}\llbracket \mu\, X \bullet P \rrbracket \sigma & \widehat{=} & \text{The unique fixed point of the contraction mapping } \lambda\, Y \bullet \mathcal{T}\llbracket P \rrbracket(\sigma[Y/X]) \text{ with respect} \\
 & & \text{to the complete metric } d \text{ on } M_{\mathcal{T}}.
\end{array}
$$

The Semantic Function $\mathcal{F}$.

We now define the semantic function $\mathcal{F} : CSP \rightarrow M_F$.

$$\begin{aligned}
\mathcal{F}\llbracket \bot \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{F}\llbracket STOP \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{F}\llbracket SKIP \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid \checkmark \notin X\} \cup \{(\langle\checkmark\rangle, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{F}\llbracket a \rightarrow P \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid a \notin X\} \\
&\qquad \cup\{(\langle a\rangle^\frown s, X) \mid (s, X) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
\mathcal{F}\llbracket a : A \rightarrow P(a) \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid A \cap X = \emptyset\} \\
&\qquad \cup\{(\langle a\rangle^\frown s, X) \mid a \in A \wedge (s, X) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
\mathcal{F}\llbracket P \Box Q \rrbracket \sigma &\;\widehat{=}\; \{(\langle\rangle, X) \mid (\langle\rangle, X) \in \mathcal{F}\llbracket P \rrbracket \sigma \cap \mathcal{F}\llbracket Q \rrbracket \sigma\} \\
&\qquad \cup\{(s, X) \in \mathcal{F}\llbracket P \rrbracket \sigma \cup \mathcal{F}\llbracket Q \rrbracket \sigma \mid s \neq \langle\rangle\} \\
\mathcal{F}\llbracket P \sqcap Q \rrbracket \sigma &\;\widehat{=}\; \mathcal{F}\llbracket P \rrbracket \sigma \cup \mathcal{F}\llbracket Q \rrbracket \sigma \\
\mathcal{F}\llbracket \bigsqcap P(a) \rrbracket \sigma &\;\widehat{=}\; \bigcup \mathcal{F}\llbracket P(a) \rrbracket \sigma \\
\mathcal{F}\llbracket P \parallel Q \rrbracket \sigma &\;\widehat{=}\; \{(s, X \cup Y) \mid (s, X) \in \mathcal{F}\llbracket P \rrbracket \sigma \\
&\qquad\qquad \wedge (s, Y) \in \mathcal{F}\llbracket Q \rrbracket \sigma\} \\
\mathcal{F}\llbracket P \;{}_X\!\parallel_Y Q \rrbracket \sigma &\;\widehat{=}\; \{(s, Z_P \cup Z_Q \cup Z) \mid s \upharpoonright (X \cup Y) = s \wedge \\
&\qquad Z_P \subseteq X \wedge Z_Q \subseteq Y \wedge Z \subseteq (\Sigma - (X \cup Y)) \\
&\qquad \wedge (s \upharpoonright X, Z_P) \in \mathcal{F}\llbracket P \rrbracket \sigma \\
&\qquad \wedge (s \upharpoonright Y, Z_Q) \in \mathcal{F}\llbracket Q \rrbracket \sigma\} \\
\mathcal{F}\llbracket P \,|||\, Q \rrbracket \sigma &\;\widehat{=}\; \{(s, X) \mid \exists\, u, v \in \Sigma^* \bullet (u, X) \in \mathcal{F}\llbracket P \rrbracket \sigma \\
&\qquad \wedge (v, X) \in \mathcal{F}\llbracket Q \rrbracket \sigma \wedge s \in Merge(u, v)\} \\
\mathcal{F}\llbracket P \,;\, Q \rrbracket \sigma &\;\widehat{=}\; \{(s, X) \mid \checkmark \notin s \wedge (s, X \cup \{\checkmark\}) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
&\qquad \cup\{(s^\frown w, X) \mid \checkmark \notin s \wedge (s^\frown\langle\checkmark\rangle, \emptyset) \in \mathcal{F}\llbracket P \rrbracket \sigma \\
&\qquad\qquad \wedge (w, X) \in \mathcal{F}\llbracket Q \rrbracket \sigma\} \\
\mathcal{F}\llbracket P \setminus X \rrbracket \sigma &\;\widehat{=}\; \{s \setminus X, Y) \mid (s, X \cup Y) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
&\qquad \cup\{(s, Y) \mid \forall\, n \geq \#s,\ \exists\, w_n \in Traces(\mathcal{F}\llbracket P \rrbracket \sigma) \bullet \\
&\qquad\qquad w_n < w_{n+1} \wedge s = w_n \setminus X\} \\
\mathcal{F}\llbracket f^{-1}(P) \rrbracket \sigma &\;\widehat{=}\; \{(s, X) \mid (f(s), f(X)) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
\mathcal{F}\llbracket f(P) \rrbracket \sigma &\;\widehat{=}\; \{(f(s), X) \mid (s, f^{-1}(X)) \in \mathcal{F}\llbracket P \rrbracket \sigma\} \\
\mathcal{F}\llbracket X \rrbracket \sigma &\;\widehat{=}\; \sigma(X) \\
\mathcal{F}\llbracket \mu\, X \bullet P) \rrbracket \sigma &\;\widehat{=}\; \text{The unique fixed point of the contraction} \\
&\qquad \text{mapping } \lambda\, Y \bullet \mathcal{F}\llbracket P \rrbracket (\sigma[Y/X]) \text{ on } M_F.
\end{aligned}$$

The Semantic Function $\mathcal{E}_T^*$.

We now define the semantic function $\mathcal{E}_T^* : TCSP \rightarrow TM_{FS}^*$.

$$\begin{aligned}
\mathcal{E}_T^*\llbracket\bot\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, \infty, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}_T^*\llbracket STOP\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, 0, X) \mid X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}_T^*\llbracket SKIP\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, 0, X) \mid \checkmark \notin X\} \\
&\quad \cup\{(\langle(0, \hat{\checkmark})\rangle, 0, X) \mid X \in \mathbf{P}(\Sigma)\} \\
&\quad \cup\{(\langle(t, \checkmark)\rangle, t, X) \mid t \geq 0 \;\wedge\; \checkmark \notin X\} \\
\mathcal{E}_T^*\llbracket WAIT\ t\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, t, X) \mid \checkmark \notin X\} \\
&\quad \cup\{(\langle(t, \hat{\checkmark})\rangle, t, X) \mid X \in \mathbf{P}(\Sigma)\} \\
&\quad \cup\{(\langle(t', \checkmark)\rangle, t', X) \mid t' \geq t \;\wedge\; X \in \mathbf{P}(\Sigma)\} \\
\mathcal{E}_T^*\llbracket WAIT\ t\ ;\ P\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, t, X) \mid \checkmark \notin X\} \\
&\quad \cup\{(s + t, \alpha + t, X) \mid (s, \alpha, X) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho\} \\
\mathcal{E}_T^*\llbracket a \longrightarrow P\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, 0, X) \mid a \notin X\} \\
&\quad \cup\{(\langle(0, \hat{a})\rangle^\frown(s + \delta), \alpha + \delta, X) \mid (s, \alpha, X) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho\} \\
&\quad \cup\{(\langle(t, a)\rangle^\frown(s + (t + \delta)), \alpha + t + \delta, X) \mid t \geq 0 \\
&\qquad \wedge (s, \alpha, X) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho\} \\
\mathcal{E}_T^*\llbracket a : A \rightarrow P(a)\rrbracket\rho \;&\widehat{=}\; \{(\langle\rangle, 0, X) \mid A \cap X = \emptyset\} \\
&\quad \cup\{(\langle(0, \hat{a})\rangle^\frown(s + \delta), \alpha + \delta, X) \mid a \in A \wedge \\
&\qquad\qquad\qquad (s, \alpha, X) \in \mathcal{E}_T^*\llbracket P(a)\rrbracket\rho\} \\
&\quad \cup\{(\langle(t, a)\rangle^\frown(s + (t + \delta)), \alpha + t + \delta, X) \mid \\
&\quad a \in A \wedge t \geq 0 \wedge\ (s, \alpha, X) \in \mathcal{E}_T^*\llbracket P(a)\rrbracket\rho\} \\
&\quad \text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded} \\
\mathcal{E}_T^*\llbracket P \Box Q\rrbracket\rho \;&\widehat{=}\; SUP(\{(\langle\rangle, \alpha, X) \mid (\langle\rangle, \alpha) \in stab(\mathcal{E}_T^*\llbracket P\rrbracket\rho \cup \mathcal{E}_T^*\llbracket Q\rrbracket\rho) \\
&\qquad\qquad \wedge (\langle\rangle, X) \in fail(\mathcal{E}_T^*\llbracket P\rrbracket\rho) \cap fail(\mathcal{E}_T^*\llbracket Q\rrbracket\rho)\} \\
&\qquad\quad \cup\{(s, \alpha, X) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho \cup \mathcal{E}_T^*\llbracket Q\rrbracket\rho \mid s \neq \langle\rangle\}) \\
\mathcal{E}_T^*\llbracket P \sqcap Q\rrbracket\rho \;&\widehat{=}\; SUP(\mathcal{E}_T^*\llbracket P\rrbracket\rho \cup \mathcal{E}_T^*\llbracket Q\rrbracket\rho) \\
\mathcal{E}_T^*\llbracket \textstyle\bigsqcap P(a)\rrbracket\rho \;&\widehat{=}\; SUP(\textstyle\bigcup \mathcal{E}_T^*\llbracket P(a)\rrbracket\rho) \\
&\quad \text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded} \\
\mathcal{E}_T^*\llbracket P \parallel Q\rrbracket\rho \;&\widehat{=}\; SUP(\{((s_P \vee s_Q), max\{\alpha_P, \alpha_Q\}, X_P \cup X_Q) \mid \\
&\quad (s_P, \alpha_P, X_P) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho \wedge (s_Q, \alpha_Q, X_Q) \in \mathcal{E}_T^*\llbracket Q\rrbracket\rho \\
&\qquad\qquad\qquad\qquad \wedge \tilde{s}_P = \tilde{s}_Q\})
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T^*\llbracket P \;{}_X\|_Y\; Q\rrbracket\rho \;&\widehat{=}\; \{(s, max\{\alpha_P, \alpha_Q\}, Z_P \cup Z_Q \cup Z) \mid \\
&\exists (s_P, \alpha_P, Z_P) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho \wedge (s_Q, \alpha_Q, Z_Q) \in \mathcal{E}_T^*\llbracket Q\rrbracket\rho \bullet \\
&Z_P \subseteq X \wedge Z_Q \subseteq Y \wedge \\
&s \in (s_P \;{}_X\|_Y\; s_Q) \wedge Z \subseteq (\Sigma - (X \cup Y))\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T^*\llbracket P \mid\mid\mid Q\rrbracket\rho \;&\widehat{=}\; SUP(\{(s, max\{\alpha_P, \alpha_Q\}, X) \mid \exists (u, \alpha_P, X) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho, \\
&\qquad (v, \alpha_Q, X) \in \mathcal{E}_T^*\llbracket Q\rrbracket\rho \bullet s \in Tmerge(u, v)\})
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T^*\llbracket P \,;\, Q\rrbracket\rho \;&\widehat{=}\; CL_{\cong}(SUP(\{(s, \alpha, X) \mid (s, \alpha, X \cup \{\checkmark\}) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho \;\wedge\; \checkmark \notin \Sigma(s)\} \\
&\qquad \cup \{(s^\frown(w + t), \alpha + t, X) \mid s^\frown\langle(t, \hat{\checkmark})\rangle \in Traces(\mathcal{E}_T^*\llbracket P\rrbracket\rho) \\
&\qquad\;\; \wedge \checkmark \notin \Sigma(s) \;\wedge\; (w, \alpha, X) \in \mathcal{E}_T^*\llbracket Q\rrbracket\rho\}))
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T^*\llbracket P \setminus X\rrbracket\rho \;&\widehat{=}\; \{(s \setminus X, \alpha, Y) \mid s \text{ is } X\text{-active in } \mathcal{E}_T^*\llbracket P\rrbracket\rho \\
&\wedge \alpha = sup\{\beta \mid \exists (w, \beta) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho \textit{ such that} \\
&w \text{ is } X\text{-active} \;\wedge\; w \setminus X = s \setminus X\} \\
&\wedge (\alpha < \infty \;\wedge\; (s, X \cup Y) \in fail(\mathcal{E}_T^*\llbracket P\rrbracket\rho) \\
&\vee (\alpha = \infty \;\wedge\; Y \in \mathbf{P}(\Sigma)))\} \\
&\text{where } s \text{ is } X\text{-active provided} \\
&s \text{ contains no element of the form } (t, a) \text{ for } a \in X \\
&\text{(all communications in } X \text{ are in the form } \hat{a}\text{).}
\end{aligned}$$

$$\mathcal{E}_T^*\llbracket f^{-1}(P)\rrbracket\rho \;\widehat{=}\; \{(s, \alpha, X) \mid (f(s), \alpha, f(X)) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho\}$$

$$\mathcal{E}_T^*\llbracket f(P)\rrbracket\rho \;\widehat{=}\; SUP(\{(f(s), \alpha, X) \mid (s, \alpha, f^{-1}(X)) \in \mathcal{E}_T^*\llbracket P\rrbracket\rho\})$$

$$\mathcal{E}_T^*\llbracket X\rrbracket\rho \;\widehat{=}\; \rho(X)$$

$$\begin{aligned}
\mathcal{E}_T^*\llbracket \mu\, X \bullet P\rrbracket\rho \;&\widehat{=}\; \text{The unique fixed point of the contraction} \\
&\text{mapping } \lambda\, Y \bullet \mathcal{E}_T^*\llbracket P\rrbracket(\rho[W_\delta(Y)/X]),\text{ where} \\
&W_\delta \text{ is the mapping on } TM_{FS}^* \text{ corresponding to} \\
&\lambda\, Y \bullet WAIT\,\delta\,;\, Y.
\end{aligned}$$

The Semantic Function $\mathcal{E}$.

We now define the semantic function $\mathcal{E} : TCSP \rightarrow TM_{FS}$.

$$\mathcal{E}_T[\![\bot]\!]\rho \;\hat{=}\; \{(\langle\rangle, \infty, \aleph) \mid \aleph \in RSET\}$$

$$\mathcal{E}_T[\![STOP]\!]\rho \;\hat{=}\; \{(\langle\rangle, 0, \aleph) \mid \aleph \in RSET\}$$

$$\begin{aligned}\mathcal{E}_T[\![SKIP]\!]\rho \;\hat{=}\;& \{(\langle\rangle, 0, \aleph) \mid \checkmark \notin \Sigma(\aleph)\}\\ &\cup\{(\langle(t,\checkmark)\rangle, t, \aleph_1 \cup \aleph_2) \mid t \geqslant 0 \wedge (I(\aleph_1) \subseteq [0,t) \wedge \checkmark \notin \Sigma(\aleph_1))\\ &\qquad\qquad\qquad\qquad \wedge\; I(\aleph_2) \subseteq [t,\infty)\}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_T[\![WAIT\ t]\!]\rho \;\hat{=}\;& \{(\langle\rangle, t, \aleph) \mid \aleph \cap ([t,\infty) \times \{\checkmark\}) = \emptyset\}\\ &\cup\{(\langle(t',\checkmark)\rangle, t', \aleph_1 \cup \aleph_2 \cup \aleph_3) \mid t' \geqslant t \wedge I(\aleph_1) \subseteq [0,t)\\ &\quad \wedge(I(\aleph_2) \subseteq [t,t') \wedge \checkmark \notin \Sigma(\aleph_2)) \wedge I(\aleph_3) \subseteq [t',\infty)\}\end{aligned}$$

$$\mathcal{E}_T[\![WAIT\ t\,;P]\!]\rho \;\hat{=}\; \{(s,\alpha,\aleph) \mid (s-t, \alpha-t, \aleph \dot{-} t) \in \mathcal{E}_T[\![P]\!]\rho\}$$

$$\begin{aligned}\mathcal{E}_T[\![a \rightarrow P]\!]\rho \;\hat{=}\;& \{(\langle\rangle, 0, \aleph) \mid a \notin \Sigma(\aleph)\}\\ &\cup\{(\langle(t,a)\rangle^\frown(s+(t+\delta)), \alpha+t+\delta, \aleph) \mid\\ &t \geqslant 0\ \wedge a \notin \Sigma(\aleph \upharpoonright t)) \wedge (s,\alpha,\aleph \dot{-} (t+\delta)) \in \mathcal{E}_T[\![P]\!]\rho\}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_T[\![a : A \rightarrow P(a)]\!]\rho \;\hat{=}\;& \{(\langle\rangle, 0, \aleph) \mid A \cap \Sigma(\aleph) = \emptyset\}\\ &\cup\{(\langle(t,a)\rangle^\frown(s+(t+\delta)), \alpha+t+\delta, \aleph) \mid\\ &t \geqslant 0\ \wedge A \cap \Sigma(\aleph \upharpoonright t) = \emptyset) \wedge (s,\alpha,\aleph \dot{-} (t+\delta)) \in \mathcal{E}_T[\![P]\!]\rho\}\\ &\text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_T[\![P \,\Box\, Q]\!]\rho \;\hat{=}\;& SUP(\{(\langle\rangle, max\{\alpha_P,\alpha_Q\}, \aleph) \mid (\langle\rangle,\alpha_P,\aleph) \in \mathcal{E}_T[\![P]\!]\rho\\ &\qquad\qquad\qquad\qquad\qquad \wedge\ (\langle\rangle,\alpha_Q,\aleph) \in \mathcal{E}_T[\![Q]\!]\rho\}\\ &\qquad \cup\{(s,\alpha,\aleph) \mid s \neq \langle\rangle\ \wedge\ (s,\alpha,\aleph) \in \mathcal{E}_T[\![P]\!]\rho \cup \mathcal{E}_T[\![Q]\!]\rho\\ &\qquad\quad \wedge (\langle\rangle, \aleph \upharpoonright begin(s)) \in fail(\mathcal{E}_T[\![P]\!]\rho) \cap fail(\mathcal{E}_T[\![Q]\!]\rho)\})\end{aligned}$$

$$\mathcal{E}_T[\![P \sqcap Q]\!]\rho \;\hat{=}\; SUP(\mathcal{E}_T[\![P]\!]\rho \cup \mathcal{E}_T[\![Q]\!]\rho)$$

$$\begin{aligned}\mathcal{E}_T[\![\textstyle\bigsqcap P(a)]\!]\rho \;\hat{=}\;& SUP(\textstyle\bigcup \mathcal{E}_T[\![P(a)]\!]\rho)\\ &\text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}\end{aligned}$$

$$\begin{aligned}\mathcal{E}_T[\![P \parallel Q]\!]\rho \;\hat{=}\;& SUP(\{(s, max\{\alpha_P,\alpha_Q\}, \aleph_P \cup \aleph_Q) \mid\\ &\qquad (s,\alpha_P,\aleph_P) \in \mathcal{E}_T[\![P]\!]\rho \wedge (s,\alpha_Q,\aleph_Q) \in \mathcal{E}_T[\![Q]\!]\rho\})\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T[\![P\;{}_X\|_Y\;Q]\!]\rho \;\widehat{=}\;& SUP\{(s, max\{\alpha_P, \alpha_Q\}, \aleph_P \cup \aleph_Q \cup \aleph_Z) \mid \\
& \exists (s_P, \alpha_P, \aleph_P) \in \mathcal{E}_T[\![P]\!]\rho, (s_Q, \alpha_Q, \aleph_Q) \in \mathcal{E}_T[\![Q]\!]\rho \bullet \\
& \sigma(\aleph_P) \subseteq X \wedge \sigma(\aleph_Q) \subseteq Y \wedge \\
& s \in (s_P\;{}_X\|_Y\;s_Q) \wedge \sigma(\aleph_Z) \subseteq (\Sigma - (X \cup Y))\} \\
& \text{where} \\
& v\;{}_X\|_Y\;w = \{s \in (T\Sigma)^*_\leqslant \mid s \in (X \cup Y)^* \wedge \\
& \qquad\qquad\qquad s \upharpoonright X = v \wedge s \upharpoonright Y = w\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T[\![P \;|||\; Q]\!]\rho \;\widehat{=}\;& SUP\;(\{(s, max\{\alpha_P, \alpha_Q\}, \aleph) \mid \exists (u, \alpha_P, \aleph) \in \mathcal{E}_T[\![P]\!]\rho \\
& \qquad \wedge (v, \alpha_Q, \aleph) \in \mathcal{E}_T[\![Q]\!]\rho \bullet s \in Tmerge(u, v)\})
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T[\![P\,;Q]\!]\rho \;\widehat{=}\;& CL_{\cong}(SUP(\{(s, \alpha, \aleph) \mid \checkmark \notin \sigma(s) \;\wedge\; \forall I \in TINT \bullet \\
& \qquad\qquad (s, \alpha, \aleph \cup (I \times \{\checkmark\})) \in \mathcal{E}_T[\![P]\!]\rho\} \\
& \cup \qquad\qquad \{(s^\frown(w+t), \alpha+t, \aleph_1 \cup (\aleph_2+t)) \mid \\
& \qquad \checkmark \notin \sigma(s) \;\wedge\; end(\aleph_1) \leqslant t \wedge (w, \alpha, \aleph_2) \in \mathcal{E}_T[\![Q]\!]\rho \\
& \qquad \wedge (s^\frown\langle(t, \checkmark)\rangle, \aleph_1 \cup ([0, t) \times \{\checkmark\})) \in fail(\mathcal{E}_T[\![P]\!]\rho)\}))
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}_T[\![P \setminus X]\!]\rho \;\widehat{=}\;& SUP(\{s \setminus X, \beta, \aleph) \mid \;\exists\, \alpha \geqslant \beta \geqslant end(s) \bullet \\
& \qquad (s, \alpha, \aleph \cup ([0, max\{\beta, end(\aleph)\}) \times X)) \in \mathcal{E}_T[\![P]\!]\rho\})
\end{aligned}$$

$$\mathcal{E}_T[\![f^{-1}(P)]\!]\rho \;\widehat{=}\; \{(s, \alpha, \aleph) \mid (f(s), \alpha, f(\aleph)) \in \mathcal{E}_T[\![P]\!]\rho\}$$

$$\mathcal{E}_T[\![f(P)]\!]\rho \;\widehat{=}\; SUP(\{(f(s), \alpha, \aleph) \mid (s, \alpha, f^{-1}(\aleph)) \in \mathcal{E}_T[\![P]\!]\rho\})$$

$$\mathcal{E}_T[\![X]\!]\rho \;\widehat{=}\; \rho(X)$$

$\mathcal{E}_T[\![\mu X \bullet P]\!]\rho \;\widehat{=}\;$ The unique fixed point of the contraction mapping $\lambda\, Y \bullet \mathcal{E}_T[\![P]\!](\rho[W_\delta(Y)/X])$, where W_δ is the mapping on TM_{FS} corresponding to $\lambda\, Y \bullet WAIT\;\delta\,;Y$.

Bibliography

[BB89] J.C.M. Baeten and J.A. Bergstra. *Real Time Process Algebra.* PRG, University of Amsterdam, 1989. (draft)

[BC87] A. Boucher and R. Gerth. *A Timed Model for Extended Communicating Processes.* Proceedings of ICALP '87, LNCS 267, pp 95–114, 1987.

[BK83] H. Barringer and R. Kuiper. *Towards the Hierarchical, Temporal Logic, Specification of Concurrent Systems.* LNCS 207 pp 157–183, 1983.

[BK84] H. Barringer and R. Kuiper. *Hierarchical Development of Concurrent Systems in a Temporal Logic Framework.* LNCS 197 pp 35–61, 1984.

[BKl84] J.A. Bergstra and J.W. Klop. *Process Algebra for Synchronous Communication.* Information and Control 60, pp 109–137, 1984.

[BKP84] H. Barringer, R. Kuiper and A. Pnueli. *Now You May Compose Temporal Logic Specifications.* Proceedings of the 16th ACM Symposium on the Theory of Computing, pp 51–63, 1984.

[BKP85] H. Barringer, R. Kuiper and A. Pnueli. *A Really Abstract Concurrent Model and its Temporal Logic.* Proceedings of the 13th ACM Symposium on the Principles of Programming Languages, pp 173–183, 1985.

[Bla89] S R. Blamey. *TCSP Processes as "Predicates".* Oxford University, 1989. (to appear)

[BR85] S.D. Brookes and A.W. Roscoe. *An Improved Failures Model for Communicating Proeesses.* Proceedings of the Pittsburgh Seminar on Concurrency, LNCS 197, pp 281–305, 1985.

[Bro83] S.D. Brookes. *A Model for Communicating Sequential Processes.* Oxford University D.Phil thesis, 1983.

[CR83] J.E. Coolahan aud N. Roussopoulos. *Timing Requirements for Time-Driven Systems Using Augmented Petri Nets.* IEEE Transactions on Software Engineering, SE-9, September 1983.

[CR85] J.E. Coolahan and N. Roussopoulos. *A Timed Petri Net Methodology for Specifying Real-Time System Timing Constraints.* Proceedings of the International Workshop on Timed Petri Nets, Torino, Italy, July 1985

[Dat85] N. Dathi. *The Pursuit of Deadlock Freedom.* Oxford University M.Sc. thesis 1985.

[DS89] J.W. Davies and S.A. Schneider. *Factorising Proofs in Timed CSP.* Proceedings of the Fifth Conference on the Mathematical Foundations of Programming Semantics, March 1989. (to appear)

[FG89] M.K. Frauklin and A. Gabrielian. *A Transformational Method for Verifying Safety Properties in Real-Time Systems.* Presented at the 10th Real-Time Systems Symposium, December 1989.

[GF89] A. Gabrielian and M.K. Franklin. *Multi-Level Specification and Verification of Real-Time Software.* Thomson-CSF, Inc. Technical Report 89-14, July 1989.

[GLZ88] R. Gerber, I. Lee and A. Zwarico. *A Complete Axiomatization of Real-time Processes.* University of Pennsylvania, November 1988.

[Hen88] M. Hennessy. *An Algebraic Theory of Processes.* MIT, 1988.

[HGR87] C. Huizing, R. Gerth, and W.P. de Roever. *Full Abstraction of a Real-Time Denotational Semantics for an* occam*-like Language.* Proceedings of the 14th ACM Symposium on Principles of Programming Languages, pp 223–237, 1987.

[HM85] M. Hennessy and R. Milner. *Algebraic Laws for Nondeterminism and Concurrency.* Journal of the ACM, 32, pp 137–161, January 1985.

[HO83] B.T. Hailpern and S.S. Owicki. *Modular Verification of Computer Communication Protocols.* IEEE Transactions on Communications, COM-31, pp 56–68, 1983.

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

[HR89] J.J.M. Hooman and W.P. de Roever. *Design and verification in real-time distributed computing: an introduction to compositional methods.* Proceedings of the Ninth International Symposinm on Protocol Specification, Testing and Verification, North Holland, 1989.

[HW89] J. Hooman and J. Widom. *A Temporal-Logic Based Compositional Proof System for Real-Time Message Passing.* Proceedings of PARLE '89 (2), pp 424–441, LNCS 366, 1989.

[Jac89] D.M. Jackson. *The Specification of Aircraft Engine Control Software Using Timed CSP.* Oxford University M.Sc. thesis, 1989.

[Jon82] G. Jones. *A Timed Model of Communicating Processes.* Oxford University D.Phil thesis, 1982.

[Koy89] R.L.C. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* PhD thesis, Eindhoven University of Technology, The Netherlands, 1989.

[KR83] R. Koymans and W.P. de Roever. *Examples of a real-time temporal logic specification.* LNCS 207, pp 231–252, 1983.

[KSRGA88] R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, and S. Arun-Kumar. *Compositional Semantics for Real-Time Distributed Computing.* Information and Computation, 79, pp 210–256, 1988.

[LM86] K.G. Larsen and R. Milner. *Verifying a Protocol Using Relativised Bisimulation.* LNCS 267, pp 126–135, 1986.

[LS87] N.G. Leveson and J.L. Stolzy. *Safety Analysis Using Petri Nets.* IEEE Transactions on Software Engineering, SE-13, March 1987.

[LZ87] I. Lee and A. Zwarico. *An Algebra of Communieating Time Dependent Processes.* University of Pennsylvania, report MS-CIS-87-110, December 1987.

[LZ88] I. Lee and A. Zwarico. *Timed Acceptances: A Model of Time Dependent Processes.* University of Pennsylvania, January 1988.

[MF76] P.M. Merlin and D.J. Farber. *Recoverability of communication protocols — Implications of a theoretical study.* IEEE Transactions on Communication, COM-24, pp 1036–1043, 1976.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MP82] Z. Manna and A. Pnueli. *Verification of Concurrent Programs: The Temporal Framework.* in R. Boyer and J. Moore (eds.) *The Correctness Problem in Computer Science.* International Lecture Series in Computer Science, Academic Press, London 1982.

[Mer87] A. Merceron. *Fair Processes.* in Advances in Petri Nets, LNCS 266, pp 181–195, 1987.

[NA88] K.T. Narayana and A.A. Aaby. *Specification of Real-Time Systems in Real-Time Temporal Interval Logic.* IEEE Real-Time Systems Symposium, Huntsville, Alabama, December 1988.

[NRSV89] X. Nicollin, J-L. Richier, J. Sifakis and J. Voiron. *ATP: an Algebra for Timed Processes.* Submitted to IFIP Working Conference on "Programming Concepts and Methods", April 1990.

[NS89] K.T. Narayana and E. Shade. *Language Concepts for Real-time Concurrency.* The Pennsylvania State University, 1989.

[OH83] E.R. Olderog and C.A.R. Hoare. *Specification-oriented Semantics for Communicating Processes.* LNCS 154, pp 561-572, 1983; Acta Informatica 23, pp 9–66, 1986.

[Pet77] J.L. Peterson. *Petri Nets.* ACM Computing Surveys, September 1977.

[Pnu77] A. Pnueli. *The Temporal Logic of Programs.* Proceedings of Foundations of Computer Science, pp 46–57, 1977.

[Pnu86] A. Pnueli. *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A survey of current trends.* LNCS 224, pp 510-584, 1986.

[PS88] K. Paliwoda and J.W. Sanders. *The Sliding-Window Protocol in CSP.* Oxford University Programming Research Group Technical Monograph PRG-66, 1988.

[QF87] J. Quemada and A. Fernandez. *Introduction of Quantitative Relative Time into LOTOS.* in Protocol Specification, Testing and Verification VII (H. Rudin, C.H. West eds), North Holland 1987.

[Ree88] G.M. Reed. *A Uniform Mathematical Theory for Real-Time Distributed Computing.* Oxford University D.Phil thesis, 1988.

[Reg89] T. Regan. *Personal Communication.* November 1989.

[Ros82] A.W. Roscoe. *A Mathematical Theory of Communicating Processes.* Oxford University D.Phil thesis, 1982.

[Ros88a] A.W. Roscoe. *An Alternative Order for the Failures Model.* Oxford University Computing Laboratory technical monograph PRG-67, 1988.

[Ros88b] A.W. Roscoe. *Unbounded Nondeterminism in CSP.* Oxford University Computing Laboratory technical monograph PRG-67, 1988.

[RR86] G.M. Reed and A. W. Roscoe. *A Timed Model for Communicating Sequential Processes.* Proceedings of ICALP'86, LNCS 226, pp 314-323, 1986; Theoretical Computer Science 58, pp 249-261, 1988.

[RR87] G.M. Reed and A. W. Roscoe. *Metric Spaces as Models for Real-time Concurrency.* Proceedings of the Third Workshop on the Mathematical Foundations of Programming Language Semantics, LNCS 298, pp 331–343, 1987.

[Sch88] S.A. Schneider. *Communication in Timed Distributed Computing.* Oxford University M.Sc. thesis 1988.

[Sif80] J. Sifakis. *Performance Evaluation of Systems Using Nets.* LNCS 84, pp 307–319, 1980.

[SL87] A.U. Shanker and S.S. Lam. *Time-dependent distributed systems: proving safety, liveness and real-time properties.* Distributed Computing 2, pp 61–79, 1987.

[SL89] A.U. Shanker and S.S. Lam. *A stepwise refinement heuristic for protocol construction.* University of Maryland Technical Report, 1989.

[SN89] E. Shade and K.T. Narayana. *Real-time Semantics for Shared-variable Concurrency.* The Pennsylvania State University, 1989.

[Sto77] J.E. Stoy. *Denotational Semantics.* MIT Press, 1977.

[Tan81] A.S. Tanenhaum. *Computer Networks.* Prentice-Hall, 1981.

[Zed89] H. Zedan. *Formal Modelling of Distributed Real-Time Systems.* Dept. of Computer Science, University of York, 1989.

Index of Notation

Notation	*Page*
$\mathcal{T}_T$	10
$\mathcal{S}_T$	10
$\mathcal{F}_T$	10
$\mathcal{E}_T^*$	10
$\mathcal{E}_T$	10
$\equiv$	10
$inits$	10

Other notation

Notation	*Page*
$\mathbf{i}$:	10
rem_i	11
$\Vert_I$	11
$\overset{\circ}{;}$	12
$\sqcap_I$	14
$a : A \rightarrow P_a$	16
$\overset{t}{\triangleright}$	19
$\overset{t}{\downarrow}$	21
$\underset{i}{\overset{t}{\nabla}}$	24
$\underset{i \in I}{\nabla}$	24
$\upharpoonleft$ (on untimed processes)	25
$\upharpoonleft$ (on timed processes)	25
$T\Sigma_{\leqslant}^{\omega}$	28
$T\Sigma_{\leqslant}^{i}$	28
$IRSET$	28
$I(P)$	28
$\mathcal{I}_T[\![P]\!]$	28
$SI(P)$	29
$\mathcal{SI}_T[\![P]\!]$	29
$\underline{\mathbf{sat}}$	31
$\overline{TF}$	42
$\mathcal{SS}$	43
$I(P_1, P_2)$	49
Θ	89
$\sqsubseteq_t$ (between processes)	90
$\sqsubseteq_t$ (between predicates)	95
sr (on M_T specifications)	96
we (to M_T specifications)	97
$\sqsubseteq_f$ (between processes)	99
$\sqsubseteq_{ts}$ (between processes)	99
$R(s, \aleph)$	103
$F(P_1, P_2)$	104
$\sqsubseteq_f$ (between predicates)	119
sr (on M_F specifications)	119
wc (to M_F specifications)	120
$e.v$	124
$c.V$	124
$c?x$	124
$c!x$	124
$\leqslant$	124
$\leqslant_n$	124
$\gg$	125
$S_{a,b}$	125
PAR	133
NET	133
$\trianglelefteq$	156