Correctness and Communication in Real-Time Systems

Steven A. Schneider

Balliol College
Oxford University Programming Research Group

Michaelmas Term 1989

Submitted for the degree of Doctor of Philosophy at the University of Oxford
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 3

2 Notation 5

3 Additions to Timed CSP 12
   3.1 Delayed Sequential Composition 12
   3.2 Indexed Choice 12
      3.2.1 Arbitrary non-deterministic choice 14
      3.2.2 Indexed prefix choice 16
   3.3 An alternative syntax for TCSP 17
   3.4 Derived Operators 18
      3.4.1 Timeout 19
      3.4.2 Tireout 20
      3.4.3 Interrupts 23
   3.5 The After Operator 25
   3.6 Infinite behaviours 27

4 Factorising Proofs 31
   4.1 Behavioural Specifications 31
   4.2 The proof system for $TMs$ 32
   4.3 Soundness 38
   4.4 Completeness 43
   4.5 A treatment of stability 48
   4.6 The proof system for $TMsI$ 52
   4.7 Soundness and Completeness 57

5 Aspects of Good Behaviour 58
   5.1 Non-retraction 59
   5.2 Responsiveness 68
   5.3 Promptness 72
   5.4 Impartiality 78
   5.5 Limited on $A$ 81
   5.6 Bounded Stability 84

6 Timewise Refinement 87
   6.1 The mapping $\Theta$ 89
   6.2 Weak Timewise Refinement 90
   6.3 Strong Timewise Refinement 98
   6.4 Other Timewise Refinements 122
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, such 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 the semantic
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 $\delta$, 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 $\delta$ 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 $\Sigma$. The set of all possible timed traces is denoted $T\Sigma^*$. We write $s_1 \overset{*}{\circ} s_2$ 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:

$$s_1 \text{ in } s_2 \equiv \exists u, v \cdot u \overset{*}{\circ} s_1 \overset{*}{\circ} v = s_2$$

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

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

$$\text{first}(((t, a)) \overset{*}{\circ} s) \equiv a$$

$$\text{last}(s \overset{*}{\circ} ((t, a))) \equiv a$$

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

$$\text{begin}(\langle \rangle) \equiv \infty$$

$$\text{begin}(((t, a)) \overset{*}{\circ} s) \equiv t$$

$$\text{end}(\langle \rangle) \equiv 0$$

$$\text{end}(s \overset{*}{\circ} ((t, a))) \equiv t$$

Again, the values chosen for the empty trace are the most convenient for the subsequent mathematics.
The \( \oplus n \) operator returns the \( n \)th timed event of a trace. It is defined as follows:

\[
\begin{align*}
\langle \rangle \oplus n & \equiv (\infty, e) \\
\langle (t, a) \rangle \oplus s \oplus l & \equiv (t, a) \\
\langle (t, a) \rangle \oplus s \oplus (n + 1) & \equiv s \oplus n
\end{align*}
\]

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.

\[
\begin{align*}
\langle \rangle \uparrow I & \equiv \langle \rangle \\
\langle (t, a) \rangle \uparrow I & \equiv (t, a) \uparrow (s \uparrow I) & \text{if } t \in I \\
\langle \rangle \uparrow t & \equiv \langle \rangle \\
\langle (t, a) \rangle \uparrow t & \equiv (t, a) \uparrow (s \uparrow t) & \text{if } t' \leq t \\
\langle \rangle \uparrow & \equiv \langle \rangle \\
\langle (t, a) \rangle \uparrow & \equiv (t, a) \uparrow (s \uparrow A) \\
\langle (t, a) \rangle \uparrow A & \equiv s \uparrow A \\
\langle (t, a) \rangle \uparrow (s \uparrow A) & \equiv (t, a) \uparrow (s \uparrow A) \\
\langle (t, a) \rangle \uparrow A & \equiv s \uparrow A \\
\langle (t, a) \rangle \uparrow (s \uparrow A) & \equiv (t, a) \uparrow (s \uparrow A)
\end{align*}
\]

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, \( \uparrow \), 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:

\[
\begin{align*}
\langle \rangle \uparrow A & \equiv \langle \rangle \\
\langle (t, a) \rangle \uparrow A & \equiv (t, a) \uparrow (s \uparrow A) & \text{if } a \in A \\
\langle (t, a) \rangle \uparrow A & \equiv s \uparrow A & \text{otherwise} \\
\langle (t, a) \rangle \uparrow (s \uparrow A) & \equiv (t, a) \uparrow (s \uparrow A) & \text{if } a \in A \\
\langle (t, a) \rangle \uparrow A & \equiv s \uparrow A & \text{otherwise}
\end{align*}
\]

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:

\[
\begin{align*}
\text{hstrip}(\langle \rangle) & \equiv \langle \rangle \\
\text{hstrip}(\langle (t, a) \rangle \uparrow s) & \equiv ((t, a)) \uparrow \text{hstrip}(s) \\
\text{hstrip}(\langle (t, a) \rangle \uparrow s) & \equiv ((t, a)) \uparrow \text{hstrip}(s)
\end{align*}
\]

Following Reed, we write \( \tilde{s} \) for \( hstrip(s) \).

The operator \( tstrip \) strips the timing information from a trace:

\[
\begin{align*}
\text{tstrip}(\langle \rangle) & \equiv \langle \rangle \\
\text{tstrip}(\langle (t, a) \rangle \uparrow s) & \equiv (a) \uparrow \text{tstrip}(s)
\end{align*}
\]
We use \textit{thstrip} to denote the composition of these two functions.

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

$$\sigma(s) \equiv \{ a \in \Sigma \mid \exists t \cdot ((t, a)) \text{ in } \text{hstrip}(s) \}$$

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

We define a temporal shift operator:

$$\langle s \rangle \triangleleft t \equiv \langle s \rangle$$

$$\langle (t_1, a) \rangle \triangleleft s \triangleleft t \equiv \langle (t_1 - t, a) \rangle \triangleleft \langle (s \triangleleft t) \rangle \quad \text{if } t_1 \geq t$$

$$\langle (t_1, a) \rangle \triangleleft s \triangleleft t \equiv s \triangleleft t \quad \text{otherwise}$$

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

$$s \downarrow A \equiv \#(s \uparrow A)$$

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

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

$$s \setminus A \equiv s \uparrow (\Sigma - A)$$

and an equivalence relation $\equiv$ on timed traces as follows: $u \equiv v$ 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, $u \equiv v$ precisely when there is a bijective function $f : \{i \in \mathbb{N} \mid 1 \leq i \leq \#u\} \rightarrow \{i \in \mathbb{N} \mid 1 \leq i \leq \#u\}$ such that $\forall i \in \{i \in \mathbb{N} \mid 1 \leq i \leq \#u\} \cdot u@i = v@f(i)$.

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

$$u \parallel_X Y \equiv \{ s \mid s \uparrow X = u \land s \uparrow Y = v \land (X \cup Y) = s \}$$

$$u \parallel v \equiv \{ s \mid \forall t \cdot s \uparrow t \cong (u \uparrow t) \cap (v \uparrow t) \}$$

We write $T\text{merge}(u, v)$ to denote the set of all traces in $T\Sigma^*_\mathbb{N}$ obtained by interleaving $u$ and $v$. When $\tilde{u} = \tilde{v}$ we define the trace $s = u \lor v$ to be such that $\tilde{s} = \tilde{v}$, and the $n$th element of $s$ is hatted precisely when the $n$th element of $u$ or $v$ is.
Timed Refusals

Recall the definitions from [Ree88]:

\[
TINT \equiv \{ (l, r) \mid l, r \in \mathbb{R} \land 0 \leq l < r < \infty \}
\]

\[
RTOK \equiv \{ I \times X \mid I \in TINT \land X \in \mathcal{P}(\Sigma) \}
\]

\[
RSET \equiv \{ \bigcup Z \mid Z \in p(RTOK) \}
\]

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 \( \mathcal{N} \) 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

\[
I(\mathcal{N}) \equiv \{ t \mid \exists a \in \Sigma \quad (t, a) \in \mathcal{N} \}
\]

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

\[
\text{begin}(\mathcal{N}) \equiv \infty \quad \text{if } \mathcal{N} = \emptyset
\]

\[
\text{begin}(\mathcal{N}) \equiv \inf(I(\mathcal{N})) \quad \text{otherwise}
\]

\[
\text{end}(\mathcal{N}) \equiv 0 \quad \text{if } \mathcal{N} = \emptyset
\]

\[
\text{end}(\mathcal{N}) \equiv \sup(I(\mathcal{N})) \quad \text{otherwise}
\]

The \textit{before}, \textit{after}, and \textit{during} operators can be defined on refusals:

\[
\mathcal{N} \uparrow t \equiv \mathcal{N} \cap (0, t) \times \Sigma
\]

\[
\mathcal{N} \downarrow t \equiv \mathcal{N} \cap (t, \infty) \times \Sigma
\]

\[
\mathcal{N} \uparrow [t_1, t_2) \equiv \mathcal{N} \cap ([t_1, t_2) \times \Sigma)
\]

Recalling that \( \Sigma \) 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 \( \uparrow \) symbol to denote set restriction:

\[
\mathcal{N} \uparrow A \equiv \mathcal{N} \cap ([0, \infty) \times A)
\]

with hiding defined in terms of restriction:

\[
\mathcal{N} \setminus A \equiv \mathcal{N} \uparrow (\Sigma - A)
\]
We define a temporal shift operator on refusals:
\[ R \downarrow t \triangleq \{ (t_i - t, a) \mid (t_i, a) \in R \land t_i \geq t \} \]

We define an alphabet operator \( \sigma \):
\[ \sigma(R) \triangleq \{ a \in \Sigma \mid \exists t \cdot (t, a) \in R \} \]

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.

\[
\begin{align*}
\text{begin}(s, R) & \triangleq \min(\{ \text{begin}(s), \text{begin}(R) \}) \\
\text{end}(s, R) & \triangleq \max(\{ \text{end}(s), \text{end}(R) \}) \\
(s, R) \uparrow I & \triangleq (s \uparrow I, R \uparrow I) \\
(s, R) \downarrow t & \triangleq (s \downarrow t, R \downarrow t) \\
(s, R) \downarrow t & \triangleq (s \downarrow t, R \downarrow t) \\
\sigma(s, R) & \triangleq \sigma(s) \cup \sigma(R) \\
(s, R) \downarrow t & \triangleq (s \downarrow t, R \downarrow t) \\
(s, R) \uparrow A & \triangleq (s \uparrow A, R \uparrow A) \\
(s, R) \setminus A & \triangleq (s \setminus A, R \setminus A)
\end{align*}
\]

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.

\[ \widehat{A}(s, R) \triangleq [0, \text{end}(s, R)) \times A \subseteq R \]

This allows an alternative formulation of the semantic equation for hiding in \( TM_F \):
\[ \mathcal{F}_T[P \setminus A] = \{(s \setminus A, R) \mid \widehat{A}(s, R) \land (s, R) \in \mathcal{F}_T[P] \land \sigma(R) \subseteq A \} \]
Processes

The functions \textit{traces}, \textit{fail} and \textit{stab} are defined on sets of \{\textit{trace}, \textit{stability}, \textit{refusal}\} triples as in [Ree88]:

\[
\begin{align*}
\text{traces}(P) & \equiv \{ s | (s, \alpha, \emptyset) \in P \} \\
\text{fail}(P) & \equiv \{ (s, \emptyset) | (s, \alpha, \emptyset) \in P \} \\
\text{stab}(P) & \equiv \{ (s, \alpha) | (s, \alpha, \emptyset) \in P \} \\
\text{SUP}(P) & \equiv \{ (s, \alpha, \emptyset) | (s, \emptyset) \in \text{fail}(P) \} \\
\text{CL}_w(P) & \equiv \{ (s, \alpha, \emptyset) | \exists (w, \alpha, \emptyset) \in P \cdot w \not\equiv s \}
\end{align*}
\]

These functions are extended to apply to TCSP processes, by applying them to the semantics (in \textit{TMps}) of their arguments.

We extend the alphabet operator to \textit{TCSP} processes:

\[
\sigma(P) \equiv \bigcup \{ \sigma(s) | s \in \text{traces}(P) \}
\]

and observe that this differs from the \textit{alphabet} concept used in other versions of \textit{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 \( T, S, F, \) and \( E \) to denote the semantic mappings from \textit{CSP} to \( M_T, M_S, M_F, \) and \( M_{FS} \) respectively. We use \( \mathcal{T}_T, \mathcal{S}_T, \mathcal{F}_T, E_T^T, \) and \( E_T \) to denote the semantic mappings from \textit{TCSP} to \( T\mathcal{M}_T, T\mathcal{M}_S, T\mathcal{M}_F, T\mathcal{M}_{FS}, \) and \( T\mathcal{M}_{FS} \) respectively (see appendix B).

We use the \( \equiv \) relation between \textit{TCSP} processes when they have the same semantics in the model \( T\mathcal{M}_{FS} \) (given in [Ree88] and reproduced in appendix B.2).

\[
P \equiv Q \equiv E_T [P] = E_T [Q]
\]

If two processes are equivalent in this sense, then they will also have the same semantics in \( T\mathcal{M}_F \). Hence the laws given later for \( T\mathcal{M}_{FS} \) also hold for \( T\mathcal{M}_F \).

The set of the possible initial events of a process \( P \) is denoted \textit{inits}(\( P \)). It is defined by

\[
inits(P) \equiv \{ a | \exists t \cdot ((t, a)) \in \text{traces}(P) \}
\]

As in untimed \textit{CSP}, we may label processes by applying an alphabet transformation to them.

\[
i : P \equiv f_i(P)
\]
where $f_i(a) = i \cdot a$ for all events $a \in \Sigma \setminus \{\top\}$; but termination is still signalled by $\top$ so we have $f_i(\top) = \top$. The labels may be removed by applying $f_i^{-1}$ to the process. We define $\text{rem}_i(P) = f_i^{-1}(P)$.

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:

$$
\begin{align*}
\left\|\left\| P_i \right\|_{X_1} \right\|_{X_2} P_2 \\
\left\|\left\| P_i \right\|_{\{X_i \mid i \in \{n+1\}\}} \right\|_{\{X_i \mid i \in \{n\}\}} P_{n+1}
\end{align*}
$$

when $n \geq 2$. 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 behaviour 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 shall 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:

\[
P ; Q \triangleq P ; \text{WAIT} \delta ; Q
\]

It follows that \( ; \) is well defined and continuous, since it is defined in terms of the basic TCSP operators. We can derive a semantic equation for \( P ; Q \) in the evaluation domain \( TMFS \) from the equations for the sequential operator \( ; \) and \( \text{WAIT} \delta \). This may be simplified to eliminate the closure operator \( CL_\infty \), 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 semantic equations for $P; Q$ may be written as follows:

$$\mathcal{E}_T[P; Q] = \text{SUP}(\{(s, \alpha, \mathcal{R}) | \exists \sigma(s) \land \forall I \in \text{TINT} \cdot$$

$$\left(\begin{array}{l}
(s, \alpha, \mathcal{R} \cup (I \times \{\top\})) \in \mathcal{E}_T[P] \\
\cup \\
(s, \alpha, \mathcal{R} | \exists t \cdot \top \notin \sigma(s \uparrow t) \\
\land ((s \uparrow t)^{\top}\{(t, \top)\}, \mathcal{R} \uparrow t \cup ([0, t) \times \{\top\})) \in \text{fail}(\mathcal{E}_T[P]) \\
\land (s \uparrow (t + \delta)), \alpha \uparrow (t + \delta), \mathcal{R} \uparrow (t + \delta) \in \mathcal{E}_T[Q]\end{array}\right)\}$$

$$\mathcal{E}_T^*[P; Q] = \text{SUP}(\{(s, \alpha, X) | \top \notin \sigma(s) \land (s, \alpha, X \cup \{\top\}) \in \mathcal{E}_T^*[P] \}

$$

$$\mathcal{E}_T^*[P; Q] = \text{SUP}(\{(s, \alpha, X) | \top \notin \sigma(s) \land (s, \alpha, X \cup \{\top\}) \in \mathcal{E}_T^*[P] \}

$$

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

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

$$\left(\begin{array}{l}
(a \rightarrow P) ; Q \equiv a \rightarrow (P ; Q) \\
(P ; Q) ; R \equiv P ; (Q ; R) \\
(\text{WAIT} t_1 ||| \text{WAIT} t_2) ; P \equiv \text{WAIT} \min\{t_1, t_2\} ; P
\end{array}\right)$$

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

$$\text{WAIT} t_1 ; \text{WAIT} t_2 \not\equiv \text{WAIT} (t_1 + t_2)$$

$$\text{SKIP} ; P \not\equiv P$$

In place of the first of these we have

$$\text{WAIT} t_1 ; \text{WAIT} t_2 \equiv \text{WAIT} (t_1 + t_2 + \delta)$$
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:

\[ \mathcal{E}_T[\bigcap_{i \in I} P_i] \triangleq \text{SUP}(\bigcup_{i \in I} \mathcal{E}_T[P_i]) \]

\[ \mathcal{E}^*_T[\bigcap_{i \in I} P_i] \triangleq \text{SUP}(\bigcup_{i \in I} \mathcal{E}^*_T[P_i]) \]

\[ \mathcal{F}_T[\bigcap_{i \in I} P_i] \triangleq \bigcup_{i \in I} \mathcal{F}_T[P_i] \]

\[ \mathcal{S}_T[\bigcap_{i \in I} P_i] \triangleq \text{SUP}(\bigcup_{i \in I} \mathcal{S}_T[P_i]) \]

\[ \mathcal{T}_T[\bigcap_{i \in I} P_i] \triangleq \bigcup_{i \in I} \mathcal{T}_T[P_i] \]

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

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

for any element \( S \) of TMFS. Now consider

\[ P_0 \triangleq \text{STOP} \]

\[ P_{n+1} \triangleq P_n \parallel a \rightarrow \text{STOP} \]

\[ P \triangleq \bigcap_{n \in \mathbb{N}} P_n \]
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

$$\exists n : [0, \infty) \rightarrow N, \forall i \in I, t \in [0, \infty) \bullet s \in \text{traces}(E_T[P_i]) \land \text{end}(s) \leq t \Rightarrow \#s \leq n(t)$$

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 $\bigcup_{i \in I} E_T[P_i]$ satisfies the bounded speed axiom. Observe that any finite set of processes is uniformly bounded in each timed model.

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

**Lemma 3.2.3** $E_T[\bigsqcap_{i \in I} P_i]$ is well-defined if and only if the set $\{P_i \mid i \in I\}$ is uniformly bounded 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 which is non-deterministic over some interval. For all $t$, the process $\text{WAIT } t$ 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 $\{\text{WAIT } t \mid t \in T\}$ is uniformly bounded, and so $\bigsqcap_{t \in T} \text{WAIT } t$ will have a well defined semantics; we will abbreviate it $\text{WAIT } T$.

**Corollary 3.2.4** The semantics of the process $\text{WAIT } T$ is always well defined, where

$$\text{WAIT } T \equiv \bigsqcap_{t \in T} \text{WAIT } t$$
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:

\[ \mathcal{E}_T[a : A \rightarrow P_a] \equiv \{ (\langle \cdot, 0, r \rangle \mid A \cap \sigma(r) = \emptyset \} \]
\[ \bigcup \{ \langle r \rangle \} \text{ where } a \in A \wedge t \geq 0 \wedge A \cap \sigma(r \uparrow t) = \emptyset \]
\[ \wedge (s, \alpha, \mathcal{R} \uparrow (t + \delta)) \in \mathcal{E}_T[P(a)] \}

\[ \mathcal{E}_T^*[a : A \rightarrow P(a)] \equiv \{ (\langle \cdot, 0, X \rangle \mid A \cap X = \emptyset \} \]
\[ \bigcup \{ ((0, a)) \uparrow (s + \delta), \alpha + \delta, X \} \mid a \in A \wedge (s, \alpha, X) \in \mathcal{E}_T^*[P(a)] \}
\[ \bigcup \{ ((t, a)) \uparrow (s + t + \delta), \alpha + t + \delta, X \} \mid a \in A \wedge t \geq 0 \wedge (s, \alpha, X) \in \mathcal{E}_T^*[P(a)] \}

\[ \mathcal{F}_T[a : A \rightarrow P_\delta] \equiv \{ (\langle \cdot, \delta \rangle \mid A \cap \sigma(\delta) = \emptyset \} \]
\[ \bigcup \{ ((0, \delta)) \uparrow (s + \delta), \alpha + \delta \} \mid a \in A \wedge \alpha \in \mathcal{S}_T[P(a)] \}
\[ \bigcup \{ ((t, a)) \uparrow (s + t + \delta), \alpha + t + \delta \} \mid a \in A \wedge t \geq 0 \wedge (s, \alpha) \in \mathcal{S}_T[P(a)] \}

\[ \mathcal{S}_T[a : A \rightarrow P(a)] \equiv \{ (\langle \cdot, 0 \rangle \} \}
\[ \bigcup \{ ((0, a)) \uparrow (s + \delta), \alpha + \delta \} \mid a \in A \wedge (s, \alpha) \in \mathcal{S}_T[P(a)] \}
\[ \bigcup \{ ((t, a)) \uparrow (s + t + \delta), \alpha + t + \delta \} \mid a \in A \wedge t \geq 0 \wedge (s, \alpha) \in \mathcal{S}_T[P(a)] \}

\[ \mathcal{T}_T[a : A \rightarrow P(a)] \equiv \{ (\cdot, 0) \}
\[ \bigcup \{ ((0, a)) \uparrow (s + \delta) \mid a \in A \wedge s \in \mathcal{T}_T[P(a)] \}
\[ \bigcup \{ ((t, a)) \uparrow (s + t + \delta) \mid a \in A \wedge t \geq 0 \wedge s \in \mathcal{T}_T[P(a)] \}

16
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 $\{P(a) | a \in A\}$ 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 $\text{WAIT } t ; P$.

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 $\text{WAIT } t$ command. With the exception of these alterations, the syntax is identical to that defined in [Ree88]. Our syntax for $TCSP$ is given as follows:

$$P ::= \text{STOP} \mid \bot \mid \text{SKIP} \mid \text{WAIT } t \mid \text{a} \rightarrow P \mid a : A \rightarrow P(a) \mid P \sqcap P \mid \sqcap_{i} P, \mid P \sqcup P \mid P \sqcup_{A} P \mid P \sqcup_{B} P \mid P || | P \mid \text{WAIT } t ; P \mid P ; P \mid P \setminus A \mid f^{-1}(P) \mid f(P) \mid \mu X \cdot F(X)$$
If we wish to give a more explicit semantic treatment of recursion, then we use a syntax with process variables, as follows:

\[
P ::= \text{STOP} \mid \bot \mid \text{SKIP} \mid \text{WAIT } t \mid \\
a \rightarrow P \mid a : A \rightarrow P(a) \mid P \cap P \mid \bigcap_i P_i \mid P \square P \\
P || P \mid P_A \parallel B P \mid P ||| P \\
\text{WAIT } t ; P \mid P \downarrow P \mid P \setminus A \\
f^{-1}(P) \mid f(P) \mid X \mid \mu X \cdot P
\]

where \( X \) is of type \text{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 syntax above gives rise to TCSP terms. The processes \text{STOP}, \text{SKIP}, \bot \text{ and } \text{WAIT } t \text{ have no free process variables; } X \text{ has as its set of free process variables the set } \{X\}, \text{ the set of free process variables of } \mu X \cdot P \text{ is that of } P \text{ with } X \text{ 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 syntax with variables in chapter 6, where an explicit treatment of recursion is required; with the exception 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 \( \sigma(P) \cap X = \emptyset \Rightarrow P \equiv P \setminus X \)

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 $\triangleright$ is defined as follows:

$$P \triangleright Q \equiv rem_i((i : P \triangleleft \text{WAIT} t ; \text{trig} \rightarrow i : Q) \setminus \{\text{trig}\})$$

If $i : P$ performs no visible actions by time $t$, then the event $\text{trig}$ is made available by $\text{WAIT} t ; \text{trig} \rightarrow i : Q$. 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 $(i : Q) \setminus \text{trig}$, which is equivalent to $i : Q$, since $\text{trig} \notin \sigma(i : Q)$. If $i : P$ does perform a visible action by time $t$, then the choice is resolved in favour of $(i : P) \setminus \text{trig}$, 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 $\text{trig}$.

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

$$P \triangleright Q \equiv (P \triangleleft \text{WAIT} t ; \text{trig} \rightarrow Q) \setminus \{\text{trig}\}$$

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

$$P \triangleright Q \equiv (P \triangleleft \text{WAIT} t) ; Q$$

Example

The process $(a \rightarrow c \rightarrow \text{STOP}) \triangleright \delta b \rightarrow \text{STOP}$ is initially prepared to engage in event $a$. If $a$ is performed within 5 time units, then (after a delay of $\delta$) the process behaves like $c \rightarrow \text{STOP}$. If the $a$ does not occur within 5 time units, then the offer is withdrawn and the process behaves like $b \rightarrow \text{STOP}$.
Theorem 3.4.3 The semantics of \( P \triangleright Q \) in TM\( _T \) simplifies to the following equation:

\[
\mathcal{F}_T[P \triangleright Q] = \{(s, \bar{t}) \mid \text{begin}(s) \leq t \land (s, \bar{t}) \in \mathcal{F}_T[P] \} \\
\lor \\
\{((), \bar{t} \uparrow t) \in \mathcal{F}_T[P] \land \text{begin}(s) \geq t + \delta \land \\
(s \sim (t + \delta), \bar{t} \sim (t + \delta)) \in \mathcal{F}_T[Q] \}
\]

The timeout operator obeys the following laws:

\[
P \triangleright (Q \triangleright R) \equiv (P \triangleright Q) \triangleright (Q \triangleright R)
\]

\[
(P \sqcap Q) \triangleright R \equiv (P \triangleright R) \sqcap (Q \triangleright R)
\]

\[
P \triangleright (Q \sqcap R) \equiv (P \triangleright Q) \sqcap (P \triangleright R)
\]

\[
(P \sqcup Q) \triangleright R \equiv (P \triangleright Q) \sqcup (Q \triangleright R)
\]

\[
P \triangleright (Q \sqcup R) \equiv (P \triangleright Q) \sqcup (P \triangleright R)
\]

\[
(P \parallel Q) \triangleright (R \parallel S) \equiv (P \triangleright R) \parallel (Q \triangleright S)
\]

\[
(P \triangleright Q) \parallel (\text{WAIT } t ; R) \equiv \text{WAIT } t ; (Q \parallel R)
\]

\[
(\text{WAIT } t_1 ; P) \triangleright (t_2 + \delta) Q \equiv \text{WAIT } t_1 ; (P \triangleright (t_2 + \delta) Q) \quad \text{if } t_2 \geq t_1
\]

\[
\text{WAIT } (t_2 + \delta) ; Q \quad \text{if } t_2 < t_1
\]

\[
(\text{WAIT } t_1 ; P) \triangleright Q \equiv \text{WAIT } t_1 ; (P \triangleright (t_2 - t_1) Q) \quad \text{if } t_2 \geq t_1
\]

\[
\text{WAIT } (t_2 - t_1) ; Q \quad \text{if } t_2 < t_1
\]

\[
((a \rightarrow P) \triangleright Q) \setminus A \equiv \text{WAIT } \delta ; (P \setminus A) \quad \text{if } a \in A, t > 0
\]

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 \( \text{HALT}_t \), 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 following fashion: given an argument \( P \), axiom 5 for \( TM_T \) yields that

\[
\exists n \cdot \forall s \in \text{traces}(P) \cdot \text{end}(s) \leq t \Rightarrow \#s \leq n
\]
Then

$$\text{HALT}_t(P) \equiv P \upharpoonright_{T'} \left( \left( \bigoplus_{i=1}^{n} (a : \Sigma' \rightarrow \text{STOP}) \right) \upharpoonright \text{WAIT } t \right) ; \text{STOP}$$

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

$$\mathcal{F}_T[\text{HALT}_t(P)] = \{(s, q) \mid \text{end}(s \setminus \checkmark) \leq t \land (s \uparrow t, q \uparrow t) \in \mathcal{F}_T[P] \land (s, (q \uparrow t) \cup (q \cap [t, \infty) \times \{\checkmark\})) \in \mathcal{F}_T[P]\}$$

Definition 3.4.4 The timeout operator, $\tau_t$, is defined as follows:

$$P \tau_t Q \equiv \text{rem}_t(\Sigma \upharpoonright_{i, \checkmark, \checkmark} \left( \left( \begin{array}{c} (i : \text{HALT}_t(P)) \\ \text{WAIT } t ; a \rightarrow \text{SKIP} \end{array} \right) ; \left( \begin{array}{c} \text{SKIP} \\ b \rightarrow i : Q \end{array} \right) \right)) \setminus \{a, b\}$$

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

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

$$\left( \left( i : \text{HALT}_t(P) \right) \upharpoonright \left( \text{WAIT } t ; a \rightarrow \text{SKIP} \right) \right)$$

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

Example

The process

$$(\mu X \cdot a \rightarrow X) \tau_{10} \mu X \cdot b \rightarrow X$$

will recursively engage in event $a$ for 10 time units, after which it will behave like $\mu X \cdot b \rightarrow X$, regardless of the number of as performed.
Theorem 3.4.5 The semantic equation for the tireout operator reduces to the following:

\[ \mathcal{F}_T [P \downarrow_t Q] = \{ (s, \mathcal{R}) \mid (s \uparrow t, \mathcal{R} \uparrow t \cup [0, t) \times \{ \emptyset \}) \in \mathcal{F}_T [P] \land \emptyset \notin \sigma(s \uparrow t) \land \\
\quad s \uparrow (t, t + 2\delta) = () \land (s \equiv (t + 2\delta), \mathcal{R} \equiv (t + 2\delta)) \in \mathcal{F}_T [Q] \} \]

\[ \cup \{ (s, \mathcal{R}) \mid (s \equiv ((t', \emptyset)), \mathcal{R} \equiv (t' \cup [0, t') \times \{ \emptyset \}) \in \mathcal{F}_T [P] \land t' \leq t \land \\
\quad \emptyset \notin \sigma(s) \land \emptyset \notin \sigma(\mathcal{R} \uparrow t') \} \]

\[ \cup \{ (s \equiv ((t'', \emptyset)), \mathcal{R}) \mid (s \equiv ((t', \emptyset)), \mathcal{R} \equiv (t' \cup [0, t') \times \{ \emptyset \}) \in \mathcal{F}_T [P] \land \\
\quad t' \leq t \land \emptyset \notin \sigma(s) \land t'' \geq t' \land \emptyset \notin \sigma(\mathcal{R} \uparrow [t', t''])) \} \]

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

\[ P \downarrow_t Q \equiv (P \triangledown \text{WAIT } t) ; \text{SKIP} ; Q \]

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 \( P \downarrow_t Q \) 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

\[ \mathcal{F}_T [P \downarrow_t Q] = \{ (s, \mathcal{R}) \mid (s \uparrow t, \mathcal{R} \uparrow t) \in \mathcal{F}_T [P] \land s \uparrow (t, t + 2\delta) = () \land \\
\quad (s \equiv (t + 2\delta), \mathcal{R} \equiv (t + 2\delta)) \in \mathcal{F}_T [Q] \} \]

The tireout operator obeys the following laws:

\[ P \downarrow_t (Q \downarrow_t R) \equiv (P \downarrow_t Q) \downarrow_t (Q \uparrow_{t+2\delta} R) \]

\[ (P \cap Q) \downarrow_t R \equiv (P \downarrow_t R) \cap (Q \downarrow_t R) \]

\[ P \downarrow_t (Q \cap R) \equiv (P \downarrow_t Q) \cap (P \downarrow_t R) \]

\[ (P \sqcap Q) \downarrow_t R \equiv (P \downarrow_t R) \sqcap (Q \downarrow_t R) \]

\[ P \downarrow_t (Q \sqcap R) \equiv (P \downarrow_t Q) \sqcap (P \downarrow_t R) \]

22
\[(P \parallel Q) \overset{t}{\rightarrow} (R \parallel S) \equiv (P \overset{t}{\rightarrow} R) \parallel (Q \overset{t}{\rightarrow} S)\]

\[(P \overset{t}{\rightarrow} Q) \parallel (\text{WAIT}(t + \delta) ; R) \equiv \text{WAIT} t + \delta ; (Q \parallel R)\]

\(\begin{align*}
(WAIT t_{i} ; P) \overset{t_{g}+\delta}{\rightarrow} Q & \equiv WAIT t_{i} ; (P \overset{t_{g}-t_{i}}{\rightarrow} Q) & \text{if } t_{g} \geq t_{i} \\
& \quad \text{WAIT}(t_{g} + 2\delta) ; Q & \text{if } t_{g} < t_{i}
\end{align*}\)

\(\begin{align*}
(WAIT t_{i} ; P) & \overset{t_{g}}{\rightarrow} Q \equiv WAIT t_{i} ; (P \overset{t_{g}-t_{i}}{\rightarrow} Q) \quad \text{if } t_{g} \geq t_{i} \\
& \quad \text{WAIT}(t_{g} + 2\delta) ; Q \quad \text{if } t_{g} < t_{i}
\end{align*}\)

\[\begin{align*}
((a \rightarrow P) \overset{t_{g}}{\rightarrow} Q) \setminus X & \equiv \text{WAIT} \delta ; (P \overset{t_{g}}{\rightarrow} Q) \setminus X \quad \text{if } a \in X
\end{align*}\]

\[\begin{align*}
(P \overset{t_{1}}{\rightarrow} Q \overset{t_{2}+\delta}{\rightarrow} R) & \equiv (P \overset{t_{1}}{\rightarrow} R) \overset{t_{2}+\delta}{\rightarrow} (Q \overset{t_{2}}{\rightarrow} R)
\end{align*}\]

\[\begin{align*}
(P \overset{t_{1}}{\rightarrow} Q \overset{t_{2}+\delta}{\rightarrow} R) & \equiv P \overset{t_{1}}{\rightarrow} (Q \overset{t_{2}}{\rightarrow} R)
\end{align*}\]

### 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 \(\triangledown\) 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 interrupt 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 \(P \triangledown Q\).
Definition 3.4.6 We define the interrupt operator as follows:

$$\begin{align*}
P \uparrow_i Q & \equiv \text{rem}_i(\Sigma_{j.i \in \mathcal{J}} (\begin{array}{c}
  j : P \\
  ||
  j.i \rightarrow SKIP
\end{array}) ; (\begin{array}{c}
  \Box \\
  b \rightarrow j : Q
\end{array})) \\
& \quad \setminus \{b\}
\end{align*}$$

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:

$$P \uparrow_i Q \equiv (P || i \rightarrow SKIP) ; Q$$

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 $P \uparrow_{i \in I} Q(i)$ 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:

$$\begin{align*}
P \uparrow_{i \in I} Q & \equiv \text{rem}_{i \in I}(\Sigma_{i \in I} \cup \mathcal{J} \cup \{\Box\}) (\begin{array}{c}
  j : P \\
  ||
  j.i : j_l \rightarrow SKIP
\end{array}) ; (\begin{array}{c}
  \Box \\
  b_i : b_l \rightarrow j : Q(i)
\end{array}) \\
& \quad \setminus \{b_l\}
\end{align*}$$

where $b_l = \{b_i \mid i \in I\}$, and $j_l = \{j.i \mid i \in I\}$.
As before, we require that $\sigma(P) \cap I = \emptyset$ in order to ensure that the operator yields the desired behaviour.

The construct allows interrupts to be prioritised. In the process

$((P \triangledown_{i \in I} Q(i)) \triangledown_{j \in J} R(j))$

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 $s \in \text{traces}(P)$, then

$$\mathcal{F}[P/s] \triangleq \{(w, X) | (s \triangledown w, X) \in \mathcal{F}[P]\}$$

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 $(s, \mathcal{E}) \in \text{fail}(\mathcal{E}[P])$ and $\text{end}(s, \mathcal{E}) \leq t$ then

$$\mathcal{E}[P/((s, \mathcal{E}), t)] \triangleq \{(u, \alpha \triangledown t, \mathcal{E}') | (s \triangledown (u + t), \alpha, \mathcal{E} \cup (\mathcal{E}' + t)) \in \mathcal{E}[P]\}$$

Otherwise $P/((s, \mathcal{E}), t)$ 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 events 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

$$P \triangleq a \rightarrow ((b \rightarrow \text{STOP}) \triangledown b \rightarrow c \rightarrow \text{STOP})$$

$$\square a \rightarrow \text{STOP} \triangledown c \rightarrow \text{STOP}$$
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 \((s, N)\) is a time \(t\) which corresponds to the time up to which the process has been under observation when \((s, N)\) has been observed. It provides information about what has not been observed: no other events other than those in \(s\) have been performed, and only those events in \(N\) have been refused, up to time \(t\).

The time \(t\) associated with the failure \((s, N)\) 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 \( (((1, a)), [0, 1+\delta] \times \{c\}) \), 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 \(\delta\) allows us to infer that c will become available after a b has been performed. This is apparent from the following equivalence:

\[
P/(((1, a)), [0, 1+\delta] \times \{c\}), \delta) \equiv b \rightarrow c \rightarrow STOP
\]

We overload the after operator in the following way:

\[
P/(s, N) \equiv P/((s, N), end(s, N))
\]
\[
P/s \equiv P/((s, 0), end(s))
\]
\[
P/(s, t) \equiv P/((s, 0), t)
\]
\[
P/t \equiv P/((\langle\rangle, 0), t)
\]

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

\[
P/\langle\rangle \equiv P
\]
\[
(P/s)/u \equiv P/s \cap (u + end(s))
\]
\[
(P/t_{1})/t_{2} \equiv P/(t_{1} + t_{2})
\]
\[
(WAIT t ; P)/t \equiv P
\]

26
\[(\text{WAIT} t ; P)/t + \delta \equiv P\]
\[(P/(s, N))/t \equiv P/((s, N), \text{end}(s, N) + t)\]

### 3.6 Infinite behaviours

There are many cases where we would wish to consider the infinite behaviour of processes. A classic case is in 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 $\Phi$'. Indeed, part of the definition of a timed buffer, presented in [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: \((s, N) \in \mathcal{F}_T[P]\) tells us that if we watch \(P\) up until time \(\text{end}(s, N)\), then \((s, N)\) 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 the 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 un-
timed 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 \( T\Sigma^\infty \) and infinite refusal sets \( IRSET \) by

\[
T\Sigma^\infty \equiv \{ s \in ([0, \infty) \times \Sigma)^\omega \mid \forall t \in [0, \infty) \cdot s \uparrow t \in T\Sigma^\infty \}
\]

where the definition of the before operator \( \uparrow \) is extended in the obvious way to allow
infinite traces as arguments.

\[
T\Sigma^i \equiv T\Sigma^\infty \cup T\Sigma^\omega
\]

\[
IRSET \equiv \{ \bigcup Z \mid Z \in P(\text{RTOK}) \land \forall t \cdot \bigcup Z \uparrow t \in RSET \}
\]

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

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

\[
I(P) \equiv \{ (s, \infty) \mid \forall t \in [0, \infty) \cdot (s, \infty) \uparrow t \in P \}
\]

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

**Definition 3.6.3** We define \( \mathcal{I}_T : TCSP \rightarrow T\Sigma^i \times IRSET \) as follows:

\[
\mathcal{I}_T[P] \equiv \{ (s, \infty) \mid \forall t \in [0, \infty) \cdot (s, \infty) \uparrow t \in F_T[P] \}
\]

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

$$SI(P) = \{(s, e) \mid \forall t \in [0, \infty) \cdot (s, e) \uparrow t \in P \land \#s \text{ is finite}\}$$

Again, this extends to a semantic mapping $SI_T$:

Definition 3.6.5 The semantic mapping $SI_T : TCSP \to T\Sigma^* \times IRSET$ is defined as follows:

$$SI_T[P] = \{(s, e) \mid (s, e) \in I_T[P] \land \#s \text{ is finite}\}$$

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.

$$I_T[STOP] = \{((\cdot), e) \mid e \in IRSET\}$$

$$I_T[\bot] = \{((\cdot), e) \mid e \in IRSET\}$$

$$I_T[\text{SKIP}] = \{((\cdot), e) \mid \nexists \sigma(e)\}
\cup \{(((\cdot), \vee), e) \mid \nexists \sigma(e) \uparrow t)\}$$

$$I_T[\text{WAIT } t] = \{((\cdot), e) \mid \nexists \sigma(e)\}
\cup \{(((\cdot), \vee), e) \mid \nexists \sigma(e) \uparrow [t, t')\}\}$$

$$I_T[a \rightarrow P] = \{((\cdot), e) \mid a \nexists \sigma(e)\}
\{(((t, a)) \cap s, e) \mid a \nexists \sigma(e) \uparrow t) \land
(s - (t + \delta), e - (t + \delta)) \in I_T[P]\}$$

$$I_T[a : A \rightarrow P(a)] = \{((\cdot), e) \mid A \cap \sigma(e) = \emptyset
\cup
\{(((t, a)) \cap s, e) \mid A \cap \sigma(e) \uparrow t) = \emptyset
\land (s - (t + \delta), e - (t + \delta)) \in I_T[P(a)]\}\}$$

$$I_T[P \cap Q] = I_T[P] \cup I_T[Q]$$
\[ \mathcal{I}_T[P \Box Q] = \{(s, \mathcal{N}) \mid (\emptyset, \mathcal{N} \uparrow \text{begin}(s)) \in \mathcal{I}_T[P] \cap \mathcal{I}_T[Q] \land (s, \mathcal{N}) \in \mathcal{I}_T[P] \cup \mathcal{I}_T[Q] \} \]

\[ \mathcal{I}_T[P \parallel Q] = \{(s, \mathcal{N}_1 \cup \mathcal{N}_2) \mid (s, \mathcal{N}_1) \in \mathcal{I}_T[P] \land (s, \mathcal{N}_2) \in \mathcal{I}_T[Q] \} \]

\[ \mathcal{I}_T[P \backslash Y \parallel Q] = \{(s, \mathcal{N}_1 \cup \mathcal{N}_2) \mid (\mathcal{N}_1 \cup \mathcal{N}_2) \uparrow (X \cup Y) = \mathcal{N}_1 \uparrow X \cup \mathcal{N}_2 \uparrow Y \land (s \uparrow X, \mathcal{N}_1 \uparrow X) \in \mathcal{I}_T[P] \land (s \uparrow Y, \mathcal{N}_1 \uparrow Y) \in \mathcal{I}_T[Q] \land s \uparrow X \cup Y = s \} \]

\[ \mathcal{I}_T[P \parallel Q] = \{(s, \mathcal{N}) \mid \exists s_1, s_2 \mid s \in s_1 \parallel || s_2 \land (s_1, \mathcal{N}) \in \mathcal{I}_T[P] \land (s_2, \mathcal{N}) \in \mathcal{I}_T[Q] \} \]

\[ \mathcal{I}_T[\text{WAIT } t ; P] = \{(s + t, \mathcal{N}) \mid (s, \mathcal{N} \downarrow t) \in \mathcal{I}_T[P] \} \]

\[ \mathcal{I}_T[P ; Q] = \{(s, \mathcal{N}) \mid (s, \mathcal{N} \cup [0, \infty) \times \{\top\}) \in \mathcal{I}_T[P] \land \neg \in \sigma(s) \} \cup \{(s \uparrow (w + (t + \delta)), \mathcal{N}) \mid (w, \mathcal{N} \downarrow (t + \delta)) \in \mathcal{I}_T[P] \land \neg \in \sigma(s) \land (s \uparrow ((t, \top)), \mathcal{N} \uparrow t \cup [0, t) \times \{\top\}) \in \mathcal{I}_T[P] \} \]

\[ \mathcal{I}_T[f^{-l}(P)] = \{(s, \mathcal{N}) \mid (f(s), f(\mathcal{N})) \in \mathcal{I}_T[P] \} \]

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 \((s, \alpha, \mathcal{N}) \in \mathcal{E}_T[P]\) then \((s, [\alpha, \infty) \times \sigma(\mathcal{N} \downarrow \alpha)) \in S\mathcal{I}_T[P]\).

**Proof** This follows immediately from axiom 12 of \(TM_{FS}\). □
4 Factorising Proofs

In this chapter we present the first of our verification methods. 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 $(s, \alpha, \pi)$ underlying $TM_{FS}$, $T \Sigma_F \times [0, \infty] \times \text{RSET}$. Similarly, a behavioural specification on $TM_F$ is a predicate on $T \Sigma_F \times \text{RSET}$; a behavioural specification on $TM_S$ is a predicate on $T \Sigma_S \times [0, \infty]$; and a behavioural specification on $TM_T$ is a predicate on $T \Sigma_T$. We use the convention that the argument to a behavioural specification identifies the model employed, so for example $S(s, \alpha, \pi)$ is a behavioural specification on $TM_{FS}$. Observe 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

$P \ sat \ S(s, \alpha, \pi) \iff \forall (s, \alpha, \pi) \in \mathcal{E}_T[P] \cdot S(s, \alpha, \pi)$

$P \ sat \ S(s, \alpha, X) \iff \forall (s, \alpha, X) \in \mathcal{E}_T[P] \cdot S(s, \alpha, X)$

$P \ sat \ S(s, \pi) \iff \forall (s, \pi) \in \mathcal{F}_T[P] \cdot S(s, \pi)$

$P \ sat \ S(s, \alpha) \iff \forall (s, \alpha) \in \mathcal{S}_T[P] \cdot S(s, \alpha)$
\[ P \text{ sat } S(s) \equiv \forall s \in T_T[P] \cdot S(s) \]

In each model, every behavioural specification has a corresponding specification on processes. In \( TM_F \) the specification \( S(s, \alpha, \kappa) \) corresponds to the predicate \( U_S \) on processes, where \( U_S(P) = T \iff \forall (s, \alpha, \kappa) \in E_T[P] \cdot S(s, \alpha, \kappa). \) In Reed's notation we obtain \( \text{Pred}_{U_S} = S. \) However, not every specification on processes has a corresponding behavioural specification: for example, the specification \( ((1, a)), \emptyset, \emptyset) \in E_T[P] \) on process \( P \) cannot be written with a behavioural specification and the \( \text{sat} \) 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 **\( STOP \)

\[
(s = \langle \rangle) \Rightarrow S(s, \kappa)
\]

\[
\text{STOP sat } S(s, \kappa)
\]

**Rule **\( \bot \)

\[
(s = \langle \rangle) \Rightarrow S(s, \kappa)
\]

\[
\bot \text{ sat } S(s, \kappa)
\]
Rule **SKIP**

\[(s = \langle \rangle \land \exists \not\in \sigma(\mathcal{N})) \lor
(s = ((t, \not\in)) \land \exists \not\in \sigma(\mathcal{N} \uparrow t) \land t \geq 0) \Rightarrow S(s, \mathcal{N})\]

**Rule** **WAIT** \(t\)

\[s = \langle \rangle \land \exists \not\in \sigma(\mathcal{N} \uparrow t) \lor
s = ((t', \not\in)) \land t' \geq t \land \exists \not\in \sigma(\mathcal{N} \uparrow [t, t')) \Rightarrow S(s, \mathcal{N})\]

**Rule** \(a \rightarrow P\)

\[P \text{ sat } S_1(s, \mathcal{N}) \land a \not\in \sigma(\mathcal{N}) \Rightarrow S(s, \mathcal{N})\]

\[s = \langle \rangle \land (t, a) \not\in \sigma(\mathcal{N}) \land t \geq \delta \land \begin{cases} s_1(s', (t + \delta), (\mathcal{N} - (t + \delta))) \end{cases} \Rightarrow S(s, \mathcal{N})\]

**Rule** \(a : A \rightarrow P_a\)

\[\forall a \in A \cdot P_a \text{ sat } S_a(s, \mathcal{N}) \land \sigma(\mathcal{N}) \cap A = \emptyset \Rightarrow S(s, \mathcal{N})\]

\[s = \langle \rangle \land begin(s') \geq t + \delta \land \begin{cases} s_a(s', (t + \delta), (\mathcal{N} - (t + \delta))) \end{cases} \Rightarrow S(s, \mathcal{N})\]

\[a : A \rightarrow P_a \text{ sat } S(s, \mathcal{N})\]
Rule \( P_1 \cap P_2 \)

\[
\begin{align*}
P_1 & \text{ sat } S_1(s, \mathcal{N}) \\
P_2 & \text{ sat } S_2(s, \mathcal{N}) \\
S_1(s, \mathcal{N}) \lor S_2(s, \mathcal{N}) & \Rightarrow S(s, \mathcal{N})
\end{align*}
\]
\( P_1 \cap P_2 \text{ sat } S(s, \mathcal{N}) \)

Rule \( \bigcap_{i \in I} P_i \)

\[
\begin{align*}
\forall i \in I \bullet P_i & \text{ sat } S_i(s, \mathcal{N}) \\
\forall i \in I \bullet (S_i(s, \mathcal{N}) & \Rightarrow S(s, \mathcal{N}))
\end{align*}
\]
\( \bigcap_{i \in I} P_i \text{ sat } S(s, \mathcal{N}) \)

Rule \( P_1 \sqcap P_2 \)

\[
\begin{align*}
P_1 & \text{ sat } S_1(s, \mathcal{N}) \\
P_2 & \text{ sat } S_2(s, \mathcal{N}) \\
(S_1(s, \mathcal{N}) \lor S_2(s, \mathcal{N}) & \land S_1(() \uparrow \text{begin}(s)) \land S_2(() \uparrow \text{begin}(s))) \\
& \Rightarrow S(s, \mathcal{N})
\end{align*}
\]
\( P_1 \sqcap P_2 \text{ sat } S(s, \mathcal{N}) \)

Rule \( P_1 \parallel P_2 \)

\[
\begin{align*}
P_1 & \text{ sat } S_1(s, \mathcal{N}) \\
P_2 & \text{ sat } S_2(s, \mathcal{N}) \\
S_1(s, \mathcal{N}_1) & \land S_2(s, \mathcal{N}_2) \Rightarrow S(s, \mathcal{N}_1 \cup \mathcal{N}_2)
\end{align*}
\]
\( P_1 \parallel P_2 \text{ sat } S(s, \mathcal{N}) \)

Rule \( P_1 \mathcal{X}_{Y} P_2 \)

\[
\begin{align*}
P_1 & \text{ sat } S_1(s, \mathcal{N}) \\
P_2 & \text{ sat } S_2(s, \mathcal{N}) \\
\sigma(s_1, \mathcal{N}_1) & \subseteq X \land \sigma(s_2, \mathcal{N}_2) \subseteq Y \\
\land \sigma(\mathcal{N}_2) & \subseteq \Sigma - (X \cup Y) \\
\land S_1(s_1, \mathcal{N}_1) & \land S_2(s_2, \mathcal{N}_2) \land s_2 \in s_1 \mathcal{X}_{Y} s_2
\end{align*}
\]
\( P_1 \mathcal{X}_{Y} P_2 \text{ sat } S(s, \mathcal{N}) \)
Rule $P_1 || P_2$

$P_1$ sat $S_1(s, \mathbb{R})$
$P_2$ sat $S_2(s, \mathbb{R})$
$(s \in T_{merge}(s_1, s_2) \land S_1(s_1, \mathbb{R}) \land S_2(s_2, \mathbb{R})) \Rightarrow S(s, \mathbb{R})$

$P_1 || P_2$ sat $S(s, \mathbb{R})$

Rule $WAIT t : P$

$P$ sat $S_1(s, \mathbb{R})$
$(S_1(s - t, \mathbb{R} - t) \land \text{begin}(s) \geq t) \Rightarrow S(s, \mathbb{R})$

$WAIT t : P$ sat $S(s, \mathbb{R})$

Rule $P_1 ; P_2$

$P_1$ sat $S_1(s, \mathbb{R})$
$P_2$ sat $S_2(s, \mathbb{R})$
$(\forall \sigma \in \sigma(s) \land \forall I \in \text{TINT} \bullet S_1(s, \mathbb{R} \cup (I \times \{\emptyset\})) \Rightarrow S(s, \mathbb{R})$
$(S_1(s \uparrow \lceil(t, \emptyset), \mathbb{R} \uparrow t \cup [0, t) \times \{\emptyset\}) \land$
$s \uparrow (t; t + \delta) = () \land S_2(s - (t + \delta), \mathbb{R} - (t + \delta)) \Rightarrow S(s, \mathbb{R})$

$P_1 ; P_2$ sat $S(s, \mathbb{R})$

Rule $P \setminus A$

$P$ sat $S_1(s, \mathbb{R})$
$(S_1(s, \mathbb{R}) \land \text{act}(A)(s, \mathbb{R}) \land \sigma(\mathbb{R}) \subseteq A) \Rightarrow S(s \setminus A, \mathbb{R} - \mathbb{R})$

$P \setminus A$ sat $S(s, \mathbb{R})$

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

$P$ sat $S_1(s, \mathbb{R})$
$S_1(f(s), f(\mathbb{R})) \Rightarrow S(s, \mathbb{R})$

$f^{-1}(P)$ sat $S(s, \mathbb{R})$
Rule \( f(P) \)

\[
P \text{ sat } S_I(s, \mathcal{R}) \\
S_I(s, f^{-1}(\mathcal{R})) \Rightarrow S(f(s), \mathcal{R})
\]

\( f(P) \text{ sat } S(s, \mathcal{R}) \)

Rule \( \mu X \cdot F(X) \)

\[
\forall X \cdot X \text{ sat } S_I(s, \mathcal{R}) \Rightarrow F(WAIT, X) \text{ sat } S_I(s, \mathcal{R}) \\
S_I(s, \mathcal{R}) \Rightarrow S(s, \mathcal{R})
\]

\( \mu X \cdot F(X) \text{ sat } S(s, \mathcal{R}) \)

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 \( F(WAIT, X) \) preserves \( S_I \), then we must also establish that \( S_I \) is satisfiable:

Rule \( \mu X \cdot F(X) \) alternative version

\[
\forall X : TM_F \cdot (X \text{ sat } S_I(s, \mathcal{R}) \Rightarrow \\
F(WAIT, X) \text{ sat } S_I(s, \mathcal{R})) \\
S_I(s, \mathcal{R}) \Rightarrow S(s, \mathcal{R})
\]

\( \mu X \cdot F(X) \text{ sat } S(s, \mathcal{R}) \)

\( \exists P : TM_F \cdot P \text{ sat } S_I(s, \mathcal{R}) \)

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 \( P_I ; P_2 \)

\[
P_I \text{ sat } S_I(s, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \mathcal{R}) \\
\forall I \in TINT \cdot S_I(s, \mathcal{R} \cup (I \times \{\top\})) \Rightarrow S(s, \mathcal{R}) \\
s \cong s_I \cap (s_2 + t) \land \\not\in \sigma(s_I) \land \text{end}(R_I) \leq t \land \\
S_I(s_I \cap ((t, \top)), \mathcal{R} \cup ([0, t] \times \{\top\})) \land S_2(s_2, \mathcal{R}_2) \\
\Rightarrow S(s, \mathcal{R} \cup (\mathcal{R}_2 + t))
\]

\( (P_I ; P_2) \text{ sat } S(s, \mathcal{R}) \)
Rule $P_1 \triangleright P_2$

\[
P_1 \text{ sat } S_1(s, \alpha) \\
P_2 \text{ sat } S_2(s, \alpha)
\]
\[
\begin{align*}
\text{begin}(s) \leq t \land S_1(s, \alpha) \Rightarrow S(s, \alpha) \\
\text{begin}(s) \geq t + \delta \land S_1(\emptyset, \alpha \downarrow t) \land S_2(s \downarrow (t + \delta), \alpha \downarrow (t + \delta)) \Rightarrow S(s, \alpha)
\end{align*}
\]

$P_1 \triangleright P_2$ sat $S(s, \alpha)$

Rule $P_1 \triangleright P_2$

\[
P_1 \text{ sat } S_1(s, \alpha) \\
P_2 \text{ sat } S_2(s, \alpha)
\]
\[
\begin{align*}
S_1(s \uparrow t, \alpha \uparrow t \cup [0, t] \times \{\lor\}) \land \lor \not\in \sigma(s \uparrow t) \land \\
s \uparrow (t + 2\delta) = (\emptyset) \land S_2(s \downarrow (t + 2\delta), \alpha \downarrow (t + 2\delta)) \Rightarrow S(s, \alpha) \\
S_1(s \setminus (t', \lor)), \alpha \uparrow t' \cup [0, t') \times \{\lor\} \land t' \leq t \land \\
\lor \not\in \sigma(s) \land \lor \not\in \sigma(\alpha \uparrow t') \Rightarrow S(s, \alpha) \\
s = s' \setminus ((t'', \lor)) \land S_1(s' \setminus ((t', \lor))), \alpha \uparrow t' \cup [0, t') \times \{\lor\} \\
\land t' \leq t \land t' \leq t'' \land \lor \not\in \sigma(s) \land \lor \not\in \sigma(\alpha \uparrow [t', t'']) \Rightarrow S(s, \alpha)
\end{align*}
\]

$P_1 \triangleright P_2$ sat $S(s, \alpha)$

The rule for timeout simplifies considerably if the first process is unable to terminate. In this case, we have $\lor \not\in \sigma(P_i)$ (which is expressible as a behavioural specification).

Rule $P_1 \triangleright P_2$ special case version

\[
P_1 \text{ sat } \lor \not\in \sigma(s) \\
P_1 \text{ sat } S_1(s, \alpha) \\
P_2 \text{ sat } S_2(s, \alpha)
\]
\[
\begin{align*}
S_1(s \uparrow t, \alpha \uparrow t) \land s \uparrow (t + 2\delta) = (\emptyset) \\
\land S_2(s \downarrow (t + 2\delta), \alpha \downarrow (t + 2\delta)) \Rightarrow S(s, \alpha)
\end{align*}
\]

$P_1 \triangleright P_2$ sat $S(s, \alpha)$
4.3 Soundness

The basic processes

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

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

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

Rule

$$\begin{align*}
T(b) & \Rightarrow S(b) \\
P & \text{sat } S(b)
\end{align*}$$

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

One place operators

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

$$\mathcal{F}_T[\bigcirc_i P] = \{ b \mid T'(b)\} \cup \{ C(b) \mid f(b) \in \mathcal{F}_T[P] \land T(b)\}$$

and the corresponding inference rule is of the form

$$\begin{align*}
P & \text{sat } S_i(b) \\
T'(b) & \Rightarrow S(b) \\
(S_i(f(b)) \land T(b)) & \Rightarrow S(C(b))
\end{align*}$$

$$\bigcirc_i, P \text{ sat } S(b)$$

In the cases where $\forall b \cdot T'(b) = F$ (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:

\[ \mathcal{F}_T[\bigcirc_1 P] = \{ b \mid T'(b) \} \cup \{ C(b) \mid f(b) \in \mathcal{F}_T[P] \land T(b) \} \]

\[ P \text{ sat } S_1(b) \]

\[ T'(b) \Rightarrow S(b) \]

\[ (S_1(f(b)) \land T(b)) \Rightarrow S(C(b)) \]

\[ \vdash b' \in \mathcal{F}_T[\bigcirc_1 P] \Rightarrow b' \in \{ b \mid T'(b) \} \]

\[ \lor \]

\[ b' \in \{ C(b) \mid f(b) \in \mathcal{F}_T[P] \land T(b) \} \]

\[ \vdash b' \in \mathcal{F}_T[\bigcirc_1 P] \Rightarrow T'(b') \lor \exists b \cdot (b' = C(b) \land f(b) \in \mathcal{F}_T[P] \land T(b)) \]

\[ \vdash b' \in \mathcal{F}_T[\bigcirc_1 P] \Rightarrow S(b') \lor \exists b \cdot (b' = C(b) \land S_1(f(b)) \land T(b)) \]

\[ \vdash b' \in \mathcal{F}_T[\bigcirc_1 P] \Rightarrow S(b') \lor \exists b \cdot (b' = C(b) \land S(C(b))) \]

\[ \forall b' \in \mathcal{F}_T[\bigcirc_1 P] \Rightarrow S(b') \lor \exists b \cdot S(b') \]

\[ \vdash \bigcirc_1 P \text{ sat } S(b) \]

Two place operators

All the semantic definitions for $TM_P$ for the two place $TCSP$ operators $P_1 \cap P_2$, $P_1 \circ P_2$, $P_1 || P_2$, $P_1 \chi P_2$, $P_1 \parallel P_2$, $P_1 ; P_2$, and also those for the two place operators $P_1 ; P_2$, $P_1 \xi P_2$, $P_1 \nabla P_2$, may be written in the form

\[ \mathcal{F}_T[P_1 \bigcirc_2 P_2] = \{ C(b) \mid f_1(b_1) \in \mathcal{F}_T[P_1] \land f_2(b_2) \in \mathcal{F}_T[P_2] \land R(b, b_1, b_2) \} \]

The corresponding rule for operator $\bigcirc_2$ is:

\[ P_1 \text{ sat } S_1(b) \]

\[ P_2 \text{ sat } S_2(b) \]

\[ (S_1(b_1) \land S_2(b_2) \land R(b, b_1, b_2)) \Rightarrow S(C(b)) \]

\[ P_1 \bigcirc_2 P_2 \text{ sat } S(b) \]

39
and the soundness proof of this rule runs as follows:

\[ \mathcal{F}_T[P_1 \bigcirc P_2] = \{ C(b) \mid f_1(b_1) \in \mathcal{F}_T[P_1] \land f_2(b_2) \in \mathcal{F}_T[P_2] \land R(b, b_1, b_2) \} \]

\[ P_1 \text{ sat } S_1(b) \]
\[ P_2 \text{ sat } S_2(b) \]

\[ (S_1(b_1) \land S_2(b_2) \land R(b, b_1, b_2)) \Rightarrow S(C(b)) \]

\[ \vdash b' \in \mathcal{F}_T[P_1 \bigcirc P_2] \Rightarrow \exists b, b_1, b_2 \cdot (b' = C(b) \land f_1(b_1) \in \mathcal{F}_T[P_1] \land f_2(b_2) \in \mathcal{F}_T[P_2] \land R(b, b_1, b_2)) \]

\[ \vdash b' \in \mathcal{F}_T[P_1 \bigcirc P_2] \Rightarrow \exists b, b_1, b_2 \cdot (b' = C(b) \land S_1(b_1) \land S_2(b_2) \land R(b, b_1, b_2)) \]

\[ \forall b' \in \mathcal{F}_T[P_1 \bigcirc P_2] \Rightarrow \exists b, b_1, b_2 \cdot (b' = C(b) \land S(C(b))) \]

\[ \vdash P_1 \bigcirc P_2 \text{ sat } S(b) \]

Hence the rule for each two place operator is sound.

**Indexed operators**

The semantics for the two arbitrary choice operators \( \sqcap_{a \in A} P_a \) and \( a : A \rightarrow P_a \) may both be written in the form

\[ \mathcal{F}_T[\bigcirc_{a \in A} P_a] = \{ b \mid T'(b) \} \cup \{ C(b) \mid \exists a \in A \cdot f(b) \in \mathcal{F}_T[P_a] \land T(b) \} \]

and the corresponding proof rule is given by

\[ \forall a \in A \cdot P_a \text{ sat } S_a(b) \]
\[ T'(b) \Rightarrow S(b) \]
\[ \forall a \in A \cdot (S_a(f(b)) \land T(b) \Rightarrow S(C(b))) \]

\[ \bigcirc_{a \in A} P_a \text{ sat } S(b) \]

In the case of the nondeterministic choice we have \( \forall b \cdot T'(b) = F \), 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:

\[ \mathcal{F}_{T}\left[ \bigcirc_{a \in A} P_{a} \right] = \{ b \mid T'(b) \} \cup \{ C(b) \mid \exists a \in A \cdot f(b) \in \mathcal{F}_{T}[P_{a}] \land T(b) \} \]

\[ \forall a \in A \quad P_{a} \text{ sat } S_{a}(b) \]

\[ T'(b) \Rightarrow S(b) \]

\[ \forall a \in A \quad S_{a}(b) \land T(b) \Rightarrow S(C(b)) \]

\[ \vdash b' \in \mathcal{F}_{T}\left[ \bigcirc_{a \in A} P_{a} \right] \Rightarrow b' = C(b) \land f(b) \in \mathcal{F}_{T}[P_{a}] \land T(b) \]

\[ \vdash b' \in \mathcal{F}_{T}\left[ \bigcirc_{a \in A} P_{a} \right] \Rightarrow S(b') \lor \exists a, b \cdot b' = C(b) \land S(f(b)) \in \mathcal{F}_{T}[P_{a}] \land T(b) \]

\[ \vdash b' \in \mathcal{F}_{T}\left[ \bigcirc_{a \in A} P_{a} \right] \Rightarrow S(b') \lor \exists a, b \cdot b' = C(b) \land S(C(b)) \]

\[ \vdash \forall b' \in \mathcal{F}_{T}\left[ \bigcirc_{a \in A} P_{a} \right] \quad S(b') \]

\[ \vdash \bigcirc_{a \in A} P_{a} \text{ sat } S(b) \]

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

Recursion

The recursion induction theorem for Timed CSP states that if \( \mathcal{C} : TM_{F} \rightarrow TM_{F} \) is a contraction mapping, and \( S \) is a continuous satisfiable specification such that \( S(P) \Rightarrow S(\mathcal{C}(P)) \), then \( S(\mu X \cdot P(X)) \). 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

\[ \forall P \in TM_{F} \cdot (\neg S(P) \rightarrow (\exists t \in [0, \infty) \cdot Q(t) = P(t) \Rightarrow \neg S(Q)) \]

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

\[ \neg(P \text{ sat } S(s, \mathcal{R})) \]

\[ \vdash \exists(s, \mathcal{R}) \in P \cdot \neg S(s, \mathcal{R}) \]

\[ \vdash P(\text{end}(s, \mathcal{R}) + 1) = Q(\text{end}(s, \mathcal{R}) + 1) \Rightarrow (s, \mathcal{R}) \in Q \]

\[ \vdash \neg(Q \text{ sat } S(s, \mathcal{R})) \]

\[ \square \]

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 $T \Sigma_{T} \times RSET$. 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 $TCSP^+ = TCSP | \cdot X$, 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:

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

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 $\overline{\mathcal{F}_T}[P]$ 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 $\overline{C}$ corresponding to $\text{WAIT } \delta; F$ will be in $TM_F$.

By the recursion induction theorem for $\overline{TF}$, in order to demonstrate that $\mu X \cdot F(X) \text{ sat } S(s, \kappa)$, we need only show that the specification " sat $S(s, \kappa)$" is continuous, satisfiable, and that $X \text{ sat } S(s, \kappa) \Rightarrow \text{WAIT } \delta; F(X) \text{ sat } S(s, \kappa)$. We have already shown that any behavioural specification gives rise to a closed specification on $\overline{TF}$. It is also trivially true that $X_\emptyset \text{ sat } S(s, \kappa)$, 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 $S(s, \kappa)$ 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, then 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

\[
S(s, \pi) \subseteq T(s, \pi) \iff \forall (s, \pi) \bullet T(s, \pi) \Rightarrow S(s, \pi)
\]

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 \( SS[P] \). It is clear that

\[
SS[P](s, \pi) \iff (s, \pi) \in \mathcal{F}_T[P]
\]

Consider first the case of a one-place process constructor. If it is the case that \( Q_i P \) sat \( S(b) \), then consider the inference rule for \( Q_i P \) with \( SS[P] \) substituted for \( S_i \). The rule becomes

\[
P \text{ sat } SS[P] \\
T'(b) \Rightarrow S(b) \\
(SS[P](f(b)) \land T(b)) \Rightarrow S(C(b))
\]

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

\[
b \in \mathcal{F}_T[Q_i P] \Rightarrow S(b)
\]

But if \( Q_i P \) sat \( S(b) \), then the second and third premisses of the rule hold. Hence if it is the case that \( Q_i P \) 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 $F_T$. 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 $STOP$, $\bot$, $SKIP$ and
$WAIT t$ yield the strongest specifications as follows:

\[
SS[STOP](s, \emptyset) \iff s = \emptyset
\]
\[
SS[\bot](s, \emptyset) \iff s = \emptyset
\]
\[
SS[SKIP](s, \emptyset) \iff (s = \emptyset \land \vee \not\in \sigma(\emptyset))
\]
\[
\quad \lor (s = \langle (t, \vee) \rangle \land \vee \not\in \sigma(\emptyset \uparrow t) \land t \geq 0)
\]
\[
SS[WAIT t](s, \emptyset) \iff s = \emptyset \land \vee \not\in \sigma(\emptyset \uparrow t)
\]
\[
\quad \lor (s = \langle (t', \vee) \rangle \land t' \geq t \land \vee \not\in \sigma(\emptyset \uparrow [t, t'])
\]

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

\[
SS[a \rightarrow P](s, \emptyset) \iff s = \emptyset \land a \not\in \sigma(\emptyset)
\]
\[
\quad \lor (s = \langle (t, a) \rangle \land s' \land a \not\in \sigma(\emptyset \uparrow t) \land begin(s') \geq t + \delta
\]
\[
\quad \land SS[P](s \downarrow (t + \delta), (\emptyset \rightarrow (t + \delta)))
\]
\[
SS[P_1 \square P_2](s, \emptyset) \iff (SS[P_1](s, \emptyset) \lor SS[P_2](s, \emptyset))
\]
\[
\quad \land SS[P_1](\emptyset, \emptyset \uparrow begin(s)) \land SS[P_2](\emptyset, \emptyset \uparrow begin(s))
\]
\[
SS[a : A \rightarrow P](s, \emptyset) \iff s = \emptyset \land \sigma(\emptyset) \cap A = \emptyset
\]
\[
\quad \lor \exists a \in A \cdot s = \langle (t, a) \rangle \land begin(s') \geq t + \delta \land
\]
\[
\quad SS[P_a](s' \downarrow (t + \delta), (\emptyset \rightarrow (t + \delta))
\]
\[ SS[P_1 \cap P_2](s, \mathbb{R}) \Leftrightarrow SS[P_1](s, \mathbb{R}) \lor SS[P_2](s, \mathbb{R}) \]

\[ SS[\bigcap_{i \in I} P_i](s, \mathbb{R}) \Leftrightarrow \exists i \in I \cdot SS[P_i](s, \mathbb{R}) \]

\[ SS[P_1 \parallel P_2](s, \mathbb{R}) \Leftrightarrow \exists \mathbb{R}_1, \mathbb{R}_2 \cdot SS[P_i](s, \mathbb{R}_1) \land SS[P_2](s, \mathbb{R}_2) \land \mathbb{R} = \mathbb{R}_1 \cup \mathbb{R}_2 \]

\[ SS[P_1 \perp Y P_2](s, \mathbb{R}) \Leftrightarrow \exists \mathbb{R}_1, \mathbb{R}_2 \cdot SS[P_i](s \uparrow X, \mathbb{R}_1 \uparrow X) \land SS[P_j](s \uparrow Y, \mathbb{R}_2 \uparrow Y) \land \mathbb{R} = \mathbb{R}_1 \uparrow X \cup \mathbb{R}_2 \uparrow Y \land s = s \uparrow (X \cup Y) \]

\[ SS[P_1 \parallel P_2](s, \mathbb{R}) \Leftrightarrow \exists s_1, s_2 \cdot (s \in Tmerge(s_1, s_2)) \land SS[P_1](s_1, \mathbb{R}) \land SS[P_2](s_2, \mathbb{R})) \]

\[ SS[P_1 \parallel P_2](s, \mathbb{R}) \Leftrightarrow \exists s_1, s_2 \cdot (s \in Tmerge(s_1, s_2)) \land SS[P_1](s_1, \mathbb{R}) \land SS[P_2](s_2, \mathbb{R})) \]

\[ SS[P_1 \parallel P_2](s, \mathbb{R}) \Leftrightarrow SS[P](s \uparrow t \cap \{t, \sqrt{\}}) \land s \uparrow (t, t + \delta) = \emptyset \land SS[P_2](s \uparrow (t + \delta), \mathbb{R} \uparrow (t + \delta)) \land \mathbb{R} = \mathbb{R} \uparrow (t + \delta) \]

\[ SS[\text{WAIT } t ; P](s, \mathbb{R}) \Leftrightarrow SS[P](s \uparrow t, \mathbb{R} \uparrow t) \land \text{begin}(s) \geq t \]

\[ SS[P \setminus \mathbb{A}](s, \mathbb{R}) \Leftrightarrow \exists s_1 \cdot SS[P](s_1, \mathbb{R} \cup \{0, \text{end}(s_1)\} \times \mathbb{A}) \land s_1 \setminus \mathbb{A} = s \]

\[ SS[f^{-1}(P)](s, \mathbb{R}) \Leftrightarrow SS[P](f(s), f(\mathbb{R})) \]

\[ SS[f(P)](s, \mathbb{R}) \Leftrightarrow SS[P](s, f^{-1}(\mathbb{R})) \]

\[ SS[P_1 ; P_2](s, \mathbb{R}) \Leftrightarrow \exists s_1, s_2 \cdot (s \equiv s_1 \cap (s_2 + t)) \land \exists \mathbb{R}_1 \cdot \text{end}(\mathbb{R}_1) \leq t \land SS[P_1](s_1 \cap (t, \sqrt{\})), \mathbb{R}_1 \cup \{t, \sqrt{\}}) \land SS[P_2](s_2, \mathbb{R}_2) \]

45
\[ SS[ P_1 \triangleleft P_2 ](s, \mathfrak{R}) \Leftrightarrow \begin{align*}
\text{begin}(s) & \leq t \land SS[ P_1 ](s, \mathfrak{R}) \\
v \\
\text{begin}(s) & \geq t + \delta \land SS[ P_1 ]((), \mathfrak{R} \uparrow t) \\
\land SS[ P_2 ](s - (t + \delta), \mathfrak{R} - (t + \delta))
\end{align*} \]

\[ SS[ P_1 \triangleleft P_2 ](s, \mathfrak{R}) \Leftrightarrow \begin{align*}
SS[ P_1 ](s \uparrow t, \mathfrak{R} \uparrow t \cup [0, t) \times \{\forall\}) \land \exists \sigma(s \uparrow t) \land \\
S & (t + 2\delta) = () \land SS[ P_2 ](s - (t + 2\delta), \mathfrak{R} - (t + 2\delta)) \\
\lor \\
SS[ P_2 ](s \otimes ((t', \forall)), \mathfrak{R} \uparrow t' \cup [0, t') \times \{\forall\}) \land t' \leq t \\
\lor \\
\exists t' \cdot \forall \not\in \sigma(s) \land \\
SS[ P_2 ](s \otimes ((t', \forall)), \mathfrak{R} \uparrow t' \cup [0, t') \times \{\forall\}) \\
\land \forall \not\in \sigma(\mathfrak{R} \uparrow [t', t'']) \land t' \leq t \land t' \leq t''
\end{align*} \]

Recursion

Recall the inference rule for recursion:

\[
\forall X \bullet X \text{ sat } S_I(s, \mathfrak{R}) \Rightarrow F(WAIT \delta ; X) \text{ sat } S_I(s, \mathfrak{R}) \\
S_I(s, \mathfrak{R}) \Rightarrow S(s, \mathfrak{R})
\]

\[ \mu X \bullet F(X) \text{ sat } S(s, \mathfrak{R}) \]

If \( S_I(s, \mathfrak{R}) \Leftrightarrow SS[ \mu X \bullet F(X) ](s, \mathfrak{R}) \), then recalling that \( S_I(s, \mathfrak{R}) \Leftrightarrow (s, \mathfrak{R}) \in F_T[ \mu X \bullet F(X) ] \), we observe that the first premiss holds:

\[
Y \text{ sat } S_I(s, \mathfrak{R}) \\
\Rightarrow F_T[Y] \subseteq F_T[ \mu X \bullet F(X) ] \\
\Rightarrow F_T[ F(WAIT \delta ; Y) ] \subseteq F_T[ F(WAIT \delta ; \mu X \bullet F(X) ) ] \text{ (monotonicity) } \\
\Rightarrow F_T[ F(WAIT \delta ; X) ] \subseteq F_T[ \mu X \bullet F(X) ] \\
\Rightarrow F(WAIT \delta ; X) \text{ sat } S_I(s, \mathfrak{R})
\]

If \( \mu X \bullet F(X) \text{ sat } S(s, \mathfrak{R}) \) then \( S_I(s, \mathfrak{R}) \Rightarrow S(s, \mathfrak{R}) \), 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 $F_T$.

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 $F_T \mu X \cdot F(X)$ is the fixed point of the constructive function

\[ \hat{F}(X) = F(WAIT \delta; X), \]

and that $\hat{F}$ is $\delta$-constructive, in that

\[ X \uparrow t = Y \uparrow t \Rightarrow \hat{F}(X) \uparrow (t + \delta) = \hat{F}(Y) \uparrow (t + \delta) \]

Since we also have that

\[ \forall X, Y \in TF \cdot X \uparrow 0 = Y \uparrow 0 \]

we are able to conclude that

\[ \forall X, Y \in TF \cdot \hat{F}(X) \uparrow n\delta = \hat{F}(Y) \uparrow n\delta \]

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

\[ \forall X \in TF \cdot \hat{F}(X) \uparrow n\delta = \mu X \cdot F(X) \uparrow n\delta \]

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

\[ \text{end}(s, \kappa) < n\delta \Rightarrow ((s, \kappa) \in F_T \hat{F}(STOP)) \iff (s, \kappa) \in F_T (\mu X \cdot F(X)) \]

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.

\[ SS[\mu X \cdot F(X)](s, \kappa) \iff \forall n \cdot \text{end}(s, \kappa) < n\delta \Rightarrow SS[\hat{F}(\text{STOP})](s, \kappa) \]

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

\[ SS[\mu X \cdot F(X)](s, \kappa) \iff SS[\hat{F}^{\lfloor \text{end}(s, \kappa)/\delta \rfloor + 1}(\text{STOP})](s, \kappa) \]

([r] 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 behavioural specifications, that a search for an equivalent specification of that form would not succeed. We provide some sufficient 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_{FS} introduced in the next section, but it does not hold for TM_{FS}. For example, the behavioural specification defined by S(s, α, R) ≡ α < ∞ is not closed.

**Theorem 4.4.2** If ∃ P, Q ∈ TM_F • S(P ∩ Q) ∧ ¬S(P) then S cannot be written as a behavioural specification.

**Proof** Assume for a contradiction that there is a behavioural specification T(s, R) such that

\[ S(P) ⇔ P \text{ sat } T(s, R) \]

Then we have \( \mathcal{F}_T[P] \subseteq \mathcal{F}_T[P \cap Q] \), and so

\[ P \cap Q \text{ sat } T(s, R) \Rightarrow P \text{ sat } T(s, R) \]

yielding a contradiction. □

**Theorem 4.4.3** If S(P), S(Q), but ¬S(P ∩ 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:

$$E_T[P_1 || P_2] = SUP(I(P_1, P_2))$$

$$I(P_1, P_2) \equiv \{(s, \alpha, \#) \mid s \in Tmerge(s_1, s_2), \alpha = max\{\alpha_1, \alpha_2\} \cap (s_1, \alpha_1, \#) \in E_T[P_1] \cap (s_2, \alpha_2, \#) \in E_T[P_2]$$

If we define

$$P_1 \equiv a \rightarrow a \rightarrow STOP$$
$$P_2 \equiv a \rightarrow WAIT\ 1; STOP$$

then we have

$$(((1, a)), 1 + \delta, \#) \in E_T[P_1]$$
$$(((0, a)), 1 + \delta, \#) \in E_T[P_2]$$

and so

$$(((0, a), (1, a)), 1 + \delta, \#) \in I(P_1, P_2)$$

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

$$(((0, a), (1, a)), 2 + \delta, \#) \in I(P_1, P_2)$$

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

\[
P_1 \text{ sat } S_1(s, \alpha, \eta) \\
P_2 \text{ sat } S_2(s, \alpha, \eta) \\
s \in T_{\text{merge}}(s_1, s_2) \land S_1(s_1, \alpha_1, \eta) \land S_2(s_2, \alpha_2, \eta) \quad \Rightarrow \quad S(s, \max\{\alpha_1, \alpha_2\}, \eta)
\]

The rule is sound: any behaviour of \( P_1 \parallel P_2 \) 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

\[
S(s, \alpha, \eta) \iff s = \langle (0, a), (1, a) \rangle \Rightarrow \alpha = 2 + \delta
\]
a behavioural specification which holds of \( P_1 \parallel P_2 \). 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 \( S_1(\langle (1, a), 1+\delta, \emptyset \rangle) \) and \( S_2(\langle (0, a), 1+\delta, \emptyset \rangle) \), and so \( S(\langle (0, a), (1, a), 1+\delta, \emptyset \rangle) \). But this is not the case. Hence, the rule cannot be used in establishing this proof obligation.

The \( \text{SUP} \) operator presents a further obstacle: consider the set of processes \( \{P_n\} \) defined by \( P_n = n \rightarrow \text{WAIT}(1 - \frac{1}{n}) \), then the process \( P = (n : N \rightarrow P_n) \setminus N \) has the behaviour \( \langle (), 1, \emptyset \rangle \), even though there is no behaviour of \( n : N \rightarrow P_n \) 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 \( \langle (), \emptyset \rangle \); no single component behaviour 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 [Bla989]. In this approach, each failure \( (s, \eta) \) 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 \( \eta \) up to that time. The instability value \( \text{end}(s) \) is always included as a trivial instability. Hence, if process \( P \) in \( TMFS \) has a behaviour \( (s, \alpha, \eta) \), then the semantics of the process described in terms of instability values will contain all elements of the set

\[
\{ (s, \gamma, \eta) \mid \text{end}(s) \leq \gamma < \alpha \lor \text{end}(s) = \gamma = \alpha \}
\]

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

$$\hat{S} = \{(s, \gamma, \#) \mid \exists \alpha \bullet (s, \alpha, \#) \in S \land (\text{end}(s) \leq \gamma < \alpha \lor \text{end}(s) = \gamma)\}$$

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

$$\hat{P} = \{(s, \alpha, \#) \mid \exists \gamma \bullet (s, \gamma, \#) \in P \land \alpha = \sup\{\gamma \mid (s, \gamma, \#) \in P\}\}$$

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 $(s, \gamma, \#) \in \mathcal{E}_{TI}[P]$ is that the process may perform $s$ and refuse $\#$, and that internal activity may still be occurring after the performance of $s$ and refusal of $\# \upharpoonright \gamma$. 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 $P \satisfies S(s, \gamma, \#)$ if and only if $\hat{P} \satisfies T_{S}(s, \alpha, \#)$, where $T_{S}$ is defined by

$$T_{S}(s, \alpha, \#) \equiv S(s, \text{end}(s), \#) \land \forall \gamma \in [\text{end}(s), \alpha) \bullet S(s, \gamma, \#)$$

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

$$S(s, \alpha, \#) \equiv \alpha = \text{end}(s) + 3$$

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 $STOP$

$$(s = () \land \gamma = 0) \Rightarrow S(s, \gamma, N)$$

$STOP$ sat $S(s, \gamma, N)$

Rule $\bot$

$$(s = ()) \Rightarrow S(s, \gamma, N)$$

$\bot$ sat $S(s, \gamma, N)$

Rule $SKIP$

$$(s = () \land \gamma = 0 \land \vee \notin \sigma(N)) \Rightarrow S(s, \gamma, N)$$
$$(s = ((\gamma, \vee)) \land \vee \notin \sigma(N \uparrow \gamma)) \Rightarrow S(s, \gamma, N)$$

$SKIP$ sat $S(s, \gamma, N)$

Rule $WAIT t$

$$(s = () \land (\gamma = 0 \lor \gamma < t) \land \vee \notin \sigma(N \uparrow t)) \Rightarrow S(s, \gamma, N)$$
$$s = ((\gamma, \vee)) \land \gamma \geq t \land \vee \notin \sigma(N \uparrow [t, \gamma)) \Rightarrow S(s, \gamma, N)$$

$WAIT t$ sat $S(s, \gamma, N)$

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 $a \rightarrow P$

$P$ sat $T(s, \gamma, N)$
$$s = () \land \gamma = 0 \land a \notin \sigma(N) \Rightarrow S(s, \gamma, N)$$
$$s = ((t, a)) \land (s \uparrow (t + \delta)) \land \gamma \geq \text{end}(s) \land a \notin \sigma(N \uparrow t) \land T(s' \downarrow (t + \delta), \gamma \downarrow (t + \delta), \gamma \downarrow (t + \delta)) \Rightarrow S(s, \gamma, N)$$

$(a \rightarrow P)$ sat $S(s, \gamma, N)$

52
Rule \( a : A \rightarrow P_a \)

\[
\forall a \in A \bullet P_a \text{ sat } S_a(s, \gamma, \mathcal{R}) \\
(s = \langle \rangle \land \gamma = 0 \land A \cap \sigma(\mathcal{R}) = \emptyset) \Rightarrow S(s, \gamma, \mathcal{R}) \\
(s = \langle(t, a)\rangle \land (s \uparrow (t + \delta))) \land \gamma \geq \text{end}(s) \land A \cap \alpha(\mathcal{R} \uparrow t) \land \mathcal{R} \uparrow (t + \delta))) \\
\{ S_a(s \uparrow (t + \delta), \gamma \uparrow (t + \delta), (\mathcal{R} \uparrow (t + \delta))) \} \Rightarrow S(s, \gamma, \mathcal{R})
\]

\( a : A \rightarrow P_a \text{ sat } S(s, \gamma, \mathcal{R}) \)

Rule \( P_1 \cap P_2 \)

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
(S_1(s, \gamma, \mathcal{R}) \lor S_2(s, \gamma, \mathcal{R})) \Rightarrow S(s, \gamma, \mathcal{R})
\]

\( P_1 \cap P_2 \text{ sat } S(s, \gamma, \mathcal{R}) \)

Rule \( \prod_i P_i \)

\[
\forall i \in I \bullet P_i \text{ sat } S_i(s, \gamma, \mathcal{R}) \\
\forall i \in I \bullet (S_i(s, \gamma, \mathcal{R}) \Rightarrow S(s, \gamma, \mathcal{R}))
\]

\( \prod_{i \in I} P_i \text{ sat } S(s, \gamma, \mathcal{R}) \)

Rule \( P_1 \sqcap P_2 \)

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
(S_1(s, \gamma, \mathcal{R}) \lor S_2(s, \gamma, \mathcal{R}) \land \\
S_1(\langle \rangle, 0, \mathcal{R} \uparrow \text{begin}(s)) \land S_2(\langle \rangle, 0, \mathcal{R} \uparrow \text{begin}(s)) \lor \\
S_1(s, \text{end}(s), \mathcal{R}_1) \land S_2(s, \text{end}(s), \mathcal{R}_2) \land \\
(S_1(s, \gamma, \mathcal{R}_1) \lor S_2(s, \gamma, \mathcal{R}_2)) \Rightarrow S(s, \gamma, \mathcal{R}_1 \cup \mathcal{R}_2)
\]

\( P_1 \sqcap P_2 \text{ sat } S(s, \gamma, \mathcal{R}) \)

Rule \( P_1 \parallel P_2 \)

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
S_1(s, \text{end}(s), \mathcal{R}_1) \land S_2(s, \text{end}(s), \mathcal{R}_2) \land \\
(S_1(s, \gamma, \mathcal{R}_1) \lor S_2(s, \gamma, \mathcal{R}_2)) \Rightarrow S(s, \gamma, \mathcal{R}_1 \cup \mathcal{R}_2)
\]

\( P_1 \parallel P_2 \text{ sat } S(s, \gamma, \mathcal{R}) \)
Rule $P_1 \parallel_Y P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathbb{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathbb{R}) \\
\sigma(s) \subseteq X \cup Y \wedge R \uparrow X \cup Y = R_1 \uparrow X \cup R_2 \uparrow Y \wedge \\
S_1(s \uparrow X, \text{end}(s \uparrow X), R_1) \wedge S_2(s \uparrow Y, \text{end}(s \uparrow Y), R_2) \wedge \\
(S_1(s \uparrow X, \gamma, R_1) \vee S_2(s \uparrow Y, \gamma, R_2)) \Rightarrow S(s, \gamma, \mathbb{R})
\]

\[
P_1 \parallel_Y P_2 \text{ sat } S(s, \gamma, \mathbb{R})
\]

Rule $P_1 ||| P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathbb{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathbb{R}) \\
(s \in T\text{merge}(s_1, s_2) \wedge S_1(s_1, \text{end}(s_1), \mathbb{R}) \wedge S_2(s_2, \text{end}(s_2), \mathbb{R})) \wedge \\
(S_1(s_1, \gamma, \mathbb{R}) \vee S_2(s_2, \gamma, \mathbb{R})) \Rightarrow S(s, \gamma, \mathbb{R})
\]

\[
P_1 ||| P_2 \text{ sat } S(s, \gamma, \mathbb{R})
\]

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

\[
P \text{ sat } S_1(s, \gamma, \mathbb{R}) \\
(S_1(s \leftarrow t, \gamma \leftarrow t, \mathbb{R} \leftarrow t) \wedge \text{begin}(s) \geq t \wedge \gamma \geq t) \Rightarrow S(s, \gamma, \mathbb{R})
\]

\[
\text{WAIT } t \text{; } P \text{ sat } S(s, \gamma, \mathbb{R})
\]

Rule $P_1 \text{; } P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathbb{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathbb{R}) \\
\exists \mathbf{\alpha} \notin \sigma(s) \wedge \forall \mathbf{\beta} \in S_1(s, \gamma, \mathbb{R} \cup \{0, \beta\} \times \{\vee\}) \Rightarrow S(s, \gamma, \mathbb{R}) \\
\exists t \bullet \exists \mathbf{\alpha} \notin \sigma(s \uparrow t) \wedge S_1(s \uparrow t \circ ((t, \vee), t, \mathbb{R} \cup \{0, t\} \times \{\vee\}) \wedge \\
S_2(s \leftarrow (t + \delta), \gamma \leftarrow (t + \delta), \mathbb{R} \leftarrow (t + \delta)) \wedge s \uparrow (t, t + \delta) = () \wedge \gamma \geq \text{end}(s) \Rightarrow S(s, \gamma, \mathbb{R})
\]

\[
P_1 \text{; } P_2 \text{ sat } S(s, \gamma, \mathbb{R})
\]
Rule $P \setminus A$

$$P \text{ sat } S_i(s, \gamma, \kappa)$$

$$\text{end}(s) \leq \gamma \land \exists s_i \bullet s_i \setminus A \land$$

$$((\exists \beta > \gamma \bullet S_i(s_i, \beta, \kappa \cup \{0, \max\{\beta, \text{end}(\kappa)\} \times A\}) \lor$$

$$\text{end}(s) = \alpha \land S_i(s_i, \gamma, \kappa \cup \{0, \max\{\alpha, \text{end}(\kappa)\} \times A\})) \Rightarrow S(s, \gamma, \kappa)$$

$P \setminus A \text{ sat } S(s, \gamma, \kappa)$

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

$$P \text{ sat } S_i(s, \gamma, \kappa)$$

$$S_i(f(s), \gamma, f(\kappa)) \Rightarrow S(s, \gamma, \kappa)$$

$f^{-1}(P) \text{ sat } S(s, \gamma, \kappa)$

Rule $f(P)$

$$P \text{ sat } S_i(s, \gamma, \kappa)$$

$$S_i(s, \gamma, f^{-1}(\kappa)) \Rightarrow S(f(s), \gamma, \kappa)$$

$f(P) \text{ sat } S(s, \gamma, \kappa)$

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

$$\forall X \bullet X \text{ sat } S_i(s, \gamma, \kappa) \Rightarrow F(WAIT \delta ; X) \text{ sat } S_i(s, \gamma, \kappa)$$

$$S_i(s, \gamma, \kappa) \Rightarrow S(s, \gamma, \kappa)$$

$$\mu X \bullet F(X) \text{ sat } S(s, \gamma, \kappa)$$

Observe that, similar to the corresponding rule in the proof system for $TM_p$, the process variable $X$ in the first premiss ranges over all sets of instability values, not only those in $TM_{p_1}$. Again, we provide an alternative rule:

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

$$\forall X : TM_{p_1} \bullet (X \text{ sat } S_i(s, \gamma, \kappa)) \Rightarrow$$

$$F(WAIT \delta ; X) \text{ sat } S_i(s, \gamma, \kappa))$$

$$S_i(s, \gamma, \kappa) \Rightarrow S(s, \gamma, \kappa)$$

$$\mu X \bullet F(X) \text{ sat } S(s, \gamma, \kappa)$$

$\exists P : TM_{p_1} \bullet P \text{ sat } S_i(s, \gamma, \kappa)$.
We also provide some additional rules:

**Rule** $P_1; P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
(\forall \beta \in \mathcal{R} \land \exists \sigma : (s, \gamma, \mathcal{R}) \cup ([0, \beta] \times \{\vee\}) \Rightarrow S(s, \gamma, \mathcal{R}) \\
s \equiv s_1^{-1}(s_2) \land \exists \beta \in \mathcal{R} \land \begin{aligned}
&\exists \beta \in \mathcal{R} \land end(s_1) \leq \beta \leq begin(s_2) \\
&\land S_1(s_1^{-1}(\beta, \vee)), \beta, \mathcal{R} \uparrow \beta \cup ([0, \beta] \times \{\vee\}) \land \\
&\gamma \geq end(s) \land S_2(s_2 - \beta, \gamma - \beta, \mathcal{R}_2) \\
\end{aligned}
\Rightarrow S(s, \gamma, \mathcal{R})
\]

\[
(P_1; P_2) \text{ sat } S(s, \gamma, \mathcal{R})
\]

**Rule** $P_1 \triangleright_i P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
(S_1(s, \gamma, \mathcal{R}) \land begin(s) \leq t \Rightarrow S(s, \gamma, \mathcal{R}) \\
\begin{aligned}
&begin(s) \geq t + \delta \land S_1(\emptyset, \emptyset, \emptyset) \land \\
&S_2(s - (t + \delta), \gamma - (t + \delta), \mathcal{R} - (t + \delta))
\end{aligned}
\Rightarrow S(s, \gamma, \mathcal{R})
\]

\[
P_1 \triangleright_i P_2 \text{ sat } S(s, \gamma, \mathcal{R})
\]

**Rule** $P_1 \nleq_i P_2$

\[
P_1 \text{ sat } S_1(s, \gamma, \mathcal{R}) \\
P_2 \text{ sat } S_2(s, \gamma, \mathcal{R}) \\
(S_1(s \uparrow t, end(s \uparrow t), \mathcal{R} \uparrow t \cup [0, t] \times \{\vee\}) \land \\
\land S_2(s - (t + \delta), \gamma - (t + \delta), \mathcal{R} - (t + \delta))
\Rightarrow S(s, \gamma, \mathcal{R})
\]

\[
\exists t' \cdot S_1(s^{\sigma^{-1}}(t', \vee)), t', \mathcal{R} \uparrow t' \cup [0, t'] \times \{\vee\}) \land \\
\land \sigma \notin \mathcal{R} \land \forall \sigma : (\mathcal{R} \uparrow t') \land \gamma = t' \\
\Rightarrow S(s, \gamma, \mathcal{R})
\]

\[
P_1 \nleq_i P_2 \text{ sat } S(s, \gamma, \mathcal{R})
\]

56
Rule $P_i \not\vdash P_2$ special case version

$\quad P_i \text{ sat } \forall \not\in \sigma(s)$
$\quad P_i \text{ sat } S_i(s, \gamma, \kappa)$
$\quad P_2 \text{ sat } S_2(s, \gamma, \kappa)$
$\quad (S_i(s \uparrow t, \text{end}(s \uparrow t), \kappa \uparrow t \cup [0, t] \times \{\vee\})$
$\quad \land S_2(s \uparrow (t + \delta), \gamma \uparrow (t + \delta), \kappa \uparrow (t + \delta)) \Rightarrow S(s, \gamma, \kappa)$
$\quad \land s \uparrow (t, t + 2\delta) = () \land \gamma \geq \text{end}(s)$

$P_i \not\vdash P_2 \text{ sat } S(s, \gamma, \kappa)$

4.7 Soundness and Completeness

Both soundness and completeness of the set of inference rules for $TM_{EI}$ follow directly from the clauses of the semantic equation defining $\mathcal{E}_{EI}$, 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 the proof systems; whether the definition 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 these properties are defined on \( TM_F \); they can be immediately extended to apply to processes in \( TM_{FS} \), since we have the identity \( \mathcal{F}_T[P] = \text{fail}(\mathcal{E}_T[P]) \). 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 stability) when considering 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 buffers 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 unstable 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

\[ R = \text{STOP} \sqcap ((a \rightarrow \text{STOP}) \not\sqsubseteq \text{STOP}) \sqcap (a \rightarrow \text{STOP}) \]

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

\[ Q = \text{STOP} \sqcap (a \rightarrow \text{STOP}) \]

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 \((\), 0\)), 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 \((s, \mathcal{R})\) if \((t, a)\) is not in \(\mathcal{R}\) 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 \((s, \mathcal{R})\) 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 \(u \supseteq w \in \text{traces}(P)\) such that \(\text{begin}(w) \geq t_2 > t_1 \geq \text{end}(u)\), and \(Y \subseteq \Sigma\) we have

\[
(u \supseteq w, ([t_1, t_2) \times Y) \cup \mathcal{R}) \in \mathcal{F}_T[P] \Rightarrow (u \supseteq w, ([\text{end}(u), t_2) \times Y) \cup \mathcal{R}) \in \mathcal{F}_T[P]
\]

**Example**

The process

\[P \equiv a \rightarrow \text{WAIT 3} ; b \rightarrow \text{STOP}\]

is non-retracting. It has the behaviour

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

as a possible failure, so we may conclude that

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

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

\[((1, a), (5, b)), [4, 7) \times \{a\}\]

is a possible behaviour of \(P\), and hence that

\[((1, a), (5, b)), [1, 7) \times \{a\}\]

is a possible behaviour.

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

60
Theorem 5.1.2 If $P, Q$ and $P(a) (a \in A)$ are all non-retracting, then so are the following:

$$\bot, \text{STOP}, \text{SKIP}, \text{WAIT} \ t, \text{WAIT} \ t; \ P, \ a \rightarrow P, \ a : A \rightarrow P(a),$$

$$P \sqcap Q, \bigcap_{a \in A} P(a), \ P \sqcap Q, \ P \parallel Q, \ P_X \parallel_Y Q, \ P \parallel Q, \ f(P), \ f^{-1}(P).$$

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

Observe that even if $P$ and $Q$ are non-retracting, $P \setminus X$ and $P \parallel Q$ 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, $((\text{WAIT} \ 1; \ a \rightarrow \text{STOP}) \sqcap b \rightarrow \text{STOP})$ 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

$$((\text{WAIT} \ 1; \ a \rightarrow \text{STOP}) \sqcap b \rightarrow \text{STOP}) \setminus a \equiv (b \rightarrow \text{STOP}) \triangleright \text{STOP}$$

which retracts the offer of $b$ after $1$ unit of time. For the same reasons, timeout and tireout 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 $u \triangleright w \in \text{traces}(P)$, $\text{begin}(w) \geq t_2 > t_1 \geq \text{end}(u)$, and $Y \subseteq X$ we have

$$(u \triangleright w, ([t_1, t_2] \times Y) \cup N) \in \mathcal{F}_T[P] \Rightarrow (u \triangleright w, ([\text{end}(u), t_2] \times Y) \cup N) \in \mathcal{F}_T[P]\,$$

A non-retracting process is non-retracting on any subset of $\Sigma$, 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 $u^w \in \text{traces}(P)$, $\text{begin}(w) > t_2 > t_1 \geq \text{end}(u)$, and $Y \subseteq X$ we have

\[(u^w, ([t_1, t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P]\]

\[\Rightarrow (u^w, ([\text{end}(u \uparrow X), t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P]\]

Example

The process

\[(a \rightarrow \text{STOP}) || ((b \rightarrow \text{STOP}) \downarrow c \rightarrow d \rightarrow \text{STOP})\]

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

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

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

\[((((1, a), (3, c), (6, d)), [1, 5]) \times \{a\})\]

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 retraction of that offer, since it has been accepted.

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

\[(u^w, ([t_1, t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P]\]

\[\Rightarrow (u^w, ([\text{end}(u \uparrow Z), t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P]\]
Example

The process

\[ P = (b \rightarrow STOP) || (a \rightarrow STOP \square c \rightarrow d \rightarrow STOP) \]

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

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

is a failure of \( P \), so we may deduce that

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

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 \( w \in \text{traces}(P) \), \( \text{begin}(w) \geq t_2 > t_1 \geq \text{end}(u) \), and \( Y \subseteq X \) we have

\[
(u \upharpoonright w, ([t_1, t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P] \land S(u) \Rightarrow (u \upharpoonright w, ([\text{end}(u \upharpoonright Z), t_2] \times Y) \cup \mathbb{N}) \in \mathcal{F}_T[P]
\]

Example

The process

\[ \mu X \bullet a \rightarrow (b \rightarrow X \not\preceq STOP) \]

is non-retracting on \( \{a, b\} \) until \( \{a, b\} \) when \( s = () \lor \text{last}(s) = b \).
This is the most general form of non-retraction; all the other forms are special cases of it:

- Non-retraction is non-retraction on \( \Sigma \) until \( \Sigma \) when \( TRUE \)
- non-retraction on \( X \) is non-retraction on \( X \) until \( \Sigma \) 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 \( \Sigma \) until \( \Sigma \) when \( S \)
- non-retraction on \( X \) when \( S \) is non-retraction on \( X \) until \( \Sigma \) 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

\[
\exists u \cdot w \in traces(P), \begin{align*}
& begin(w) \geq t_2 > t_1 \geq end(u) \land Y \subseteq X \bullet \\
& (u \cdot w, [t_1, t_2] \times Y \cup \emptyset) \in \mathcal{F}[P] \land S(u) \land \\
& (u \cdot w, [end(u) \uparrow Z], t_2) \times Y \cup \emptyset \notin \mathcal{F}[P]
\end{align*}
\]

Now let \( Q \) be an arbitrary \( TM_F \) process such that

\[
Q(max\{end(u \cdot w), t_2\} + 1) = P(max\{end(u \cdot w), t_2\} + 1)
\]

It is clear that

\[
\exists u \cdot w \in traces(Q), \begin{align*}
& begin(w) \geq t_2 > t_1 \geq end(u) \land Y \subseteq X \bullet \\
& (u \cdot w, [t_1, t_2] \times Y \cup \emptyset) \in \mathcal{F}[Q] \land S(u) \land \\
& (u \cdot w, [end(u) \uparrow Z], t_2) \times Y \cup \emptyset \notin \mathcal{F}[Q]
\end{align*}
\]

64
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:

\[
\bot, \text{STOP}, \text{SKIP}, \text{WAIT } t, \text{WAIT } t ; P, a \rightarrow P, a : A \rightarrow P(a),
\]
\[
P \sqcap Q, \bigsqcap_{a \in A} P(a), P \sqcup Q, P \parallel Q, P \parallel_Y Q, P \parallel Q
\]

Further, if \( \text{inits}(P) \cap X = \emptyset \) then $P \not\triangleright Q$ 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

\[
P \equiv (a \rightarrow \text{STOP} \not\triangleright \text{STOP})
\]

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

\[
P \equiv a \rightarrow \text{STOP} \not\triangleright \text{STOP}
\]
\[
Q \equiv b \rightarrow \text{STOP} \not\triangleright \text{STOP}
\]
are non-retracting when \( s \neq \emptyset \), but the processes \( P \parallel Q \) and \( P \parallel_{\{s\}} Q \) are not non-retracting when \( s \neq \emptyset \). We have \( ((0, a)), [1, 2) \times \{b\} \in \mathcal{F}_T[P \parallel Q] \) and \( ((0, a)) \neq \emptyset \), but \( ((0, a)), [0, 2) \times \{b\} \notin \mathcal{F}_T[P \parallel Q] \). 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 following:

**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:

\[
\perp, \text{ STOP, SKIP, } \text{WAIT } t, \text{ WAIT } t; P, \\
P \cap Q, \bigcap_{a \in A} P(a), P \boxdot Q, P \parallel Q
\]

In addition, \( a \rightarrow P \) and \( a : A \rightarrow P(a) \) are non-retracting on \( X \) until \( Z \) when \( S(\text{tail}(s)) \). 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

\[
P \equiv (a \rightarrow \text{STOP}) \downarrow\downarrow \text{STOP} \\
Q \equiv \text{STOP}
\]

we have \( P \cap Q \) is non-retracting, but \( P \) is not. \( \Box \)

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 \( P_X \parallel_Y Q \) 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 \( (P_X \parallel_Y Q) \setminus C \) 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 \( P \downarrow Q \) 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

\[
(s, \mathcal{R}) \in \mathcal{F}_T[P] \Rightarrow (s, \mathcal{R}') \in \mathcal{F}_T[P]
\]

where \( \mathcal{R}' = \{(t, a) \mid a \in Y \land \exists t' \geq t \geq \text{end}(s) \bullet (t', a) \in \mathcal{R}\} \)

**Proof** This follows straightforwardly from the fact that \( \mathcal{R} \) is a finite union of refusal tokens, so the definition of non-retraction will apply to each of them in turn. \( \Box \)

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

\[
(s, \mathcal{R}) \in \mathcal{F}_T[P] \land A \subseteq \sigma(\mathcal{R} \uparrow [T, T + T']) \cap Y
\]

\[
\Rightarrow (s, \mathcal{R} \cup \text{end}(s), T) \times A) \in \mathcal{F}_T[P]
\]

**Proof** Follows immediately from lemma 5.1.14, (where \( \text{end}(s), T) \times A) \subseteq \mathcal{R}' \).

\( \Box \)

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

\[
(s, \mathcal{R}) \in \mathcal{SIT}_T[P] \land A \subseteq \sigma(\mathcal{R} \uparrow [T, T + T']) \Rightarrow (s, \mathcal{R} \cup \text{end}(s), T) \times A) \in \mathcal{SIT}_T[P]
\]

**Proof**

\[
\forall t \geq T + T' \bullet (s, \mathcal{R} \uparrow t) \in \mathcal{F}_T[P] \land A \subseteq \sigma(\mathcal{R} \uparrow t \uparrow [T, T + T'])
\]

so by corollary 5.1.15 we obtain

\[
\forall t \geq T + T' \bullet (s, \mathcal{R} \uparrow t \cup \text{end}(s), T) \times A) \in \mathcal{F}_T[P]
\]

so

\[
(s, \mathcal{R} \cup \text{end}(s), T) \times A) \in \mathcal{SIT}_T[P]
\]

as required. \( \Box \)
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 refuse 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

$$
\forall (s, N) \in F_T[P], T \in [0, \infty) \quad [T, T + t) \times A \subseteq N \Rightarrow (s \uparrow A) \uparrow [T, T + t) \neq ()
$$

**Example**

The process

$$\mu X \cdot a \rightarrow X$$

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

$P \text{ sat } \forall T \in [0, \infty) \quad ([T, T + t) \times A \subseteq N \Rightarrow (s \uparrow A) \uparrow [T, T + t) \neq ()$)

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 $a \in A$)
- $\text{WAIT } t'; P$ is $(t + t', A)$-responsive
- $a : B \rightarrow P(a)$ is $(t + \delta, B)$-responsive if $A \subseteq B, A \neq \emptyset$
- $\bigcap_{a \in B} P(a)$ is $(t, A)$-responsive
• \( \text{WAIT} \ t' ; \ a : B \rightarrow P(a) \) is \((\max\{t'', t + \delta\}, B)\)-responsive if \( A \subseteq B, A \neq \emptyset, t'' > t' \)

• \( P \sqcap Q, P \sqcap Q \) and \( P \parallel Q \) 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 \( F \circ \text{WAIT} \delta \) (\( F \) composed with \( \text{WAIT} \delta \)) preserves \((t, A)\)-responsive, then \( \mu X \cdot F(X) \) 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

\[
P \text{ sat } \forall T \in [0, \infty) \cdot (\{T, T + t\} \times A \cup [0, T + t] \times \{\bot\}) \subseteq \mathcal{R}
\Rightarrow (s \uparrow A) \uparrow [T, T + t] \neq \emptyset \lor \exists \in \sigma(s)
\]

**Example**

The process

\[
a \rightarrow \text{WAIT} 3 ; b \rightarrow \text{SKIP}
\]

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

\[
a \rightarrow \text{WAIT} 3 ; b \rightarrow \text{STOP}
\]

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

• \( \text{SKIP} \) is \((t, A)\)-responsive until termination, for any \( t > 0 \)

• \( \text{WAIT} \ t \) is \((t', A)\)-responsive until termination, for any \( t' > t \)
• $P \triangleright' Q$ is $(2t + \delta, A)$-responsive until termination
• $P \triangleright'' Q$ is $(2t + 2\delta, A)$-responsive until termination
• $P; R$ is $(2t, A)$-responsive
• $P \parallel R$ 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 $a \rightarrow P$, $P \sqcap Q$, $P \sqcap Q$, $\text{WAIT } t; P$, $P \parallel Q$, $P; Q$

To say a process is responsive is to say that it will respond to some event from $\Sigma$.

**Definition 5.2.8** A process is responsive if it is $\Sigma$-responsive.

**Definition 5.2.9** A process $P$ 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

$$\text{SKIP, \text{WAIT } t, a \rightarrow P, P \sqcap Q, P \sqcap Q,}$$
$$\text{\text{WAIT } t; P, P \parallel Q, P; Q, P \triangleright Q, P \triangleright Q}$$

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

The predicate $A$-responsive is not closed. For instance, if $F(X) \equiv \text{WAIT } t; X$ then $\mu X \cdot F(X) = \bot$, which is not $\{a\}$-responsive although each of the approximations $F^n(\mu X \cdot a \rightarrow X)$ 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 $P_X \parallel Y Q$ 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

$$P \text{ sat } [\text{end}(s), \text{end}(s) + t) \times A \subseteq \mathbb{R}$$

Example

The process

$$\mu X \cdot ((a \rightarrow X) \downarrow \text{STOP})$$

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 $A \cap B \neq \emptyset$
- $\parallel P(a), P \cap Q, P \bowtie Q$
- $\text{WAIT } t'; a : B \rightarrow P(a)$ if $t' < t, A \cap B \neq \emptyset$
- $P \parallel Q$
- $P \bowtie Q$ if $t' \geq t$

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, $t' \geq t$, $B \supseteq A$, 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 $P \leq_0 Q$ 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 $P \frac{1}{i} Q$ 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 $\mu X \cdot P \frac{1}{i} f(X)$ 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 $(s, \mathbb{N}) \in \mathcal{F}_T[P]$ we have:

$$T \geq \text{end}(s) \land [T, T + t) \times X \subseteq \mathbb{N} \Rightarrow (s, \mathbb{N} \cup [T, \infty) \times X) \in \mathcal{S}_T[P]$$

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

$$P \equiv (a \rightarrow STOP) \triangleright b \rightarrow STOP$$

is 3-prompt: from the information that

$$(((), \{0, 1\} \times \{b\} \cup [3, 6) \times \{a\}) \in \mathcal{F}_T[P]$$
we may conclude that
\[(\emptyset, [0, 1) \times \{b\} \cup [3, \infty) \times \{a\}) \in SI\_T[P]\]

However, \(P\) is not 1-prompt; it is not the case that
\[(\emptyset, [0, \infty) \times \{b\} \cup [3, 6) \times \{a\}) \in SI\_T[P]\]

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 \(s \in \text{traces}(\mathcal{F}_T[P])\) we have:
\[T' \geq \text{end}(s) + t \wedge T < T' - t \wedge (s, [T, T') \times X) \in \mathcal{F}_T[P] \Rightarrow (s, [T, \infty) \times X) \in SI\_T[P]\]

**Example**
The process
\[(a \rightarrow \text{STOP}) \upharpoonright b \rightarrow \text{STOP}\]

is weakly 3-prompt. We may conclude from
\[((2, b)), [1, 5) \times \{b\}) \in \mathcal{F}_T[P]\]

that
\[((2, b)), [1, \infty) \times \{b\}) \in SI\_T[P]\]

However, in contrast with the previous example, weak 3-promptness is not strong enough to conclude
\[([\emptyset, 1) \times \{b\} \cup [3, \infty) \times \{a\}) \in SI\_T[P]\]

from
\[([\emptyset, 1) \times \{b\} \cup [3, 6) \times \{a\}) \in \mathcal{F}_T[P]\]

**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

\[
P \text{ sat } ([T, T + t) \times \{a\} \subseteq \delta \wedge s \uparrow (T, T + t) = \emptyset) \Rightarrow \text{first}(s \uparrow (T + t)) \neq a
\]

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

\[
\text{STOP} \not\subseteq (\text{STOP} \cap a \rightarrow \text{STOP})
\]

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

- \( P \triangleleft Q, P \triangleright Q, P \parallel Q, \bigcap_{a \in A} P(a), f(P), f^{-t}(P) \) are all \( t \)-prompt
- \( \text{STOP}, \perp, \text{SKIP} \) are \( t \)-prompt
- \( a \rightarrow P, a : A \rightarrow P(a) \) are \( t + \delta \)-prompt
- \( \text{WAIT} t' \) is \( t \) prompt when \( t' \leq t \)
- \( P ; Q \) is \( 2t + \delta \)-prompt
- \( \text{WAIT} t'; P \) 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

\[
P \equiv \mu X \bullet (a \rightarrow \text{STOP}) \not\subseteq (\text{STOP} \not\subseteq X)
\]
\[
Q \equiv (\text{WAIT} 2 ; P) \frac{t}{T} a \rightarrow \text{STOP}
\]

we have \( P \) and \( Q \) are both \( 4 \)-prompt, but the process \( P \parallel Q \) is not \( t \)-prompt for any \( t \leq T \), 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 \((s, \mathcal{R}) \in \mathcal{F}_T[P]\):

\[
\forall T \geq \text{end}(s), Y \subseteq X \cdot (T, T + t) \times Y \subseteq \mathcal{R} \Rightarrow (s, \mathcal{R} \cup [T, \infty) \times Y) \in \mathcal{S}_T[P]
\]

**Example**

The process

\[
a \rightarrow \text{WAIT 3} ; b \rightarrow a \rightarrow \text{STOP}
\]

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 \(P \mid_Y Q\) 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 \((s, \mathcal{R}) \in \mathcal{F}_T[P]\) with \(T \geq \text{end}(s) \wedge [T, T + t) \times (A \cup B) \subseteq \mathcal{R}\). Then

\[
(s, \mathcal{R} \cup [T, T + t) \times (A \cup [T, \infty) \times (A \cup B)) \in \mathcal{S}_T[P]
\]

**Definition 5.3.9** A process \(P\) is \(t\)-prompt on \(X\) when \(S\) if for any

\((s, \mathcal{R}) \in \mathcal{F}_T[P]\):

\[
\forall T \geq \text{end}(s) \cdot S(s) \wedge [T, T + t) \times X \subseteq \mathcal{R} \Rightarrow (s, \mathcal{R} \cup [T, \infty) \times X) \in \mathcal{S}_T[P]
\]
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

$$
\mu X \bullet (b \rightarrow ((b \rightarrow X) \cap a \rightarrow X) \quad \Box c \rightarrow \text{WAIT } S ; a \rightarrow X)
$$

is both 1-prompt on \{a\} when $\text{last}(s) = b$, and strongly 1-prompt on \{a\} when $\text{last}(s) = b$. 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

$$
P \equiv \text{WAIT } 4 ; a \rightarrow \text{STOP}
$$

$$
Q \equiv \text{STOP}
$$

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

$$
\exists (s, \mathcal{N}) \in \mathcal{F}_T[\mathcal{P}], T \geq \text{end}(s), Y \subseteq X \bullet
\quad S(s) \land [T, T + t) \times Y \subseteq \mathcal{N} \land (s, \mathcal{N} \cup [T, \infty) \times Y) \notin \mathcal{F}_T[P]
$$

76
But if \((s,\mathbb{N} \cup [T,\infty) \times Y) \notin SI_T[P]\), then
\[
\exists T' > T \cdot (s,\mathbb{N} \cup [T, T') \times Y) \notin SI_T[P]
\]
Define \(T'' = 1 + \max\{\text{end}(s,\mathbb{N}), T'\}\). 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. \(\square\)

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 \(P \upharpoonright Y Q\) is prompt

**Proof** Let \(P\) be \(t_1\)-prompt, and \(Q\) be \(t_2\)-prompt. We will prove that \(P \upharpoonright Y Q\) is \(t_1 + t_2\)-prompt. Consider \((s,\mathbb{N}) \in SI_T[P \upharpoonright Y Q]\), and that \([T, T + t_1 + t_2] \times Z \subseteq \mathbb{N}\) for some \(T \geq \text{end}(s)\). Without loss of generality, assume \(Z \subseteq X \cup Y\). Define
\[
s_1 = s \uparrow X
\]
\[
s_2 = s \uparrow Y
\]
Then
\[
\exists \mathbb{N}_1, \mathbb{N}_2 \cdot (s_1, \mathbb{N}_1) \in SI_T[P] \land (s_2, \mathbb{N}_2) \in SI_T[Q] \land \mathbb{N} = \mathbb{N}_1 \cup \mathbb{N}_2 \land \\
\mathbb{N}_1 \uparrow (X \setminus Y) = \mathbb{N} \uparrow (X \setminus Y) \land \mathbb{N}_2 \uparrow (Y \setminus X) = \mathbb{N} \uparrow (Y \setminus X)
\]
and so
\[
[T, T + t_1 + t_2] \times (Z \cap (X \setminus Y)) \subseteq \mathbb{N}_1
\]
\[
\land \quad [T, T + t_1 + t_2] \times (Z \cap (Y \setminus X)) \subseteq \mathbb{N}_2
\]
and so
\[
(s_1, \mathbb{N}_1 \cup [T, \infty) \times (Z \cap (X \setminus Y))) \in SI_T[P]
\]
\[
\land \quad (s_2, \mathbb{N}_2 \cup [T, \infty) \times (Z \cap (Y \setminus X))) \in SI_T[Q]
\]
Define
\[
Z_1 = X \cap Y \cap Z \cap \sigma(\mathbb{N}_1 \uparrow [T + t_1, T + t_1 + t_2])
\]
\[
Z_2 = X \cap Y \cap Z \cap \{a | [T + t_1, T + t_1 + t_2] \times \{a\} \subseteq \mathbb{N}_2\}
\]
\[
Z_3 = X \cap Y \cap Z \cap \sigma(\mathbb{N}_2 \uparrow [T + t_1, T + t_1 + t_2])
\]
\[
Z_4 = X \cap Y \cap Z \cap \{a | [T + t_1, T + t_1 + t_2] \times \{a\} \subseteq \mathbb{N}_1\}
\]

77
Then lemma 5.1.15 yields that

\[ (s_1, \mathcal{N}_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T, T + t_1) \times Z_1) \in \mathcal{F}_T[P] \]

\[ \Rightarrow (s_1, \mathcal{N}_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T + t_1, T + t_1 + t_2) \times Z_1 \cup Z_2) \in \mathcal{F}_T[P] \]

\[ \Rightarrow (s_1, \mathcal{N}_1 \cup [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T + t_1, \infty) \times Z_1 \cup Z_4) \in \mathcal{F}_T[P] \]

By symmetry we also obtain that

\[ (s_2, \mathcal{N}_2 \cup [T, \infty) \times (Z \cap (Y \setminus X)) \cup [T + t_1, \infty) \times Z_2 \cup Z_3) \in \mathcal{F}_T[Q] \]

and so

\[ (s, [T, \infty) \times (Z \cap (X \setminus Y)) \cup [T + t_1, \infty) \times (Z \cap (Y \setminus X)) \cup \mathcal{N}_1 \cup \mathcal{N}_2 \cup [T + t_1, \infty) \times (Z_1 \cup Z_2 \cup Z_3 \cup Z_4)) \in \mathcal{SIT}_T[P, X \parallel Y \parallel Q] \]

thus we obtain

\[ (s, \mathcal{N} \cup [T, \infty) \times Z) \in \mathcal{F}_T[P, X \parallel Y \parallel Q] \]

as required. \( \Box \)

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 \( P, X \parallel Y \parallel Q \) 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 \( P, X \parallel Y \parallel Q \) 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

\[ \forall (s, \mathcal{N}) \in \mathcal{SIT}_T[P], a \in A \quad \bullet \quad (s^{-}(t, a), \mathcal{N}) \in \mathcal{SIT}_T[P] \Rightarrow \]

\[ \forall b \in A \quad \bullet \quad (s^{-}(t, b), \mathcal{N} \cup (I(\mathcal{N} \uparrow A) \times A)) \in \mathcal{SIT}_T[P] \]
Example

The process
\[ \mu X \bullet (a \rightarrow X \sqcup b \rightarrow X \sqcup c \rightarrow X) \]

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:

- \( \text{STOP} \), \( \perp \)
- \( \text{SKIP} \), \( \text{WAIT} t \) if \( \neq \notin A \) or \( A \subseteq \{\neq\} \)
- \( P \sqcup Q, P \sqcap Q, \sqcap_a P(a) \)
- \( P \parallel Q, P \parallel Q, P ; Q, \text{WAIT} t ; P \)
- \( a : B \rightarrow P(a) \) if \( A \subseteq B \) or \( A \cap B = \emptyset \)
- \( P_X \parallel_Y Q \) if \( (A \subseteq X \vee A \cap X = \emptyset) \) and \( (B \subseteq Y \vee B \cap Y = \emptyset) \)
- \( P \backslash B \) if \( A \subseteq B \) or \( A \cap B = \emptyset \)
- \( P \uparrow Q \) and \( P \downarrow Q \)
- \( f^{-1}(P) \) 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 \( \{P_a\} \rightarrow P \), with each \( P_a \) impartial on \( A \). Now let

\[ (s^\wedge((t, a)), \mathbb{N}) \in \mathcal{ST}_T[P] \]

Given \( T(> t) \) we have

\[ \exists N_T \bullet n > N_T \Rightarrow P_a \uparrow T = P \uparrow T \]
\[ \Rightarrow (s^\wedge((t, a)), \mathbb{N} \uparrow T) \in \mathcal{ST}_T[P_a] \]
\[ \Rightarrow \forall b \in A \bullet (s^\wedge((t, b)), \mathbb{N} \uparrow T \cup (I((\mathbb{N} \uparrow T) \uparrow A) \times A)) \in \mathcal{ST}_T[P_a] \]
\[ \Rightarrow \forall b \in A \bullet (s^\wedge((t, b)), \mathbb{N} \uparrow T \cup (I((\mathbb{N} \uparrow T) \uparrow A) \times A)) \in \mathcal{ST}_T[P] \]
This is true for all $T$, so

$$\forall b \in A \cdot (s \cap ((t, b)), \emptyset \cup (I(\emptyset \uparrow A) \times A) \in ST[P]$$

as required. $\square$

**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 $P \parallel Y Q$ 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 $P \parallel Y Q$ is $(t_1 + t_2, X \cup Y)$-responsive.

Assume not, for a contradiction. Then for some $(s, \emptyset) \in F_T[P \parallel Y Q]$, and for some $T \in [0, \infty)$, we have

$$([T, T + t_1 + t_2] \times (X \cup Y) \subseteq \emptyset \land (s \uparrow (X \cup Y) \uparrow [T, T + t_1 + t_2]) = \emptyset$$

and without loss of generality we may assume

$$\text{end}(s) \leq T + t_1 + t_2 \land \text{end}(\emptyset) \leq T + t_1 + t_2$$

Hence $\text{end}(s) \leq T$. Define

$$s_1 = s \uparrow X$$
$$s_2 = s \uparrow Y$$

Then

$$\exists \emptyset, \emptyset \cdot \emptyset = \emptyset \uparrow \emptyset \land (s_1, \emptyset) \in F_T[P] \land (s_2, \emptyset) \in F_T[Q]$$

and so

$$([T, T + t_1 + t_2] \times (X \setminus Y) \subseteq \emptyset \land [T, T + t_1 + t_2] \times (Y \setminus X) \subseteq \emptyset$$

We have two cases to consider:

**Case** $[T + t_2, T + t_2 + t_1] \subseteq I(\emptyset \uparrow (X \cap Y))$

$$\Rightarrow (s_1, [T + t_2, T + t_2 + t_1] \times X) \in F_T[P]$$
$$\Rightarrow (s_1 \uparrow X) \uparrow (T + t_2, T + t_2 + t_1) \neq \emptyset$$
$$\Rightarrow (s \uparrow (X \cup Y)) \uparrow (T, T + t_2 + t_1) \neq \emptyset$$
which yields a contradiction.

Case \( [T + t_2, T + t_2 + t_1] \not\subseteq I(\mathbb{R}_T \uparrow (X \cap Y)) \)

\[
\Rightarrow \exists [t', t''] \subseteq [T + t_2, T + t_2 + t_1] \cdot \{t', t''\} \times (X \cap Y) \subseteq \mathbb{R}_T
\]
\[
\Rightarrow [t', t''] \times Y \subseteq \mathbb{R}_T
\]
\[
\Rightarrow (s_{t_2}, [\text{end}(s_{t_2}), t''] \times Y) \in \mathcal{F}_T[Q]
\]
\[
\Rightarrow (s_{t_2}, [T, T + t_2] \times Y) \in \mathcal{F}_T[Q]
\]
\[
\Rightarrow (s_{t_2} \uparrow Y) \uparrow (T, T + t_2) \neq \emptyset
\]
\[
\Rightarrow (s \uparrow (X \cup Y)) \uparrow (T, T + t_2 + t_1) \neq \emptyset
\]

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

\[
\forall s \in \text{traces}(\mathcal{F}_T[P \setminus A]), w \in \text{traces}(\mathcal{F}_T[P]) \cdot w \setminus A = s \Rightarrow \#(w \uparrow \text{end}(s)) < n
\]

Example

The process

\[
\mu X \cdot a \rightarrow b \rightarrow c \rightarrow X
\]

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

\[
\mu X \cdot a \rightarrow X \bowtie b \rightarrow X
\]

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 \( P \parallel_X Y Q \) 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

\[
\forall s \in \text{traces}(F_T[\mu X \cdot P]), \exists n, \forall w \in \text{traces}(F_T[P]) \cdot w \setminus A = s \Rightarrow \#w < n
\]

Example

The counter process

\[
\mu Y \cdot (\mu X \cdot up \to (X \uparrow down \to SKIP) \cap down \to SKIP) \setminus Y
\]

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 \( P \parallel_X Y Q \) is limited on \( A \cup B \).

Proof We have

\[
\exists n \cdot w \in \text{traces}(P) \land s \in \text{traces}(P \setminus A) \land w \setminus A = s \Rightarrow \#w \uparrow \text{end}(s) < n
\]

and

\[
\exists m \cdot w \in \text{traces}(Q) \land s \in \text{traces}(Q \setminus A) \land w \setminus A = s \Rightarrow \#w \uparrow \text{end}(s) < m
\]

Now consider

\[
w \in \text{traces}(P \parallel_X Y Q) \land s \in \text{traces}((P \parallel_X Y Q) \setminus A \cup B) \land w \setminus A \cup B = s
\]

Since

\[
A \cap Y = B \cap X = \emptyset
\]

82
we have
\[ s \in \text{traces}(P \setminus A \cup B, Q \setminus B) \]
so
\[ s \uparrow X = s \uparrow (X \setminus A) \in \text{traces}(P \setminus A) \]
Now \( w \setminus (A \cup B) = s \) so
\[ s \uparrow X = (w \setminus (A \cup B)) \uparrow X = (w \setminus A) \uparrow X = (w \uparrow X) \setminus A \]
Now
\[ w \uparrow X \in \text{traces}(P) \]
and so
\[ \#(w \uparrow X) \uparrow \text{end}(s \uparrow X) < n \]
By symmetry,
\[ \#(w \uparrow Y) \uparrow \text{end}(s \uparrow Y) < m \]
Hence
\[ \#(w \uparrow X \cup Y) \uparrow \text{max}\{\text{end}(s \uparrow X), \text{end}(s \uparrow Y)\} < m + n \]
and so \( \#w \uparrow \text{end}(s) < m + n \) as required. \( \square \)

**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
\[ s \in \text{traces}(P \setminus A \setminus B) \land w \in \text{traces}(P \setminus A) \land w \setminus B = s \]
Then
\[ \exists u \bullet u \in \text{traces}(P) \land u \setminus A = w \]
Then
\[ u \setminus (A \cup B) = s \]
so
\[ \#u \uparrow \text{end}(s) < n \]
so \( \#w \uparrow \text{end}(s) < n \). \( \square \)

Now let \( P \) be weakly \( t \)-prompt. We will prove that \( P \setminus A \) is weakly \( nt \)-prompt. For \( s \in \text{traces}(\mathcal{FT}[P \setminus A]) \) let
\[ T' \geq \text{end}(s) + nt, T \leq T' - nt, (s, [T, T') \times X) \in \mathcal{FT}[P \setminus A] \]

83
From lemma A.1.2 we have that
\[ \exists w \cdot (w, [T, T')] \times X \cup \{0, T'\} \times A) \in \mathcal{F}_T[P] \land w \setminus A = s \]
Since \( P \) is \( n \)-limited, and \( T' \geq \text{end}(s) + nt \), we have that
\[ \exists T'' \cdot T \leq T'' \leq T' - t \land \uparrow w(T'', T'' + t] = \langle \rangle \]
But then
\[ (w \uparrow (T'' + t), [T, T'' + t) \times X \cup [0, T' + t) \times A) \in \mathcal{F}_T[P] \]
so by weak promptness of \( P \) we obtain
\[ (w \uparrow (T'' + t), [T, \infty) \times X \cup [0, \infty) \times A) \in \mathcal{S}_T[P] \]
so \((s, [T, \infty) \times X) \in \mathcal{S}_T[P]\) as required. \( \square \)

**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 \( \mathcal{M}_T \) if
\[ \forall tr \in T[P], \exists w \in A^* \land \text{w in tr) } \Rightarrow \#w < n \cdot \text{w} \setminus A = s \]

A CSP process \( P \) is limited on \( A \) in \( \mathcal{M}_T \) 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 \( \mathcal{M}_T \) if
\[ \forall s \in T[P \setminus A], \exists n, \forall w \in T[P] \cdot (w \setminus A = s \Rightarrow \#tr < n) \]

**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
\[ (s, \alpha) \in \text{stab}(E_T[P]) \Rightarrow \alpha < \infty \]
Definition 5.6.2 A TCSP process $P$ is stable when $S$ if

$$((s, \alpha) \in \text{stab}(E_T[P]) \land S(s)) \Rightarrow \alpha < \infty$$

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

$$(s, \alpha) \in \text{stab}(E_T[P]) \Rightarrow \alpha \leq \text{end}(s) + t$$

Example

The process

$$\mu X ((a \rightarrow X)^{a} b \rightarrow a \rightarrow X)$$

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 $t' \leq t$, then

- $P || Q, P \mid\mid Q, P \bigtriangleup Q, P \nabla Q, P \sqhat Q$ are all $t$-stable
- $P ; Q$ is $(2t + \delta)$-stable
- $a \rightarrow P$ and $a : A \rightarrow P(a)$ are $(t + \delta)$-stable
- $\text{WAIT } t' ; a \rightarrow P$ and $\text{WAIT } t' ; a : A \rightarrow P(a)$ are $(t + \delta)$-stable
- $f(P)$ and $f^{-1}(P)$ are $t$-stable
- $\text{WAIT } t' ; P$ is $(t + \delta)$-stable

Further, $t$-stable is closed in $TMFS$.

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 $P$ be $t$-stable. Consider $s \in \text{traces}(P \setminus A)$. Then

$$\exists n \cdot w \in \text{traces}(P) \land w \setminus A = s \Rightarrow \#w \uparrow \text{end}(s) < n$$

We will prove that

$$(s, \alpha, 0) \in \mathcal{E}_T[P \setminus A] \Rightarrow \alpha \leq \text{end}(s) + nt$$

By the definition of the hiding operator, we have

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

for some $\alpha, \beta$ such that $\text{end}(w) \leq \beta \leq \alpha$. Now if

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

then axiom 8 of $TM_F$ yields that $t_2 \leq t_1 + t$, since if $t_2 > t_1 + t$ then

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

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

$$(w \uparrow ((t_2 + t_1 + t)/2) \cap ((t_2, a_2)), [0, (t_2 + t_1 + t)/2) \times A)$$

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

Hence

$$\text{end}(w) \leq \text{end}(s) + \#(w \uparrow \text{end}(s))t \leq \text{end}(s) + (n - 1)t$$

and so

$$(w, \alpha, 0) \in \mathcal{E}_T[P] \Rightarrow \alpha \leq \text{end}(s) + nt$$

so $\beta$ defined above is less than $\text{end}(s) + nt$.

Therefore $(s, \alpha, 0) \in \mathcal{E}_T[P \setminus A] \Rightarrow \alpha < \infty$, 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 \( \subseteq \) ('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

- \( S(P) \)
- \( P \subseteq Q \)
- \( \forall P, Q \cdot ((S(P) \land P \subseteq Q) \Rightarrow T(Q)) \)

The third condition may be thought of as the corresponding refinement relation between \( S \) and \( T \). The 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 the aspects of behaviour we are able to describe. The mapping \( \Pi \) 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 \( \Pi \) is the mapping from \( TMFS \) to \( MT \), then the information that \((((1, a), (3, b)), 4, [0, 3] \times \{b\})\) is a possible behaviour of a process in \( TMFS \) translates under the mapping \( \Pi \) to the information that \((a, b)\) is a possible behaviour in \( MT \).

In considering refinement relations between processes in different models in the hierarchy, we will need to use the projection mappings to translate the semantics of the process in the higher 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, the information in \( MT \) 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 \( \Theta : TCSP \to CSP \), which removes the timing information from the syntactic description of a process. We will be interested in conditions which yield \( \Theta(Q) \subseteq Q \), since in those cases the production of an untimed process \( P \) such that \( P \subseteq Q \) will reduce to an application of \( \Theta \) 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 \( MT \) and the timed failures stability model \( TMFS \). We will see that our definition of \( P \sqsubseteq_t Q \) is equivalent to \( \Theta(Q) \subseteq P \). We obtain the result that \( \Theta(Q) \subseteq_t Q \) 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 \( MF \) and the timed failures stability model \( TMFS \). We will see that the projection mapping \( \Pi : TMFS \to MF \) is too coarse for the refinement relation between \( MF \) and \( TMFS \): 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 \( MF \) and \( TMFS \), and it is shown that for any TCSP process \( Q \) we have

\[
P \sqsubseteq_{ts} E_T^\ast[Q] \Rightarrow P \sqsubseteq_f E_T[Q]
\]
6.1 The mapping $\Theta$

The mapping $\Theta$ 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:

\[
\begin{align*}
\Theta(\text{STOP}) & \equiv \text{STOP} \\
\Theta(\bot) & \equiv \text{STOP} \\
\Theta(\text{SKIP}) & \equiv \text{SKIP} \\
\Theta(\text{WAIT } t) & \equiv \text{SKIP} \\
\Theta(a \rightarrow P) & \equiv a \rightarrow \Theta(P) \\
\Theta(a : A \rightarrow P_a) & \equiv a : A \rightarrow \Theta(P_a) \\
\Theta(P \cap Q) & \equiv \Theta(P) \cap \Theta(Q) \\
\Theta[\bigwedge_{i \in I} P_i] & \equiv \bigwedge_{i \in I} \Theta(P_i) \\
\Theta(P \parallel Q) & \equiv \Theta(P) \parallel \Theta(Q) \\
\Theta(P || Q) & \equiv \Theta(P) || \Theta(Q) \\
\Theta(P_\cdot A \parallel_b Q) & \equiv \Theta(P_\cdot A \parallel_b \Theta(Q)) \\
\Theta(P ||| Q) & \equiv \Theta(P) ||| \Theta(Q) \\
\Theta(\text{WAIT } t; P) & \equiv \Theta(P) \\
\Theta(P ; Q) & \equiv \Theta(P) ; \Theta(Q) \\
\Theta(P \setminus A) & \equiv \Theta(P) \setminus A \\
\Theta(f(P)) & \equiv f(\Theta(P)) \\
\Theta(f^{-1}(P)) & \equiv f^{-1}(\Theta(P)) \\
\Theta(X) & \equiv X \\
\Theta(\mu X \cdot P) & \equiv \mu X \cdot \Theta(P)
\end{align*}
\]

The definition of timeout yields the definition

\[
\Theta(P \triangleright Q) \equiv \Theta(Q) \cap (\Theta(P) \square \Theta(Q))
\]
Examples

$$\Theta(WAIT\ 5\ a\rightarrow SKIP)\ ;\ WAIT\ 3\ ;\ b\rightarrow WAIT\ 4)$$

$$= (SKIP\ a\rightarrow SKIP)\ ;\ b\rightarrow SKIP$$

$$\Theta(\mu X\star(WAIT\ 3\ ;\ a\rightarrow X)\frac{\not\sqsubseteq}{\not\sqsubseteq} b\rightarrow X)$$

$$= \mu X\star((b\rightarrow X)\sqcap(a\rightarrow X\sqcap b\rightarrow X))$$

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

$$\Pi(S) \equiv tstrip(traces(S))$$

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

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

$$Q_1 \sqsubseteq_t Q_2 \equiv \Pi(Q_2) \sqsubseteq Q_1$$

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

**Definition 6.2.2**

$$\sigma \sqsubseteq_t \rho \equiv \forall X : \text{var} \cdot \sigma(X) \sqsubseteq_t \rho(X)$$

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

$$P \sqsubseteq_t Q \equiv \sigma \sqsubseteq_t \rho \Rightarrow T[P] \sigma \sqsubseteq_t E_T[Q] \rho$$
Observe that the TCSP process \textit{STOP} will refine any CSP process.

We now consider which TCSP processes have \( \Theta(Q) \subseteq_t Q \).

**Definition 6.2.4** A TCSP term \( Q \) preserves \( \subseteq_t \)-refinement if

\[
\sigma \subseteq_t \rho \Rightarrow T \left[ \Theta(Q) \right] \sigma \subseteq_t E_T \left[ Q \right] \rho
\]

**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 \( S_1(X) \equiv Q_1 \subseteq_t X \) and \( S_2(X) \equiv X \subseteq_t Q_2 \) 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 \( s \in \text{traces}(P) \). Then \( \exists i \cdot s \in \text{traces}(P_i) \), so \( tstrip(s) \in Q_1 \). Hence \( \Pi(P) \subseteq Q_1 \) as required. Further, we have \( S_1(\text{STOP}) \), 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 \( s \in \text{traces}(Q_2) \). Then \( \forall i \cdot tstrip(s) \in P_i \), and so \( tstrip(s) \in P \). Hence we have \( S_2(P) \), so \( S_2 \) is continuous. It is also satisfiable, since \( S_2(\text{RUN}) \) (where \( \text{RUN} = a : \Sigma \rightarrow \text{RUN} \)). \( \square \)

**Lemma 6.2.6** If \( P \subseteq_t Q \), then \( \mu X \cdot P \subseteq_t \mu X \cdot Q \)

**Proof** Assume \( \sigma \subseteq_t \rho \), and that \( P \subseteq_t Q \). Let \( W_\delta(X) \) be the semantic function corresponding to \( \text{WAIT} \delta ; X \). Consider \( P_1 \in TM_{FS} \) such that

\[
T \left[ \mu X \cdot P \right] \sigma \subseteq_t P_1
\]

Now \( \Pi(P_1) = \Pi(W_\delta(P_1)) \), so we have

\[
\Pi(W_\delta(P_1)) \subseteq T \left[ \mu X \cdot P \right] \sigma
\]

so

\[
\sigma[T \left[ \mu X \cdot P \right] \sigma/X] \subseteq_t \rho[W_\delta(P_1)/X]
\]

and so

\[
T[P](\sigma[T \left[ \mu X \cdot P \right] \sigma/X]) \subseteq_t E_T \left[ Q \right](\rho[W_\delta(P_1)/X])
\]

so

\[
T \left[ \mu X \cdot P \right] \sigma \subseteq_t E_T \left[ Q \right](\rho[W_\delta(P_1)/X])
\]

But

\[
E_T \left[ \mu X \cdot Q \right] \rho = \text{fix}(\lambda P_1 \cdot E_T \left[ Q \right](\rho[W_\delta(P_1)/X]))
\]

91
Defining

$$S_t(P_t) \equiv (T[\mu X \cdot P] \sigma \subseteq_t P_t)$$

we have from lemma 6.2.5 that $S_t$ is continuous and satisfiable, and we have just obtained that $S_t$ is preserved by the contraction mapping above, whose fixed point is $\mathcal{E}_T[\mu X \cdot Q] \rho$, which yields that $\mu X \cdot P \subseteq_t \mu X \cdot Q$. □

**Theorem 6.2.7** All TCSP terms $Q$ preserve $\subseteq_t$-refinement.

**Proof** By structural induction. We examine each case in turn, assuming that the syntactic subcomponents preserve $\subseteq_t$-refinement, and that $\sigma \subseteq_t \rho$.

$$\Pi(\mathcal{E}_T[STOP] \rho) = \{()\} \subseteq \{()\} = T[\Theta(STOP)] \sigma$$

$$\Pi(\mathcal{E}_T[\bot] \rho) = \{()\} \subseteq \{()\} = T[\Theta(\bot)] \sigma$$

$$\Pi(\mathcal{E}_T[SKIP] \rho) = \{(), (\checkmark)\} \subseteq \{(), (\checkmark)\} = T[\Theta(SKIP)] \sigma$$

$$\Pi(\mathcal{E}_T[WAIT \ t] \rho) = \{(), (\checkmark)\} \subseteq \{(), (\checkmark)\} = T[\Theta(WAIT \ t)] \sigma$$

We next consider the one-place operators.

$$\Pi(\mathcal{E}_T[a \rightarrow Q] \rho) = \{()\} \cup \{(a) \triangleright tr \mid tr \in \Pi(\mathcal{E}_T[Q] \rho)\}$$

$$\subseteq \{()\} \cup \{(a) \triangleright tr \mid tr \in T[\Theta(Q)] \sigma\}$$

$$\subseteq T[\Theta(a \rightarrow Q)] \sigma$$

$$\Pi(\mathcal{E}_T[WAIT \ t; Q] \rho) = \{tr \mid tr \in \Pi(\mathcal{E}_T[Q] \rho)\}$$

$$\subseteq \{tr \mid tr \in T[\Theta(Q)] \sigma\}$$

$$\subseteq T[\Theta(Q)] \sigma$$

$$\Pi(\mathcal{E}_T[Q \setminus A] \rho) = \{tr \mid tr \in \Pi(\mathcal{E}_T[Q] \rho)\}$$

$$\subseteq \{tr \mid tr \in T[\Theta(Q)] \sigma\}$$

$$\subseteq T[\Theta(Q \setminus A)] \sigma$$

$$\Pi(\mathcal{E}_T[f(Q)] \rho) = \{f(tr) \mid tr \in \Pi(\mathcal{E}_T[Q] \rho)\}$$

$$\subseteq \{f(tr) \mid tr \in T[\Theta(Q)] \sigma\}$$

$$\subseteq T[\Theta(f(Q))] \sigma$$

92
\begin{align*}
\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{align*}

We now consider the two place operators.

\begin{align*}
\Pi(\mathcal{E}_T[Q_1 \cap 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 \cap Q_2)]\sigma
\end{align*}

\begin{align*}
\Pi(\mathcal{E}_T[Q_1 \cap Q_2]\rho) &= \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 \cap Q_2)]\sigma
\end{align*}

\begin{align*}
\Pi(\mathcal{E}_T[Q_1 || Q_2]\rho) &= tstrip(\text{traces}(\mathcal{E}_T[Q_1]\rho) \cap \text{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 || Q_2)]\sigma
\end{align*}

\begin{align*}
\Pi(\mathcal{E}_T[Q_1 A B Q_2]\rho) &= tstrip(\{s \mid s = s \uparrow A \cup B \land s \uparrow A \in \text{traces}(\mathcal{E}_T[Q_1]\rho) \\
&\land s \uparrow B \in \text{traces}(\mathcal{E}_T[Q_2]\rho)\}) \\
&\subseteq \{tr \mid tr = tr \uparrow A \cup B \land tr \uparrow A \in \Pi(\mathcal{E}_T[Q_1]\rho) \\
&\land tr \uparrow B \in \Pi(\mathcal{E}_T[Q_2]\rho)\} \\
&\subseteq \{tr \mid tr = tr \uparrow A \cup B \land tr \uparrow A \in \mathcal{T}[\Theta(Q_1)]\sigma \\
&\land tr \uparrow B \in \mathcal{T}[\Theta(Q_2)]\sigma\} \\
&\subseteq \mathcal{T}[\Theta(Q_1 A B Q_2)]\sigma
\end{align*}

\begin{align*}
\Pi(\mathcal{E}_T[Q_1 || B Q_2]\rho) &= tstrip(\{s \mid \exists u, v \bullet u \in \text{traces}(\mathcal{E}_T[Q_1]\rho) \\
&\land v \in \text{traces}(\mathcal{E}_T[Q_2]\rho) \land s \in \mathcal{T}[\Theta(Q_1)]\sigma \\
&\land v \in \mathcal{T}[\Theta(Q_2)]\sigma\}) \\
&\subseteq \{tr \mid \exists u, v \bullet u \in \Pi(\mathcal{E}_T[Q_1]\rho) \land v \in \Pi(\mathcal{E}_T[Q_2]\rho) \\
&\land s \in \mathcal{T}[\Theta(Q_1)]\sigma \land v \in \mathcal{T}[\Theta(Q_2)]\sigma\} \\
&\subseteq \{tr \mid \exists u, v \bullet u \in \mathcal{T}[\Theta(Q_1)]\sigma \land v \in \mathcal{T}[\Theta(Q_2)]\sigma \\
&\land s \in \mathcal{T}[\Theta(Q_1 || Q_2)]\sigma\} \\
&\subseteq \mathcal{T}[\Theta(Q_1 || Q_2)]\sigma
\end{align*}
\[
\Pi(\mathcal{E}(Q_1 ; Q_2) \rho) \subseteq \text{tstrip}\{(s \mid \sqrt{\neg} \sigma(s) \land s \in \text{traces}(\mathcal{E}(Q_1) \rho) \}
\cup \{(s^- (w + (t + \delta)) \mid \sqrt{\neg} \sigma(s) \land \\
\quad s^- ((t, \sqrt{\cdot})) \in \text{traces}(\mathcal{E}(Q_1) \rho) \land \\
\quad w \in \text{traces}(\mathcal{E}(Q_2) \rho)\}
\cup \{tr \mid \sqrt{\neg} \sigma(tr) \land tr \in \Pi(\mathcal{E}(Q_1) \rho)\}
\cup \{tr^-tr' \mid \sqrt{\neg} \sigma(tr) \land tr^- (\sqrt{\cdot}) \in \Pi(\mathcal{E}(Q_1) \rho) \land \\
\quad w \in \Pi(\mathcal{E}(Q_2) \rho)\}
\cup \{tr \mid tr \in \mathcal{T}[\Theta(Q_1)] \sigma \land \sqrt{\neg} \sigma(tr)\}
\cup \{tr^-tr' \mid tr^- (\sqrt{\cdot}) \in \mathcal{T}[\Theta(Q_1)] \sigma \land tr' \in \mathcal{T}[\Theta(Q_2)] \sigma\}
\subseteq \mathcal{T}[\Theta(Q_1) ; \Theta(Q_2)] \sigma
\]

We next consider the indexed operators.

\[
\Pi(\mathcal{E}(a : A \rightarrow Q_2) \rho) = \{()\} \cup \{(a)^-tr \mid a \in A \land tr \in \Pi(\mathcal{E}(Q_2) \rho)\}
\subseteq \{()\} \cup \{(a)^-tr \mid a \in A \land tr \in \mathcal{T}[\Theta(Q_2)] \sigma\}
\subseteq \mathcal{T}[\Theta(a : A \rightarrow Q_2)] \sigma
\]

\[
\Pi(\mathcal{E}(\prod_{a \in A} Q_2) \rho) = \bigcup_{a \in A} \Pi(\mathcal{E}(Q_2) \rho)
\subseteq \bigcup_{a \in A} \mathcal{T}[\Theta(Q_2)] \sigma
\subseteq \mathcal{T}[\Theta(\prod_{a \in A} Q_2)] \sigma
\]

The process variable case is trivial:

\[
\Pi(\mathcal{E}(X) \rho) = \Pi(\rho(X))
\subseteq \sigma(X)
\subseteq \sigma(\Theta(X))
\subseteq \mathcal{T}[\Theta(X)] \sigma
\]

The recursion case follows from lemma 6.2.6:

\[
\Theta(\mu \cdot Q) = \mu \cdot \Theta(Q)
\subseteq \mu \cdot Q
\]

\[\square\]

94
Corollary 6.2.8 Every TCSP process Q has $\Theta(Q) \subseteq Q$

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 $\cdot$ 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

$$(a \rightarrow STOP \parallel SKIP) \ ; b \rightarrow STOP$$

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 $\Theta$ extends in the obvious way to functions on TCSP processes built out of the basic process constructors. We immediately obtain the following corollary:

Corollary 6.2.9 If $P$ is a vector of CSP processes, $Q$ is a vector of TCSP processes, and $\forall i \cdot P_i \subseteq_i Q_i$, and $F$ is a function built from TCSP process constructors, then $\Theta(F)(P) \subseteq_i F(Q)$

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 $S_i \subseteq_i S_z$, then in order to establish $S_z(Q)$ it will be enough to establish $S_i(P)$ for some $P$ such that $P \subseteq_i Q$.

Definition 6.2.10 If $S_i$ is a predicate on elements of $M_T$, and $S_z$ is a predicate on elements of $TM_{FS}$, then we define:

$$S_i \subseteq_i S_z \triangleq \forall Q \cdot (\exists P \cdot S_i(P) \land P \subseteq_i Q \Rightarrow S_z(Q))$$

The following lemma provides an example.

95
Lemma 6.2.11 'weakly limited on $A'$ $\subseteq_t$ 'weakly limited on $A'$

The predicate 'weakly limited on $A$' on $M_T$ is refined by the predicate 'weakly limited on $A$' on $TM_F$.

Our proof strategy is then encapsulated by the following theorem:

Theorem 6.2.12 If $P \subseteq_t Q$, $S \subseteq_t T$ and $S(P)$, then $T(Q)$

Proof Follows immediately from the definition of $\subseteq_t$ on specifications □

The set of specifications $\{T \mid S \subseteq_t T\}$, for a given $S$, is a complete lattice under the $\subseteq$ 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

$$S \subseteq_t sr(S) \land \forall S_2 \bullet ((sr(S) \Rightarrow S_2) \Leftrightarrow S \subseteq_t S_2)$$

It follows from the definition of $\subseteq_t$ for specifications that $T$ defined by

$$T(Q) \equiv \exists P \bullet (S(P) \land P \subseteq_t Q)$$

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.

$$sr(S)(Q) \Leftrightarrow \exists P \bullet S(P) \land P \subseteq_t Q$$

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

$$sr(X \text{ sat } A(tr))(P) \Leftrightarrow P \text{ sat } A'(s, \alpha, N)$$

where $A'(s, \alpha, N) \Leftrightarrow A(tstrip(s))$
Proof We must prove that
\[ \exists P \cdot P \text{ sat } A(tr) \land P \subseteq Q \iff Q \text{ sat } A'(s, \alpha, \mathcal{N}) \]

We prove first that
\[ \exists P \cdot P \text{ sat } A(tr) \land P \subseteq Q \Rightarrow Q \text{ sat } A'(s, \alpha, \mathcal{N}) \]

We have \( P \subseteq Q \), so \( \Pi(\mathcal{E}_T[Q]) \subseteq T[P] \). Also, \( tr \in T\, [P] \Rightarrow A(tr) \). Now
\[
(s, \alpha, \mathcal{N}) \in \mathcal{E}_T[Q] \Rightarrow s \in \text{traces}(\mathcal{E}_T[Q]) \\
\Rightarrow tstrip(s) \in T[Q] \\
\Rightarrow A(tstrip(s)) \\
\Rightarrow A'(s, \alpha, \mathcal{N})
\]

which yields the required result.

We now prove
\[ Q \text{ sat } A'(s, \alpha, \mathcal{N}) \Rightarrow \exists P \cdot P \text{ sat } A(tr) \land P \subseteq Q \]

Any \( Q \in \text{TCSP} \) will have that \( U = tstrip(\text{traces}(\mathcal{E}_T[Q])) \) satisfies the trace axioms (immediate from axioms (1) and (2) of \( TM_{FS} \)). Now for a given trace \( u = (u_1, u_2, \ldots, u_n) \), define
\[ P_u = u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_n \rightarrow \text{STOP} \]

Defining
\[ P = \prod_{u \in U} P_u \]

we have \( T[P] = U \), and so \( P \text{ sat } A(tr) \) and \( P \subseteq Q \). \( \square \)

Example

The theorem allows the translation of a behavioural specification by an alteration to its syntax. For example:
\[ sr(X \text{ sat } \#(tr \uparrow out) \leq \#(tr \uparrow in)) = X \text{ sat } \#(tstrip(s) \uparrow out) \leq \#(tstrip(s) \uparrow in) \]

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
\[ wc(T)(P) \iff \forall Q \cdot (P \subseteq Q \Rightarrow T(Q)) \]
Theorem 6.2.15 If $A$ is a predicate on untimed traces, then

$$wc(X \text{ sat } A'(s, \alpha, \delta))(P) \iff P \text{ sat } A(\text{tr})$$

where $A'(s, \alpha, \delta) \iff A(\text{tstrip}(s))$

**Proof** If $P \text{ sat } A(\text{tr})$, then whenever $P \sqsubseteq_t Q$ we must have

$$s \in \text{traces}(Q) \Rightarrow \text{tstrip}(s) \in \text{traces}(P) \Rightarrow A(\text{tstrip}(s))$$

and so $Q \text{ sat } A'(s, \alpha, \delta)$.

If $\neg(P \text{ sat } A(\text{tr}))$ then $\exists \text{tr} \in \text{traces}(P) \cdot \neg A(\text{tr})$. Let $\text{tr} = \langle a_1, a_2, \ldots, a_n \rangle$. Then $Q = a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n \rightarrow \text{STOP}$ is certainly a refinement of $P$, and clearly $\neg(Q \text{ sat } A'(s, \alpha, \delta))$ as required. $\square$

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

$$S(P) \iff \langle a, b \rangle \in \text{traces}(P)$$

Then for any TCSP process $Q$ we have

$$(\Theta(Q) \cap a \rightarrow b \rightarrow \text{STOP}) \sqsubseteq_t Q$$

so $sr(S)$ holds of $Q$. Hence $sr(S) = \text{TRUE}$, and so $wc(sr(S)) = \text{TRUE}$.

### 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 \(P_2\) were a timewise refinement of \(P_1\) in this sense, that

\[
(s, [t, \infty) \times X) \in SI(P_2) \Rightarrow (tstrip(s), X) \in P_1
\]

We would not expect the converse implication, since the introduction of timing information will in general mean that some traces are no longer possible, and also that some sets may never be refusible.

If \(P_1\) is a process in \(M_F\), and \(P_2\) is a process in \(TM_{FS}\), then we write \(P_1 \subseteq_f P_2\) to say that \(P_2\) is a strong timewise refinement of \(P_1\). This is defined as follows:

**Definition 6.3.1**

\[
P_1 \subseteq_f P_2 \iff \forall (s, \mathbb{N}) \in SI(P_2), X \in \mathcal{P}(\Sigma) \cdot \\
\exists t \cdot ([t, \infty) \times X \subseteq \mathbb{N}) \Rightarrow (tstrip(s), X) \in P_1
\]

We extend the relation \(\subseteq_f\) to a relation between environments for \(M_F\) and environments for \(TM_{FS}\).

**Definition 6.3.2** If \(\sigma : \text{var} \rightarrow M_F\) and \(\rho : \text{var} \rightarrow TM_{FS}\) are environments to their respective models, then

\[
\sigma \subseteq_f \rho \iff \forall X : \text{var} \cdot \sigma(X) \subseteq_f \rho(X)
\]

Finally, we extend \(\subseteq_f\) 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

\[
P \subseteq_f Q \iff (\sigma \subseteq_f \rho) \Rightarrow \mathcal{F}[P] \sigma \subseteq \mathcal{E}_T[Q] \rho
\]

We introduce one more form of timewise refinement, which will be useful in resolving when \(P \subseteq_f Q\). 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 \(\infty\) then every set is refusible after that trace).
Definition 6.3.4 If \( P_1, P_2 \in M_F \) and \( P_2 \in TM_{FS} \), then
\[
P_1 \sqsubseteq_{ts} P_2 \Leftrightarrow \forall (s, \alpha, X) \in P_2 \bullet (tstrip(s), X) \in P_1
\]
If \( \sigma : \text{var} \rightarrow M_F \) and \( \rho : \text{var} \rightarrow TM_{FS} \) then
\[
\sigma \sqsubseteq_{ts} \rho \Leftrightarrow \forall X : \text{var} \bullet \sigma(X) \sqsubseteq_{ts} \rho(X)
\]
If \( P \) is a CSP term, and \( Q \) is a TCSP term, then
\[
P \sqsubseteq_{ts} Q \Leftrightarrow \sigma \sqsubseteq_{ts} \rho \Rightarrow \mathcal{F}[P] \sigma \sqsubseteq_{ts} \mathcal{E}_t^*[Q] \rho
\]

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
1. \( Q \) preserves \( \sqsubseteq_f \)-refinement if \( \Theta(Q) \sqsubseteq_f Q \)
2. \( Q \) 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 \( Q_1, Q_2 \in TM_{FS} \):
\[
S_1(X) \equiv X \sqsubseteq_f Q_1
\]
\[
S_2(X) \equiv X \sqsubseteq_{ts} Q_2
\]

Proof Let \( \{P_i\} \) be a convergent sequence of processes in \( M_F \) whose limit is \( P \). Assume that \( S_i(P_i) \) for each \( i \). Let \( s \) and \( X \) be such that
\[
\exists \pi, t \bullet (s, \pi) \in SI(Q_i) \land ([t, \infty) \times X \subseteq \pi)
\]

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 \( P \sqsubseteq_f Q_i \), and so \( S_i \) is continuous. Further the process \( \text{CHAOS} \), defined by
\[
\text{CHAOS} \equiv \mu X \bullet (a : \Sigma \rightarrow X) \cap \bot
\]
satisfies \( S_1 \), since it exhibits every possible \( \text{(trace, refusal)} \) pair.

The proof for the continuity and satisfiability of \( S_2 \) is entirely similar. \( \Box \)
Conversely, neither \( S_1(X) \equiv P \sqsubseteq_t X \) nor \( S_2(X) \equiv P \sqsubseteq_{ts} X \) are continuous for general \( M_F \) processes \( P \), although they may be continuous for some \( P \) (e.g. CHAOS). Consider

\[
P_n \equiv \text{WAIT } n ; a \to \text{STOP}
\]

\[
a \to \text{STOP } \sqsubseteq_f P_n
\]

\[
a \to \text{STOP } \sqsubseteq_{ts} \lim_{n \to \infty} P_n = \bot
\]

Both \( a \to \text{STOP } \sqsubseteq_f P_n \) and \( a \to \text{STOP } \sqsubseteq_{ts} X \) hold of each \( P_n \), but neither holds of \( \lim_{n \to \infty} P_n \).

However, the predicate \( S_2(X) \equiv P \sqsubseteq_{ts} X \) 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 \( P \in M_F \), \( t > 0 \):

\[
S_1(X) \equiv P \sqsubseteq_f X \wedge X \text{ is } t\text{-prompt}
\]

\[
S_2(X) \equiv P \sqsubseteq_{ts} X \wedge X \text{ is } t\text{-stable}
\]

**Proof** We prove \( S_1 \) is continuous. The proof for \( S_2 \) is entirely similar. Assume \( \{Q_i\} \to Q \), with \( S_1(Q_i) \) for each \( i \). Now consider \( (s, n) \in ST \mathcal{T}[Q] \) with \( (t', \infty) \times X \subseteq \mathcal{N} \). Then \( (t', t' + t) \times X \subseteq \mathcal{N} \), so

\[
(s, [t', t' + t] \times X) \in \text{fail}(\mathcal{E}_T[Q])
\]

and so for some \( n \) we have

\[
(s, [t', t' + t] \times X) \in \text{fail}(\mathcal{E}_T[Q_n])
\]

Now \( Q_n \) is \( t \)-prompt, so we deduce

\[
(s, [t', \infty) \times X) \in \text{fail}(\mathcal{E}_T[Q_n])
\]

and since \( P \sqsubseteq_f Q_n \) we have \( t \text{strip}(s), X) \in \mathcal{F}[P] \), as required. \( \Box \)

**Lemma 6.3.8** For any CSP term \( P \), TCSP term \( Q \) and \( t \geq 0 \) we have

\[
P \sqsubseteq_{ts} Q \Rightarrow P \sqsubseteq_{ts} \text{WAIT } t ; Q
\]

\[
P \sqsubseteq_f Q \Rightarrow P \sqsubseteq_f \text{WAIT } t ; Q
\]
Proof Assume $P \subseteq_{ts} Q$, and that $\sigma \subseteq_{ts} \rho$. Then

$$(s, \alpha, X) \in \mathcal{E}_T[\text{WAIT } t ; Q] \rho \Rightarrow (s - t, \alpha - t, X) \in \mathcal{E}_T[Q] \rho$$

$$\Rightarrow (\text{thstrip}(s - t), X) \in \mathcal{F}[P] \sigma$$

$$\Rightarrow (\text{thstrip}(s), X) \in \mathcal{F}[P] \sigma$$

so $P \subseteq_{ts} \text{WAIT } t ; Q$. □

Assume $P \subseteq_{f} Q$, and that $\sigma \subseteq_{f} \rho$. Then

$$(s, \infty) \in S_{\mathcal{I}_T}[\text{WAIT } t ; Q] \rho \land [t', \infty) \times X \subseteq \mathbb{N}$$

$$\Rightarrow (s - t, \infty) \in S_{\mathcal{I}_T}[Q] \rho \land [t' - t, \infty) \times X \subseteq \mathbb{N} - t$$

$$\Rightarrow (\text{thstrip}(s - t), X) \in \mathcal{F}[P] \sigma$$

$$\Rightarrow (\text{thstrip}(s), X) \in \mathcal{F}[P] \sigma$$

so $P \subseteq_{f} \text{WAIT } t ; Q$. □

**Lemma 6.3.9** Let $P$ be a CSP term that represents a contraction mapping, and let $Q$ be a TCSP term.

- If $P \subseteq_{f} Q$ then $\mu X \cdot P \subseteq_{f} \mu X \cdot Q$
- If $P \subseteq_{ts} Q$ then $\mu X \cdot P \subseteq_{ts} \mu X \cdot Q$

**Proof** $\mathcal{F}[\mu X \cdot P]$ is the unique fixed point of the contraction mapping $C \equiv \lambda Y \cdot \mathcal{F}[P](\sigma[Y/X])$ on $\mathcal{M}_F$. Now let $\sigma \subseteq_{f} \rho$. Assume $P \subseteq_{f} \mathcal{E}_T[\mu X \cdot Q] \rho$. Then from lemma 6.3.8 we have $P \subseteq_{f} \mathcal{E}_T[\text{WAIT } \delta ; \mu X \cdot Q] \rho$, so

$$\sigma[P_1/X] \subseteq_{f} \rho[\mathcal{E}_T[\text{WAIT } \delta ; \mu X \cdot Q] \rho/X]$$

so we have

$$\mathcal{F}[P](\sigma[P_1/X]) \subseteq_{f} \mathcal{E}_T[Q](\rho[\mathcal{E}_T[\text{WAIT } \delta ; \mu X \cdot Q] \rho/X])$$

and so

$$C(P_1) \subseteq_{f} \mathcal{E}_T[\mu X \cdot Q] \rho$$

Hence $C$ preserves $S(X) \equiv X \subseteq_{f} \mathcal{E}_T[\mu X \cdot Q] \rho$, 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 $S(\text{fix } C)$, as required.

The proof for $\subseteq_{ts}$ is entirely similar. □

We will prove that the refinement relation $\subseteq_{f}$ holds whenever $\subseteq_{ts}$ holds. However, the converse is not the case. Consider

$$a \rightarrow \text{STOP} \ \subseteq_{f} \ \mu X \cdot ((a \rightarrow \text{STOP}) \downarrow X)$$

102
But $E_T[\mu X \cdot ((a \rightarrow \text{STOP}) \not\in X)]$ has stability value $\infty$ associated with the empty trace. Hence axiom 13 for $TM_{FS}$ tells us that $((\emptyset, \{a\})$ is a possible failure of the process. But it is not a possible failure of $\mathcal{F}[a \rightarrow \text{STOP}]$, so

$$a \rightarrow \text{STOP} \not\subseteq_{ts} \mu X \cdot ((a \rightarrow \text{STOP}) \not\in X)$$

In order to prove that $P \subseteq_{ts} Q \Rightarrow P \subseteq f Q$, we must first obtain some subsidiary results.

**Definition 6.3.10** A trace $s$ in $T\Sigma_{\leq}$ is reflected in $\mathcal{N}$ if

$$\langle (t, \hat{a}) \rangle \text{ in } s \wedge t > 0 \Rightarrow \exists t' < t \cdot [t', t) \times \{a\} \subseteq \mathcal{N}$$

When timed refusals are present, we deduce that an event happens at the instant it becomes available from the observation on $\mathcal{N}$ that it was refused for some interval up to that time. Hence a trace $s$ is reflected in $\mathcal{N}$ if there is evidence in $\mathcal{N}$ that the hatted events in $s$ occurred at the instant they became available. We write $R(s, \mathcal{N})$ to abbreviate ‘$s$ is reflected in $\mathcal{N}$’. The following simple consequences of this definition will be used in the next theorem.

**Corollary 6.3.11** The following hold of the relation $R$:

- $R(s, \mathcal{N}) \Rightarrow R(s \setminus t, \mathcal{N} \setminus t)$
- If $R(s, \mathcal{N})$ and $s \in T\text{merge}(u, v)$ then $R(u, \mathcal{N})$ and $R(v, \mathcal{N})$.
- If $R(s, \mathcal{N})$ and $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$ then $\exists s_1, s_2 \cdot s = s_1 \lor s_2 \land R(s_1, \mathcal{N}_1) \land R(s_2, \mathcal{N}_2)$

We will now define a relation between semantic sets $P_1$ in $TM_{FS}$ and $P_2$ in $TM_{FS}$. This relation captures the situation 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 $P_2$; the 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 $(\tilde{s}, \alpha, \mathcal{N}) \in P_1$, 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 $\mathcal{N}$, the refusal information contained in $\mathcal{N}$ 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 $(\tilde{s}, \mathcal{N})$ in $TM_{FS}$.

When this relation holds between $P_1$ and $P_2$, we say $P_2$ follows $P_1$:

---

\(^1\) $P$ is more deterministic than $Q$ if $Q = P \cap Q$ (‘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 $s \in T\Sigma^*$ we have

$$(\tilde{s}, \alpha, \overline{N}) \in P_1 \land R(s, \overline{N}) \Rightarrow \exists \beta \geq \alpha \bullet (s, \beta, \sigma(\overline{N} \uparrow \alpha)) \in P_2$$

We abbreviate $S_2$ follows $S_1$ to $F(S_2, S_1)$. If $\rho_1 : \text{var} \rightarrow TM_{FS}$ and $\rho_2 : \text{var} \rightarrow TM_{FS}^*$, then $\rho_2$ follows $\rho_1$ 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

$$F(\rho_2, \rho_1) \Rightarrow F(E_T^*[Q], \rho_2, E_T^*[P], \rho_1)$$

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 $\Pi(E_T^*[P])$ follows $E_T^*[P]$, and is more deterministic than $E_T^*[P]$.

The consequence of the next theorem that is presently most useful is that it will allow us to conclude that $\subseteq_{\alpha}$-refinement is stronger than $\subseteq_{\beta}$-refinement, which will aid us in establishing the presence of the $\subseteq_{\beta}$-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 $R(s, \overline{N})$ to mean that $s$ is reflected in $\overline{N}$.

Case $\sqcap_i P_i$

Consider $(\tilde{s}, \alpha, \overline{N}) \in E_T^[[\sqcap_i P_i]]$. Then define

$$S = \{ \gamma \mid \exists i : (\tilde{s}, \gamma, \overline{N}) \in E_T^[[P_i]] \rho_1 \}$$

$$T = \{ \beta \mid \exists i : (s, \beta, \sigma(\overline{N} \uparrow \alpha)) \in E_T^[[P_i]] \rho_2 \}$$

104
Then $\alpha = \sup(S)$. Now if $T \neq \emptyset$ then $(s, \sup(T), \sigma(\mathcal{N} \upharpoonright \alpha)) \in \mathcal{E}_T[\prod_i P_i]_{\rho_2}$. So to prove the required result, we need only show that $\gamma \in S \Rightarrow \exists \beta \geq \gamma \cdot \beta \in T$

\[
\begin{align*}
\gamma \in S \\
\Rightarrow \exists i \cdot (\tilde{s}, \gamma, \mathcal{N}) \in \mathcal{E}_T[P_i]_{\rho_1} \\
\Rightarrow \exists i, \beta \geq \gamma \cdot (s, \beta, \sigma(\mathcal{N} \upharpoonright \gamma)) \in \mathcal{E}_T[P_i]_{\rho_2} \\
\Rightarrow \exists i, \beta \geq \gamma \cdot (s, \beta, \sigma(\mathcal{N} \upharpoonright \alpha)) \in \mathcal{E}_T[P_i]_{\rho_2} \\
\Rightarrow \exists \beta \geq \gamma \cdot \beta \in T
\end{align*}
\]
as required.

**Case** $P \parallel Q$

Now consider $(\tilde{s}, \alpha, \mathcal{N}) \in \mathcal{E}_T[P \parallel Q]_{\rho_i}$, where $R(s, \mathcal{N})$. Define:

\[
S = \{ \max\{\alpha_1, \alpha_2\} \mid \exists \mathcal{N}_1, \mathcal{N}_2 \cdot \mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2 \wedge \\
(\tilde{s}, \alpha_1, \mathcal{N}_1) \in \mathcal{E}_T[P]_{\rho_1} \wedge (\tilde{s}, \alpha_2, \mathcal{N}_2) \in \mathcal{E}_T[Q]_{\rho_2}\}
\]

\[
T = \{ \max\{\beta_1, \beta_2\} \mid \exists s_1, s_2, X_1, X_2 \cdot s = s_1 \lor s_2 \land X_1 \cup X_2 = \sigma(\mathcal{N} \upharpoonright \alpha)\}
\]

Then $\alpha = \sup(S)$. Now given $\mathcal{N}_1$ and $\mathcal{N}_2$ such that $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$ we have from corollary 6.3.11 that there are $s_1$ and $s_2$ such that $s = s_1 \lor s_2$ and $R(s_1, \mathcal{N}_1)$ and $R(s_2, \mathcal{N}_2)$, so.

\[
S = \{ \max\{\alpha_1, \alpha_2\} \mid \exists \mathcal{N}_1, \mathcal{N}_2, s_1, s_2 \cdot \mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2 \land s = s_1 \lor s_2 \land \\
(\tilde{s}_1, \alpha_1, \mathcal{N}_1) \in \mathcal{E}_T[P]_{\rho_1} \wedge (\tilde{s}_2, \alpha_2, \mathcal{N}_2) \in \mathcal{E}_T[Q]_{\rho_2} \\
R(s_1, \mathcal{N}_1) \wedge R(s_2, \mathcal{N}_2)\}
\]

Consider $\gamma \in S$. Then $\gamma = \max\{\alpha_1, \alpha_2\}$, where

\[
(\tilde{s}_1, \alpha_1, \mathcal{N}_1) \in \mathcal{E}_T[P]_{\rho_1} \wedge R(s_1, \mathcal{N}) \wedge (\tilde{s}_2, \alpha_2, \mathcal{N}_2) \in \mathcal{E}_T[Q]_{\rho_1} \wedge R(s_2, \mathcal{N})
\]

Hence

\[
\exists \beta_1 \geq \alpha_1, \beta_2 \geq \alpha_2 \cdot (s_1, \beta_1, \sigma(\mathcal{N}_1 \upharpoonright \alpha_1)) \in \mathcal{E}_T[P]_{\rho_2} \wedge (s_2, \beta_2, \sigma(\mathcal{N}_2 \upharpoonright \alpha_2)) \in \mathcal{E}_T[Q]_{\rho_2}
\]

But $\alpha_1 \leq \alpha$ and $\alpha_2 \leq \alpha$, so

\[
\sigma(\mathcal{N}_1 \upharpoonright \alpha) \subseteq \sigma(\mathcal{N}_1 \upharpoonright \alpha_1) \land \sigma(\mathcal{N}_2 \upharpoonright \alpha) \subseteq \sigma(\mathcal{N}_2 \upharpoonright \alpha_2)
\]

so

\[
(s_1, \beta_1, \sigma(\mathcal{N}_1 \upharpoonright \alpha)) \in \mathcal{E}_T[P]_{\rho_2} \wedge (s_2, \beta_2, \sigma(\mathcal{N}_2 \upharpoonright \alpha)) \in \mathcal{E}_T[Q]_{\rho_2}
\]

Therefore, $\max\{\beta_1, \beta_2\} \in T$, so $\exists \gamma' \geq \gamma \cdot \gamma' \in T$. Hence

\[
(s, \sup(T), \sigma(\mathcal{N} \upharpoonright \alpha)) \in \mathcal{E}_T[P \parallel Q]_{\rho_2}
\]
and \( \text{sup}(T) \geq \text{sup}(S) \). \( \square \)

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 \( P \parallel Q \)

Now consider \((\tilde{s}, \alpha, \pi) \in \mathcal{E}_T[P \parallel Q]_{\rho_1}\), where \(R(s, \pi)\). Define:

\[
\begin{align*}
S &= \{\alpha \mid \forall \ i \cdot (\tilde{s}, \alpha, \pi \cup (I \times \{\checkmark}\}) \in \mathcal{E}_T[P]_{\rho_1} \land \checkmark \notin \sigma(\tilde{s})
\cup \{\alpha + t + \delta \mid \checkmark \notin \sigma(\tilde{s} \rightarrow t) \land (\tilde{s} \rightarrow (t + \delta), \alpha, \pi \cup (t + \delta)) \in \mathcal{E}_T[Q]_{\rho_1}
\land ((\tilde{s} \rightarrow t)^{-}(t, \checkmark)), \pi \rightarrow t \cup [0, t) \times \{\checkmark\} \} \in \text{fail}(\mathcal{E}_T[P]_{\rho_1})
\end{align*}
\]

\[
T = \{\alpha \mid (s, \alpha, X \cup \{\checkmark\}) \in \mathcal{E}_T[P]_{\rho_2} \land \checkmark \notin \sigma(s)
\cup \{\alpha + (t + \delta) \mid (s \rightarrow t)^{-}(t, \checkmark)) \in \text{traces}(\mathcal{E}_T[P]_{\rho_2}) \land \checkmark \notin \sigma(s \rightarrow t)
(s \rightarrow (t + \delta), \alpha, \sigma(\pi \cup \alpha)) \in \mathcal{E}_T[Q]_{\rho_2}
\}
\]

Then \( \alpha = \text{sup}(S) \), and if \( T \) is not empty then \((s, \text{sup}(T), \sigma(\pi \cup \alpha)) \in \mathcal{E}_T[Q]_{\rho_2}\). It is sufficient to prove that \( \gamma \in S \Rightarrow \exists \gamma' \geq \gamma \cdot \gamma' \in T \). So consider \( \gamma \in S \). Then either:

\[
\begin{align*}
(s, \gamma, \pi \cup [0, \gamma + 1) \times \{\checkmark\}) \in \mathcal{E}_T[P]_{\rho_1} \land \checkmark \notin \sigma(\tilde{s}) \land R(s, \pi \cup [0, \gamma + 1) \times \{\checkmark\})
\Rightarrow \exists \gamma' \geq \gamma \cdot (s, \gamma', \sigma(\pi \cup \alpha) \cup \{\checkmark\}) \in \mathcal{E}_T[P]_{\rho_2}
\Rightarrow \exists \gamma' \geq \gamma \cdot (s, \gamma', \sigma(\pi \cup \alpha) \cup \{\checkmark\}) \in \mathcal{E}_T[Q]_{\rho_2}
\Rightarrow \exists \gamma' \geq \gamma \cdot \gamma' \in T
\end{align*}
\]
or:

\[
\begin{align*}
(s \rightarrow (t + \delta)) \in \text{traces}(\mathcal{E}_T[P]_{\rho_2})
\land \checkmark \notin \sigma(s \rightarrow (t + \delta), \beta') \land (\pi \cup (t + \delta)) \in \mathcal{E}_T[Q]_{\rho_1}
\land R((s \rightarrow t)^{-}(t, \checkmark)), \pi \rightarrow t \cup [0, t) \times \{\checkmark\})
\Rightarrow (s \rightarrow (t, \checkmark)) \in \text{traces}(\mathcal{E}_T[P]_{\rho_2})
\land \exists \beta' \geq \beta \cdot (s \rightarrow (t + \delta), \beta', \sigma(\pi \cup (t + \delta) \cup \beta)) \in \mathcal{E}_T[Q]_{\rho_2}
\Rightarrow (s \rightarrow (t, \checkmark)) \in \text{traces}(\mathcal{E}_T[P]_{\rho_2})
\land (s \rightarrow (t + \delta), \beta', \sigma(\pi \cup \alpha)) \in \mathcal{E}_T[Q]_{\rho_2}
\Rightarrow \beta' + (t + \delta) \in T \land \beta' + (t + \delta) \geq \gamma
\Rightarrow \exists \gamma' \geq \gamma \cdot \gamma' \in T
\end{align*}
\]

\( \square \)

Case \( P \setminus A \)

To prove this case, we first require a technical sublemma:
Sublemma 6.3.15  If \((s, \alpha, \mathcal{R}) \in E_T[P \setminus A] \rho \land \alpha < \infty\) then

\[\forall \nu > \alpha \bullet \alpha = \sup \{\beta \mid \exists w \bullet s = w \setminus A \land (w, \beta, \mathcal{R} \cup \{0, \nu\} \times A) \in E_T[P] \rho\}\]

Proof  Consider \((s, \alpha, \mathcal{R}) \in E_T[P \setminus A] \rho\), and \(\nu > \alpha\). Define

\[S = \{\beta \mid \exists w \bullet s = w \setminus A \land (w, \beta, \mathcal{R} \cup \{0, \nu\} \times A) \in E_T[P] \rho\}\]

\[S_{\nu} = \{\beta \mid \exists w \bullet s = w \setminus A \land (w, \beta, \mathcal{R} \cup \{0, \nu\} \times A) \in E_T[P] \rho\}\]

Then from the definition of hiding, \(\alpha = \sup(S)\). Now lemma A. 1.2 yields that \(\gamma \in S \Rightarrow \exists \gamma' > \gamma \bullet \gamma' \in S_{\nu}\), so we obtain \(\sup(S_{\nu}) \geq \sup(S)\)

We wish to prove \(\sup(S_{\nu}) = \sup(S)\), so we assume \(\sup(S_{\nu}) > \sup(S)\) for a contradiction:

\[\sup(S_{\nu}) > \sup(S)\]
\[\Rightarrow \exists \beta \in S_{\nu} \bullet \beta > \alpha\]
\[\Rightarrow \exists w \bullet s = w \setminus A \land (w, \beta, \mathcal{R} \cup \{0, \nu\} \times A) \in E_T[P] \rho\]
\[\Rightarrow \exists \beta' > \beta, w \bullet s = w \setminus A \land (w, \beta', \mathcal{R} \cup \{0, \min\{\nu, \beta\}\} \times A) \in E_T[P]\]
\[\Rightarrow \min\{\nu, \beta\} \in S\]
\[\Rightarrow \sup(S) \geq \min\{\nu, \beta\} > \alpha\]

which yields a contradiction. \(\square\)

We can now handle the hiding case:

Let \((\tilde{s}, \alpha, \mathcal{R}) \in E_T[P \setminus A] \rho_1 \land R(s, \mathcal{R})\). Define:

\[S = \{\beta \mid \exists w \bullet \tilde{s} = w \setminus A \land \exists \alpha' \geq \beta \geq \text{end}(w) \bullet (w, \alpha', \mathcal{R} \cup \{0, \text{max}\{\text{end}(\mathcal{R}, \beta)\} \times A) \in E_T[P] \rho_1\}\]

\[S_{\gamma} = \{\beta \mid \exists w \bullet s = w \setminus A \land w \text{ is A-active} \land (\tilde{w}, \beta, \mathcal{R} \cup \{0, \gamma\} \times A) \in E_T[P] \rho_1\}\]

\[T = \{\beta \mid \exists (w, \beta) \in \text{stab}(E_T[P] \rho_2) \bullet w \setminus A = s \land w \text{ is A-active}\}\]

Case \(\alpha = \infty\)

The result follows immediately, since \(\mathcal{R} \uparrow \infty = \emptyset\)

Case \(\alpha < \infty\)

For any \(\gamma\), if \(\beta \in S_{\gamma}\) then \(\exists \beta' \in T \bullet \beta' \geq \beta\). Hence if \(\forall \gamma \bullet \sup(S_{\gamma}) = \infty\) then \(\sup(T) = \infty\), and \((s, \infty, \sigma(\mathcal{R} \uparrow \alpha)) \in E_T[P] \rho_2\).

If \(\exists \gamma \bullet \sup(S_{\gamma}) = \alpha' < \infty\), then set \(\nu = \text{max}\{\gamma, \alpha', \alpha\} + 1\). Then from sublemma 6.3.15 we obtain \(\sup(S_{\nu}) = \alpha\), and so \(\sup(T) \geq \alpha\). Also,

\[\beta \in S_{\nu}\]
\[\Rightarrow \exists w \bullet s = w \setminus A \land w \text{ is A-active} \land (\tilde{w}, \beta, \mathcal{R} \cup \{0, \nu\} \times A) \in E_T[P] \rho_1\]
\[\Rightarrow \exists w \bullet s = w \setminus A \land w \text{ is A-active} \land (w, \sigma(\mathcal{R} \uparrow \beta) \cup A) \in \text{fail}(E_T[P] \rho_2)\]
\[\Rightarrow (s, \sigma(\mathcal{R} \uparrow \alpha)) \in \text{fail}(E_T[P \setminus A] \rho_2)\] \(\square\)
Case $\mu X \cdot P$

Define $G(Y) = P[(\text{WAIT } \delta; Y)/X]$. We will first prove that $G^n(\text{STOP})$ follows $G^n(\text{STOP})$ for all $n$.

Base Case: $\text{STOP}$ follows $\text{STOP}$ is immediate.

Case $n + 1$:

Assume $F(G^n(\text{STOP}), G^n(\text{STOP}))$ and $F(\rho_2, \rho_1)$.

$$\mathcal{E}_T^{*}[G^{n+1}(\text{STOP})]_{\rho_2} = \mathcal{E}_T^{*}[G(G^n(\text{STOP}))]_{\rho_2} = \mathcal{E}_T^{*}[P]\left(\rho_2[\mathcal{E}_T^{*}[\text{WAIT } \delta; G^n(\text{STOP})]_{\rho_2}/X]\right)$$

which follows

$$\mathcal{E}_T^{*}[P]\left(\rho_1[\mathcal{E}_T^{*}[\text{WAIT } \delta; G^n(\text{STOP})]_{\rho_1}/X]\right) = \mathcal{E}_T^{*}[G^{n+1}(\text{STOP})]_{\rho_1}$$

Hence we have that $\forall n \cdot F(G^n(\text{STOP}), G^n(\text{STOP}))$.

Now $\{G^n(\text{STOP})\}$ has limit $\mu X \cdot P$ in both $T_{MFS}$ and $T_{MFS}$. Consider $(\xi, \alpha, \eta) \in \mathcal{E}_T[\mu X \cdot P]$.

Case $\alpha < \infty$

Then $\exists N \cdot N\delta > \max\{\text{end}(s, \xi), \alpha\}$. So then

$$\forall n > N \cdot (\xi, \alpha, \eta) \in \mathcal{E}_T[G^n(\text{STOP})]$$

But $\forall n \cdot F(G^n(\text{STOP}), G^n(\text{STOP}))$, so

$$\forall n > N, \exists \beta \geq \alpha \cdot (s, \beta, \sigma(\xi \uparrow \alpha)) \in \mathcal{E}_T^{*}[F^n(\text{STOP})]$$

Hence we obtain

$$\exists \beta \geq \alpha \cdot (s, \beta, \sigma(\xi \uparrow \alpha)) \in \mathcal{E}_T^{*}[\mu X \cdot F(X)]$$

Case $\alpha = \infty$

$$\Rightarrow \sup\{\alpha_n | (\xi, \alpha_n) \in \text{stab}(\mathcal{E}_T^{*}[F^n(\text{STOP})])\} = \infty$$

$$\Rightarrow \sup\{\beta_n | (s, \beta_n) \in \text{stab}(\mathcal{E}_T^{*}[F^n(\text{STOP})])\} = \infty$$

$$\Rightarrow (s, \infty) \in \text{stab}(\mathcal{E}_T^{*}[\mu X \cdot F(X)])$$

$$\Rightarrow (s, \infty, \sigma(\xi \uparrow \alpha)) \in \mathcal{E}_T^{*}[\mu X \cdot F(X)]$$
Corollary 6.3.16 \((s, \infty, N) \in E_T[P] \Rightarrow (s, \infty) \in stab(E_T[P])\)

We are now in a position to prove

Theorem 6.3.17 For any CSP process \(P\) and TCSP process \(Q\) we have

\[ P \sqsubseteq_u Q \Rightarrow P \sqsubseteq_f Q \]

Proof Assume \(P \sqsubseteq_u Q\). Consider \((s, N) \in S_I[T][Q]\), with \([t, \infty) \times X \subseteq N\) for some \(t\). Then \((s, \alpha, 0) \in E_T[Q]\) for some \(\alpha\).

Case \(\alpha = \infty\)

Then

\[ (s, \infty) \in stab(E_T[Q]) \]

\[ \Rightarrow (s, X) \in fail(E_T[Q]) \]

\[ \Rightarrow (tstrip(s), X) \in F[P] \]

Case \(\alpha < \infty\)

Then from axiom 10 for \(TM_{F2}\) we have

\[ \exists \beta \leq \alpha \cdot (s, \beta, [t, \max\{t, \alpha\} + 1) \times X) \in E_T[Q] \]

\[ \Rightarrow (s, X) \in fail(E_T[Q]) \text{ since } F(Q, Q) \]

\[ \Rightarrow (tstrip(s), X) \in F[P] \]

\(\square\)

Theorem 6.3.18 All the process constructors except hiding, infinite nondeterminism, and infinite-to-one alphabet renaming preserve \(\sqsubseteq_u\) refinement.

Proof By a straightforward case analysis. \(\square\)

We identify sufficient conditions on timed processes for the preservation of \(\sqsubseteq_u\)-refinement:

Definition 6.3.19 A set of processes \(\{Q_i\}\) has bounded stability if

\[ \sup\{\alpha \mid \exists \; i \cdot (s, \alpha) \in stab(E_T[Q_i])\} = \infty \Rightarrow \exists \; i \cdot (s, \infty) \in stab(E_T[Q_i]) \]

Lemma 6.3.20 If \(\{Q_i\}\) has bounded stability and \(\{P_i\}\) is such that \(\forall \; i \cdot P_i \sqsubseteq_u Q_i\) then \(\bigcap_{i} P_i \sqsubseteq_u \bigcap_{i} Q_i\)

109
Proof Consider \((s, \alpha, X) \in \mathcal{E}_T^* \{\bigcap_i Q_i\}\). We have \(\alpha = \sup(S)\), where

\[ S = \{\beta \mid \exists i \cdot (s, \beta, X) \in \mathcal{E}_T^* \{Q_i\}\} \]

Case \(\alpha = \infty\)

Then by bounded stability we have

\[ \exists i \cdot (s, \infty, X) \in \text{stab(} \mathcal{E}_T^* \{Q_i\}\text{)} \]

so

\[ (s, \infty, X) \in \mathcal{E}_T^* \{Q_i\}\]

Hence \((\text{strip}(s), X) \in \mathcal{F}[P_i]\), and so \((\text{strip}(s), X) \in \mathcal{F}[\bigcap_i P_i]\).

Case \(\alpha < \infty\)

Then

\[ \exists i \cdot (s, \beta, X) \in \mathcal{E}_T^* \{Q_i\}\]

\[ \Rightarrow (\text{strip}(s), X) \in \mathcal{F}[P_i]\]

\[ \Rightarrow (\text{strip}(s), X) \in \mathcal{F}[\bigcap_i P_i]\]

\[ \Box \]

Definition 6.3.21 A process \(P\) is bounded under \(A\) if

\[ \forall s \in \text{traces(} \mathcal{E}_T^* \{P\}\text{)} \cdot \sup\{\beta \mid \exists (w, \beta) \in \text{stab(} \mathcal{E}_T^* \{P\}\). \cdot w \setminus A = s\} = \infty \]

\[ \Leftrightarrow \exists (w, \infty) \in \text{stab(} \mathcal{E}_T^* \{P\}\). \cdot w \setminus A = s \]

Lemma 6.3.22 If \(Q\) is bounded under \(A\) and \(P \subseteq_s Q\) then \(P \setminus A \subseteq_s Q \setminus A\)

Proof Consider \((s, \alpha, X) \in \mathcal{E}_T^* \{Q \setminus A\}\).

Case \(\alpha = \infty\)

Then by bounded stability we have

\[ \exists (w, \infty) \in \text{stab(} \mathcal{E}_T^* \{Q\}\). \cdot w\text{ is }A\text{-active} \wedge w \setminus A = s \]

\[ \Rightarrow (w, \infty, X \cup A) \in \mathcal{E}_T^* \{Q\}\]

\[ \Rightarrow (\text{strip}(w), X \cup A) \in \mathcal{F}[P]\]

\[ \Rightarrow (\text{strip}(w) \setminus A, X) \in \mathcal{F}[P \setminus A]\]

\[ \Rightarrow (\text{strip}(s), X) \in \mathcal{F}[P \setminus A]\]

Case \(\alpha < \infty\)

\[ \Rightarrow \exists w \cdot (w, X \cup A) \in \text{fail(} \mathcal{E}_T^* \{Q\}\). \wedge w \setminus A = s \]

\[ \Rightarrow (\text{strip}(w), X \cup A) \in \mathcal{F}[P]\]

\[ \Rightarrow (\text{strip}(w) \setminus A, X) \in \mathcal{F}[P \setminus A]\]

\[ \Rightarrow (\text{strip}(s), X) \in \mathcal{F}[P \setminus A]\]

\[ \Box \]
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 $\bigcap_{i \in I} P_i$ 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 $\Theta(P) \subseteq_{t_\uparrow} P$

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 $\Theta$.

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 $\bigcap_{i \in I} P_i$ 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 $\Theta(P) \subseteq_{t} P$

We have obtained, via $\subseteq_{t_\uparrow}$-refinement, a sufficient condition for TCSP terms to preserve $\subseteq_t$-refinement. We now examine which TCSP operators preserve $\subseteq_t$-refinement. We shall find that it is not preserved by parallel composition, despite the fact that 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 $\Theta(Q) \subseteq_t Q$.

Proof Immediate, since they each satisfy $\Theta(Q) \subseteq_{t_\uparrow} Q$ □

Lemma 6.3.26 The following TCSP process constructors preserve $\subseteq_t$-refinement:

- $a \rightarrow P$, $a : A \rightarrow P(a)$.
- $P \sqcap Q$, $\bigcap_{i} P_i$
- $P \sqcup Q$, $P \parallel Q$, $P ; Q$, $f^{-1}(P)$

Proof Straightforward □
However, the '||' and 'X_i\parallel X_j' operators do not in general preserve \( C_i / C_j \)-refinement. Consider

\[
\begin{align*}
P & \equiv a \rightarrow \text{STOP} \\
Q_1 & \equiv \mu X \cdot (a \rightarrow \text{STOP} \downarrow \text{WAIT} \ 3 \ ; \ X) \\
Q_2 & \equiv \text{WAIT} \ 2 \ ; \ Q_i
\end{align*}
\]

Then \( P \subseteq C_i \ Q_1 \) and \( P \subseteq C_i \ Q_2 \), but

\[
P \parallel P \not\subseteq C_i \ Q_1 \parallel Q_2
\]

since \( P \parallel P \) is unable to refuse \( a \) after the empty trace, but \( Q_1 \parallel Q_2 \) can refuse it. Formally:

\[
\begin{align*}
(\emptyset, \{a\}) & \notin \mathcal{F}[P \parallel P] \\
(\emptyset, (0, \infty) \times \{a\}) & \in \mathcal{T}_{C_i}[Q_1 \parallel Q_2]
\end{align*}
\]

We therefore seek conditions where \( C_i / C_j \)-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, \( P_1 \subseteq C_i \ Q_1 \), \( P_2 \subseteq C_i \ Q_2 \), both \( Q_1 \) and \( Q_2 \) are prompt on \( X_i \cap X_2 \), \( Q_1 \) is non-retracting on \( Y_1 \) and \( Q_2 \) is non-retracting on \( Y_2 \), and \( X_i \cap X_2 \subseteq Y_i \cup Y_2 \) then

\[
P_1 \parallel X_1 \parallel X_2 \ P_2 \subseteq C_i \ Q_1 \parallel X_1 \parallel X_2 \ Q_2
\]

**Proof** Let \( Q_1 \) and \( Q_2 \) be \( t' \)-prompt. Consider \( (s, R) \in \mathcal{T}_{C_i}[Q_1 \parallel X_1 \parallel X_2 \ Q_2] \), and \((t, \infty) \times A \subseteq R\). Then

\[
\exists s_1, s_2, R_1, R_2 \cdot s \in s_1 \parallel X_1 \parallel X_2 \ s_2 \land R = R_1 \cup R_2 \land \sigma(R_1) \cap (X_2 \setminus X_1) = \emptyset \land \sigma(R_2) \cap (X_1 \setminus X_2) = \emptyset \land (s_1, R_1) \in \mathcal{T}_{C_i}[Q_1] \land (s_2, R_2) \in \mathcal{T}_{C_i}[Q_2]
\]

We require the following definitions:

\[
\begin{align*}
T & = \max\{\text{end}(s), t\} + t' \\
T' & = T + t' \\
B_1 & = A \cap Y_1 \cap (\sigma(R_1) \uparrow [T, T']) \\
B_2 & = A \cap Y_2 \cap (\sigma(R_2) \uparrow [T, T']) \\
C_1 & = A \cap Y_1 \setminus (\sigma(R_1) \uparrow [T, T']) \\
C_2 & = A \cap Y_2 \setminus (\sigma(R_2) \uparrow [T, T']) \\
D_1 & = A \cap (X_1 \setminus X_2) \\
D_2 & = A \cap (X_2 \setminus X_1)
\end{align*}
\]

112
Now $B_I \subseteq \sigma(\mathcal{N} \uparrow [T, T'])$, so by non-retraction of $Q_I$ on $Y_I$ and lemma 5.1.16 we have

$$(s_I, \mathcal{N}_I \cup \{end(s), T\} \times B_I) \in \mathcal{S}\mathcal{T}_T[Q_I]$$

so by promptness on $Y_1 \cup Y_2$ and $B_I \subseteq Y_I$ we have

$$(s_I, \mathcal{N}_I \cup [T, \infty) \times B_I) \in \mathcal{S}\mathcal{T}_T[Q_I]$$

Now

$$(T, T') \times C_I \subseteq \mathcal{N}_I \cup [T, \infty) \times B_I$$

so

$$(s_I, \mathcal{N}_I \cup [T, \infty) \times (C_I \cup B_I)) \in \mathcal{S}\mathcal{T}_T[Q_I]$$

Clearly

$$[T, \infty) \times D_I \cap \mathcal{N}_I = \emptyset$$

and so

$$[T, \infty) \times D_I \subseteq \mathcal{N}_I$$

Therefore

$$(s_I, \mathcal{N}_I \cup [T, \infty) \times (D_I \cup C_I \cup B_I)) \in \mathcal{S}\mathcal{T}_T[Q_I]$$

By a symmetric argument, we obtain

$$(s_2, \mathcal{N}_2 \cup [T, \infty) \times (D_2 \cup C_2 \cup B_2)) \in \mathcal{S}\mathcal{T}_T[Q_2]$$

Hence by the definition of $\subseteq_f$-refinement we obtain

$$(tstrip(s_I), D_I \cup C_I \cup B_I) \in \mathcal{F}[P_I]$$

and

$$(tstrip(s_2), D_2 \cup C_2 \cup B_2) \in \mathcal{F}[P_2]$$

Now

$$D_I \cup C_I \cup B_I \cup D_2 \cup C_2 \cup B_2 = A$$

and

$$tstrip(s) \in tstrip(s_I) \parallel_{X_1} tstrip(s_2)$$

and so

$$(tstrip(s), A) \in \mathcal{F}[P_1 \parallel_{X_1} P_2]$$

as required. □

**Corollary 6.3.28** If $P_I \subseteq_f Q_I$ and $P_2 \subseteq_f Q_2$ and both $Q_I$ and $Q_2$ are prompt, and $Q_I$ is non-retracting, then

$$P_I \parallel_Y P_2 \subseteq_f Q_I \parallel_Y Q_2$$
Proof If \( Q_i \) is non-retracting then it is non-retracting on \( X \cap Y \); \( Q_i \) and \( Q_2 \) are both prompt on \( X \cap Y \), so the previous theorem applies with \( Y_i = X \cap Y \), \( Y_2 = \emptyset \).

Theorem 6.3.29 If \( P_1 \subseteq_f Q_1 \) and \( P_2 \subseteq_f Q_2 \) and \( Q_1 \) and \( Q_2 \) are boundedly stable, then \( P_1 X \parallel Y P_2 \subseteq_f Q_1 X \parallel Y Q_2 \)

Proof Consider
\[
(s, N) \in ST_\forall [Q_1 X \parallel Y Q_2], [t, \infty) \times Z \subseteq N
\]
Then \( Q_i X \parallel Y Q_2 \) is boundedly stable, so
\[
(s, \alpha, [t, \max\{t, \alpha\} + 1) \times Z) \in E_\forall [Q_1 X \parallel Y Q_2]
\]
so for some
\[
\mathcal{N}_i, \mathcal{N}_g, \alpha_i \leq \alpha, \alpha_g \leq \alpha
\]
we have
\[
(s \uparrow X, \alpha_i, \mathcal{N}_i) \in E_\forall [P] \land \sigma(\alpha_X) \cap (Y \setminus X) = \emptyset
\]
and
\[
(s \uparrow Y, \alpha_g, \mathcal{N}_g) \in E_\forall [Q] \land \sigma(\alpha_Y) \cap (X \setminus Y) = \emptyset
\]
and
\[
\mathcal{N}_i \cup \mathcal{N}_g = [t, \max\{t, \alpha\} + 1) \times Z
\]
Let
\[
Z_i = \sigma(\mathcal{N}_i \uparrow \alpha_i), Z_g = \sigma(\mathcal{N}_g \uparrow \alpha_g)
\]
Then \( Z_i \cup Z_g = Z \) Now from theorem 3.6.6 we have that
\[
(s \uparrow X, \alpha_i, \infty) \times Z_i) \in ST_\forall [Q_1]
\]
and
\[
(s \uparrow Y, \alpha_g, \infty) \times Z_g) \in ST_\forall [Q_2]
\]
Hence
\[
(tstrip(s \uparrow X), Z_i) \in F[P_1]
\]
and
\[
(tstrip(s \uparrow Y), Z_g) \in F[P_2]
\]
so
\[
(tstrip(s), Z_i \cup Z_g) \in F[P_1 X \parallel Y P_2]
\]
as required. \( \Box \)

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 \( \subseteq_f \)-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

\[
(s, [t, \infty) \times X) \in ST_T[P \setminus A] \Rightarrow \\
\exists w \cdot \#w \text{ is finite } \land (w, [0, \infty) \times A \cup [t, \infty) \times X) \in IT_P \land w \setminus A = s
\]

Observe that not all processes are well behaved on $A$. For example, the process

\[
P \equiv n : \mathbb{N} \rightarrow \text{WAIT } n ; a \rightarrow \text{STOP}
\]

is not well behaved on $\mathbb{N}$. When the set $\mathbb{N}$ is hidden, the choice of $n$ made by $P$ is not visible, so $a$ can be refused for any finite length of time: $\forall n \bullet (\langle \rangle, [0, n) \times \{a\}) \in FT[P \setminus \mathbb{N}]$. The limit of these finite approximations is therefore a semi-infinite behaviour of $P \setminus \mathbb{N}$:

\[
(\langle \rangle, [0, \infty) \times \{a\}) \in ST_T[P \setminus \mathbb{N}]
\]

However, there is no single semi-infinite behaviour of $P$ that gives rise to that behaviour of $P \setminus \mathbb{N}$: that is, there is no trace $w$ of $P$ such that $w \setminus \mathbb{N} = \langle \rangle$ and

\[
(w, [0, \infty) \times (\mathbb{N} \cup \{a\})) \in ST_T[P]
\]

Hence $P$ is not well behaved on $\mathbb{N}$ (and so we would not expect $P_i \subseteq I P$ to imply that $P_i \setminus \mathbb{N} \subseteq I P \setminus \mathbb{N}$).

Lemma 6.3.31 If $Q$ is well behaved on $A$, and $P \subseteq I Q$, then $P \setminus A \subseteq I Q \setminus A$

Proof Consider

\[
(s, R) \in ST_T[Q \setminus A] \land ([t, \infty) \times X) \subseteq R
\]

Then

\[
(s, [t, \infty) \times X) \in ST_T[Q \setminus A]
\]

so

\[
\exists w \cdot s = w \setminus A \land \exists (w, [t, \infty) \times X \cup [0, \infty \times A) \in ST_T[Q]
\]

Now $P \subseteq I Q$, so

\[
(tstrip(w), A \cup X) \in FT[P]
\]

so

\[
(tstrip(w) \setminus A, X) \in FT[P \setminus A]
\]

so

\[
(tstrip(s), X) \in FT[P \setminus A]
\]

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

$$(s,[t',\infty) \times B) \in ST_P[P \setminus A]$$

Then

$$\exists n \cdot w \in \text{traces}(F_P[P]) \land w \setminus A = s \Rightarrow \# w < n$$

Define $T = \max\{\text{end}(s), t'\} + nt + 1$. Then

$$(s,[t', T) \times B) \in F_P[P \setminus A]$$

so

$$\exists w \cdot w \setminus A = s \land (w,[t', T) \times B \cup [0,T) \times A) \in F_P[P]$$

Then

$$\#(w \uparrow \text{end}(s)) < n$$

so

$$\exists T' < T - t \cdot T' > t' \land w \uparrow [T', T' + t] = \langle \rangle$$

Define $w' = w \uparrow T' = w \uparrow T' + t$. Then

$$(w',[t', T' + t) \times B \cup [0,T' + t) \times A) \in F_P[P]$$

so since

$$T' > \text{end}(w') \land [T', T' + t) \times A \cup B \subseteq [t', T' + t) \times B \cup [0,T' + t) \times A$$

we have by weak promptness

$$(w',[t', \infty) \times B \cup [0,\infty) \times A) \in F_P[P]$$

as required. $\square$

Observe that we require that $P$ is weakly limited: if it is not, then the result need not follow. Consider the following process definition:

$$Q_{n,0} \triangleq b \rightarrow \text{STOP}$$

$$Q_{n,m+1} \triangleq n \rightarrow Q_{n,m}$$

$$Q \triangleq n : N \rightarrow Q_{n,n}$$

116
Then \( Q \) is weakly prompt, but is not weakly limited on \( N \). We see that since all the finite approximations are failures of \( P \setminus N \). However, there is no finite trace \( w \) such that

\[
(w, [\emptyset, \infty) \times N \cup \{b\}) \in SI_T[Q \setminus N]
\]

so \( Q \) is not well behaved on \( N \). Indeed, although \( \Theta(Q) \subseteq Q \), it is not the case that \( \Theta(Q) \setminus N \subseteq Q \setminus N \).

**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 \( P_1 \subseteq Q_1 \) and \( P_2 \subseteq Q_2 \) and \( P_i \) and \( Q_i \) and \( Q_k \) are both weakly prompt, \( Q_i \) is non-retracting on \( A \) and weakly limited on \( A' \), \( Q_k \) is non-retracting on \( B \) and weakly limited on \( B' \), and \( A \cup B = A' \cup B' = X \cap Y \), then

\[
(P_1 \parallel Y P_2) \setminus (X \cap Y) \subseteq (Q_i \parallel Y Q_k) \setminus (X \cap Y)
\]

**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 \( \subseteq_f \)-refinement:

**Definition 6.3.35** A set of processes \( \{P_i\} \) are well behaved if

\[
SI_T[\bigcap_i P_i] = \bigcup_i SI_T[P_i]
\]

Not all sets of processes are well behaved. If \( P_n = \text{WAIT} n \; ; \; a \rightarrow \text{STOP} \) then we have

\[
(\emptyset, [\emptyset, \infty) \times \{a\}) \in IT[\bigcap_i P_i]
\]

but

\[
(\emptyset, [\emptyset, \infty) \times \{a\}) \notin \bigcup_i IT[P_i]
\]

so the set \( \{P_i\} \) is not well behaved.

**Lemma 6.3.36** If \( \{Q_i\} \) is well behaved, and \( \forall i \cdot P_i \subseteq_f Q_i \) then \( \bigcap_i P_i \subseteq_f \bigcap_i Q_i \).
Proof Let \( \{ Q_i \} \) be well behaved, and \( \forall i \cdot P_i \subseteq_f Q_i \). Consider

\[
(s, \mathcal{R}) \in SI_T[\bigcap_i Q_i] \cdot [t, \infty) \times A \subseteq \mathcal{R}
\]

Then

\[
\Rightarrow \exists i \cdot (s, \mathcal{R}) \in SI_T[Q_i]
\]

\[
\Rightarrow \exists i \cdot (tstrip(s), A) \in \mathcal{F}[P_i]
\]

\[
\Rightarrow (tstrip(s), A) \in \mathcal{F}[\bigcap_i P_i]
\]

as required. \( \Box \)

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 \( \subseteq_f \)-refinement relation for such processes.

Stable Processes

If \( Q \) is a stable process, then we have

\[
P \subseteq_f Q \iff (tstrip(s), \sigma(\mathcal{R} \uparrow \alpha)) \in \mathcal{F}_T[P]
\]

This follows from the fact that for stable processes we have

\[
(\exists(s, \mathcal{R}) \in SI_T[P] \cdot [t, \infty) \times X \subseteq \mathcal{R} \iff \exists(s, \alpha, \mathcal{R}) \in \mathcal{E}_T[P] \cdot \sigma(\mathcal{R} \uparrow \alpha) = X
\]

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

\[
P \subseteq_f Q \equiv
\]

\[
(s, \mathcal{R}) \in \mathcal{F}_T[Q] \land [end(s), end(s) + t) \times X \subseteq \mathcal{R} \Rightarrow (tstrip(s), X) \in \mathcal{F}[P]
\]

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 $\subseteq$ [Ros82, Hoa85] is defined between processes in a semantic model (with $\mathcal{E}$ as the corresponding semantic mapping) by

**Definition 6.3.37**

$$P \subseteq Q \iff \mathcal{E}[P \cap Q] = \mathcal{E}[P]$$

Refinement in the non-determinism order preserves $\subseteq_f$-refinement

**Theorem 6.3.38**

$$P \subseteq P' \subseteq_f Q \Rightarrow P \subseteq_f Q$$

$$P \subseteq_f Q \subseteq Q' \Rightarrow P \subseteq_f Q'$$

**Proof** This follows immediately from the facts that

$$\mathcal{E}_T[P \cap Q] = \mathcal{E}_T[P] \Rightarrow \text{fail}(\mathcal{E}_T[Q]) \subseteq \text{fail}(\mathcal{E}_T[P])$$

$$\mathcal{F}[P \cap Q] = \mathcal{F}[P] \Rightarrow \text{fail}(\mathcal{F}[Q]) \subseteq \text{fail}(\mathcal{F}[P])$$

Specifications

The treatment for $\subseteq_f$-refinement of specifications is very similar to the treatment for $\subseteq_f$-refinement. We extend $\subseteq_f$-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 $\subseteq_f$ between specifications in an identical fashion to the corresponding definition for $\subseteq_f$-refinement:

**Definition 6.3.39**

$$S_1 \subseteq_f S_2 \equiv \forall Q \cdot (\exists P \cdot S_1(P) \land P \subseteq_f Q \Rightarrow S_2(Q))$$

We can also define the strongest refinement $sr(S)$ of a given untimed predicate $S$ in the same way:

$$S \subseteq_f sr(S) \land ((S \Rightarrow sr(S)) \Leftrightarrow S \subseteq_f S_1)$$

It is given by a similar condition:

$$sr(S)(Q) \Leftrightarrow \exists P \cdot S(P) \land P \subseteq_f Q$$
We can again capture the *weakest coarsening* of a given timed predicate $T$. It is given by the following:

$$we(T)(P) \Leftrightarrow \forall Q \cdot (P \subseteq Q \Rightarrow T(Q))$$

We can express the strongest refinement of a behavioural specification on $MF$ as a behavioural specification on the semi-infinite behaviours of a timed process:

**Theorem 6.3.40**

$$sr(Y \text{ sat } A(tr, X))(Q) \Leftrightarrow Q \text{ sat } (\mathcal{R} = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$$

**Proof** It is clear that $Y \text{ sat } (\mathcal{R} = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$ is a refinement of $Y \text{ sat } A(tr, X)$. Further, if $Q \text{ sat } (\mathcal{R} = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$, then the set

$$U = \{(tstrip(s), X) | \exists t \cdot (s, [t, \infty) \times X) \in \mathcal{ST}(Q)\}$$

satisfies the axioms of $MF$ (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 $v$:

$$P_{(s,X),v} = a : (\text{inits}(U \uparrow v)) \rightarrow P_{(s,X),v}(a) \quad \text{if } u \neq v$$

$$P_{(s,X),v} = a : (\text{inits}(U \uparrow v) - X) \rightarrow P_{(s,X),v}(a) \quad \text{if } u = v$$

Now axiom 4 for $MF$ yields for any $(u, X) \in U$ that

$$(u, X) \in \mathcal{F} \left[ P_{(s,X),()} \right] \subseteq U$$

so we obtain

$$U \subseteq \bigcup_{(u, X) \in U} (P_{(s,X),()}) \subseteq U$$

and hence

$$\mathcal{F} \left[ \bigcap_{(u, X) \in U} (P_{(s,X),()}) \right] = U$$

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 $U$. $\Box$

**Lemma 6.3.41**

$$Y \text{ sat } (\mathcal{R} = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$$

$$\Leftrightarrow Y \text{ sat } (([t, \infty) \times X \subseteq \mathcal{R}) \Rightarrow A(tstrip(s), X))$$

120
Proof Assume $Y$ sat $(N = [t, \infty) \times X \Rightarrow A(tstrip(s), X))$. Then consider $(s, N) \in S\mathcal{I}_T[Y]$ such that $[t, \infty) \times X \subseteq N$. By axiom 10 for $TM_{FS}$ we deduce $(s, [t, \infty) \times X) \in S\mathcal{I}_T[Y]$, 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 $\sqsubseteq_r$-refinement:

Definition 6.3.42

$$wc(T)(P) \iff \forall Q \cdot (P \sqsubseteq_f Q \Rightarrow T(Q))$$

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

$$wc(Y \text{ sat } (N = [t, \infty) \times X \Rightarrow S(tstrip(s), X)) = Y \text{ sat } S(tr, X)$$

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

$$\exists(s, N), t \cdot \mathcal{F}_T[P/((s, N), t)] = \mathcal{F}_T[P/((s, N), t) \cap STOP]$$

This is equivalent to saying that

$$\exists(s, N), t \cdot (s, N \cup [t, \infty) \times \Sigma) \in S\mathcal{I}_T[P]$$

This leads to a definition of deadlock free.

Definition 6.3.45 A process $P$ is deadlock-free if

$$(s, N) \in S\mathcal{I}_T[P] \Rightarrow \forall t \cdot [t, \infty) \times \Sigma \not\subseteq N$$

Lemma 6.3.46 $sr(\text{deadlock-free}) = \text{deadlock-free}$

121
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

\[
sr(Y \text{ sat } X \neq \Sigma) = Y \text{ sat } (\{[t, \infty) \times X \subseteq \Delta \} \Rightarrow X \neq \Sigma)
\]

which is equivalent to our definition of deadlock-freedom for \( TM_{FS} \).

We also define deadlock-freedom for \( TM_{PS} \).

**Definition 6.3.47** A process \( P \) is deadlock-free in \( TM_{PS} \) if

\[
s \in \text{traces}(\mathcal{E}_T[P]) \Rightarrow (s, \Sigma) \not\in \text{fail}(\mathcal{E}_T[P])
\]

**Lemma 6.3.48** If \( P \) is a deadlock-free CSP process, and \( Q \) is a TCSP process such that \( P \preceq \omega Q \), then \( Q \) is deadlock-free in \( TM_{FS} \).

**Proof** Trivial

**Lemma 6.3.49** If a TCSP process \( Q \) is deadlock-free in \( TM_{PS} \) then it is deadlock-free in \( TM_{FS} \).

**Proof** If \( Q \) is deadlock-free in \( TM_{PS} \) then it must be stable (since \( \Sigma \) is refusible after an unstable trace), and unable to refuse \( \Sigma \) after stability. Since \( Q \) follows \( Q \), we also have that \( Q \) is stable in \( TM_{FS} \), and unable to refuse \( \Sigma \) after stability. Hence \( Q \) is deadlock-free in \( TM_{FS} \). \( \square \)

Hence we see that deadlock-freedom is preserved by both \( \preceq_f \)-refinement and \( \preceq_\omega \)-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 \( \preceq \), relation between \( MS \) and \( TM_{FS} \).

**Definition 6.4.1** If \( S_1 \in MS \), and \( S_2 \in TM_{FS} \), then

\[
S_1 \preceq, S_2 \equiv
\text{tstrip(traces}(S_2)) \subseteq \text{traces}(S_1) \land (s, \infty) \in \text{stab}(S_2) \Rightarrow (\text{tstrip}(s), \infty) \in S_1
\]

Hence this form of timewise refinement preserves stability in processes: if \( P \) is stable, and \( P \preceq, Q \), 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

$$S_1 \sqsubseteq S_2 \equiv tstrip(traces(S_2)) \subseteq traces(S_1) \land (s, \infty) \in stab(S_2) \Rightarrow (thstrip(s), \infty) \in S_1$$

The refinement relations $\sqsubseteq_I$ and $\sqsubseteq_I$ 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

$$S_1 \sqsubseteq S_2 \equiv (\exists t \bullet (s, [t, \infty) \times X) \in SI(S_2)) \Rightarrow (tstrip(s), X) \in fail(S_1))$$

$$\land (\exists t, \forall T > t \bullet (s, \infty, [t, T) \times X) \in S_2) \Rightarrow (tstrip(s), \infty, X) \in S_1$$

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

$$S_1 \sqsubseteq S_2 \equiv (s, \alpha, X) \in S_2 \Rightarrow (thstrip(s), X) \in fail(S_1)$$

$$\land (\alpha = \infty \Rightarrow (thstrip(s), \infty, X) \in S_1)$$

Each of the refinement relations above extends to environments and to relations between $CSP$ and $TCSP$ terms, in the same way as the relations $\sqsubseteq_I$ and $\sqsubseteq_I$ 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 $\subseteq_f$-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 $c.V$.

\[ c?x : V \to P(x) \equiv c.x : c.V \to Q(c.x) \quad \text{where} \quad Q(c.x) = P(x) \]

\[ c!x \to P \equiv c.x \to P \]

$c?x : V \to P(x)$ 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 context, then we may omit it and write $c?x \to P(x)$.

The construction $c!x \to P$ 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 $tstrip(s \uparrow c)$. This convention will allow us to succinctly relate the messages passed along different channels in specifications. We therefore write $out \leq in$ as shorthand for $tstrip(s \uparrow out.V) \leq tstrip(s \uparrow in.V)$, to specify that the sequence of messages passed on the $out$ channel is a prefix of the sequence passed on the $in$ channel. ($tr_1 \leq tr_2$ means that $tr_1$ is a prefix of $tr_2$; also, from [PS88], $tr_1 \leq_n tr_2$ means that $tr_1 \leq tr_2$ and $\#tr_1 + n \geq \#tr_2$.)

Definition 7.1.1 A pipe is a process $P$ such that $\sigma(P) \subseteq in.\Sigma \cup out.\Sigma$.

The definition 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:

$$ P \gg Q = (S_{out, a}(P) \|_{\{i, a\}} S_{c, in}(Q)) \setminus c $$

where $S_{a,b}$ is an alphabet transformation, indexed by channel names, defined by

$$ S_{a,b}(a.x) = b.x $$
$$ S_{a,b}(b.x) = a.x $$
$$ S_{a,b}(y) = y \text{ if } y \notin a.\Sigma \cup b.\Sigma $$

**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. $P \|_{Y \cup Z} (Q \|_{Y \cup Z} R) = (P \|_{Y \cup Z} Q) \|_{Y \cup Z} R$
2. $\{a, b\} \cap \sigma(P) = \{\} \Rightarrow S_{a,b}(P) = P$
3. $f$ bijective $\Rightarrow f(P \|_{Y} Q) = f(P) f(X) \|_{f(Y)} f(Q) \wedge f(P \setminus c) = f(P) \setminus f(c)$
4. $\forall a, b \cdot S_{a,b}$ 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 $Z \cap Y = \emptyset \Rightarrow (P \setminus Z) X \|_{Y} Q = (P \setminus X \cup Y) \|_{Y} Q \setminus Z$

Let $ic, co, id, do, idc, cdo$ abbreviate $in \cup c, c \cup out, in \cup d, d \cup out, in \cup d \cup c, and, c \cup d \cup out$ 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.
\[(P \gg Q) \gg R\]

\[= (S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg d\]  

(defn)

\[= ((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg c) \gg d\]  

(3, 4)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg c)_{id}\|_{do} S_{d,\text{in}}(R)) \gg d\]  

(A.1.4)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg c)_{id}\|_{do} S_{d,\text{in}}(R)) \gg d\]  

(6)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg d) \gg c\]  

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg c)_{id}\|_{do} S_{d,\text{in}}(R)) \gg d\]  

(3, 4)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg c)_{id}\|_{do} S_{d,\text{in}}(R)) \gg d\]  

(1, 2)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg d) \gg c\]  

(5)

\[= (((S_{\text{out},d}(S_{\text{out},c}(P)_{ic}\|_{co} S_{c,\text{in}}(Q))_{id}\|_{do} S_{d,\text{in}}(R)) \gg d) \gg c\]  

(A.1.4)

\[= P \gg (Q \gg R)\]  

(defn)

\[\Box\]

### 7.2 Real-time Buffers

In addition to being interesting in their own right, buffers 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

\[P \text{ sat } out \subseteq in\]

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, p.158]:

$$P \text{ sat }(out \leq in \land \text{ if } out = in \text{ then } in \in \text{ ref } \text{ else } out \in \text{ ref})$$

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, that a process is a timed buffer if there is no time after which it can always refuse 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 pipe such that

$$(s,\mathcal{R}) \in \mathcal{SI}_T[P] \Rightarrow (out \leq in)$$

\[ \land \text{ if } out = in \]

then \[ \forall t \in [\text{end}(s), \infty), u \in C \cdot [t, \infty) \times in.u \not\subseteq \mathcal{R} \]

else \[ \forall t \in [\text{end}(s), \infty) \cdot [t, \infty) \times out \not\subseteq \mathcal{R} \]

where $C$ is the set of possible messages.

**Lemma 7.2.2** 'timed buffer' = $sr(\text{'untimed buffer'})$

**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

$$sr(B)(Q) \Leftrightarrow \exists P \cdot B(P) \land P \sqsubseteq_f Q$$

so we are required to show that

$$C(Q) \Leftrightarrow \exists P \cdot B(P) \land P \sqsubseteq_f Q$$

We prove first that

$$C(Q) \Rightarrow \exists P \cdot B(P) \land P \sqsubseteq_f Q$$

Consider $(s,\mathcal{R}) \in \mathcal{SI}_T[Q], [t, \infty) \times A \subseteq \mathcal{R}$.

\[ (tstrip(s), A) \in \mathcal{F}[P] \land out \leq in \]

\[ \land out = in \Rightarrow in.u \not\subseteq A \]

\[ \land out < in \Rightarrow out \not\subseteq A \]

$$\Rightarrow out \leq in$$

\[ \land out = in \Rightarrow [t, \infty) \times in.u \not\subseteq \mathcal{R} \]

\[ \land out < in \Rightarrow [t, \infty) \times out \not\subseteq \mathcal{R} \]
so $Q$ is a timed buffer.

We now show that

$$C(Q) \Rightarrow \exists P \cdot B(P) \land P \subseteq f Q$$

Define:

$$B() = \text{in} ? x \rightarrow B(x)$$

$$B_{(y) = s} = \text{out} ! y \rightarrow B_s$$

$$\exists$$

$$\text{in} ? x \rightarrow B_{(y) = s} \land \text{out} ! y \rightarrow B_s$$

$$\exists$$

$$P = B()$$

Then $P$ is the most non-deterministic buffer. Its failure set is given by

$$\mathcal{F}[P] = \{(s, X) \mid \text{out} \leq \text{in} \land (\text{out} = \text{in} \Rightarrow \forall u : C \cdot \text{in} . u \notin X) \land \text{out} < \text{in} \Rightarrow \text{out} \notin X\}$$

Consider $Q \in TCSP$ such that $C(Q)$. Let $(s, \mathcal{N}) \in \mathcal{S}T_{T}[Q]$. Then $\text{out} \leq \text{in}$. Now let $[t, \infty) \times A \subseteq \mathcal{N}$.

Case $\text{out} = \text{in}$

Then $\forall u : C \cdot \text{in} . u \notin A$.

Case $\text{out} < \text{in}$

Then $\text{out} \notin A$.

Hence in either case, $(\text{tstrip}(s), A) \in \mathcal{F}[P]$. Therefore $P \subseteq f Q$, as required.

$\Box$

Timewise refinement preserves the property that a process is a buffer.

### Categorising buffers

Buffers may be classified according to their 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

$$\exists (s, R) \in \mathcal{F}_T[P], T \in (0, \infty) \cdot [T, T + t) \times \Sigma \subseteq R \wedge s \uparrow [T, T + t) = \emptyset$$

So $(s \uparrow T + t, R \uparrow T + t) \in \mathcal{F}_T[P]$. But $end(s) < T$, so by promptness and the fact that $[T, T + t) \times \Sigma \subseteq R \uparrow T + t$ we obtain

$$(s \uparrow T, R \uparrow (T + t) \cup [T, \infty) \times \Sigma) \in \mathcal{F}_T[P]$$

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

$$B = \mu X \cdot in?n : N \rightarrow (in?y \rightarrow out!n \rightarrow out!y \rightarrow X$$

$$WAIT n ; out!n \rightarrow X$$

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 $out \leq_i in$, 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 $out \leq_i in$, $P$ 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 COPYING has the same definition as its untimed counterpart.

\[ \text{COPY} = \mu X \cdot \text{in}!x \rightarrow \text{out}!x \rightarrow X \]

The results of the preceding chapters yield the following aspects of good behaviour exhibited by COPYING:

- COPY is a buffer

7.3 Chaining

The specification \text{buffer}(tr) of a buffer in the traces model \( M_T \) is that \( \text{out} \leq \text{in} \). It is clear that any timed buffer \( P \) must satisfy \( sr(\text{buffer}(tr)) \), and hence that if \( P \) and \( Q \) are two buffers, then \( P \gg Q \) sat \( sr(\text{buffer}(tr)) \), since chaining preserves \( \text{buffer}(tr) \) in \( M_T \). Hence for any two buffers \( P \) and \( Q \) we obtain that \( P \gg Q \) is a pipe with \( \text{out} \leq \text{in} \). However, \( P \) and \( Q \) may fail to synchronise on their mutual channel. Consider

\[
P = (\mu X \cdot ((\text{in}!x \rightarrow \text{STOP}) \uparrow_2 P_x) \uparrow \text{WAIT} 3 ; X)
\]

\[
P_x = (\mu X \cdot ((\text{out}!x \rightarrow \text{STOP}) \uparrow_2 P) \uparrow \text{WAIT} 3 ; X)
\]

\( P \) 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 non-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 $\subseteq_f$-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 $\subseteq_f$-refined by $P$ and $Q$ respectively, and hence that $B_1 \gg B_2 \subseteq_f P \gg Q$. 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 $TM_F$. 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

$$S_{out,c}(P) \parallel_{\{in,c\}} S_{in,c}(Q)$$

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 $S_{out,c}(P) \parallel_{\{out,c\}} S_{in,c}(Q)$ we have $c \leq n$. in. □

Lemma 7.3.2 If $P$ and $Q$ are buffers then $S_{out,c}(P) \parallel_{\{out,c\}} S_{in,c}(Q)$ is weakly limited on $c$.

**Proof** Follows from the fact that for any trace $s$ of $S_{out,c}(P) \parallel_{\{out,c\}} S_{in,c}(Q)$ we have $c \leq in$. □

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 $B_1 \subseteq_f P$ and $B_2 \subseteq_f Q$. From theorem 6.3.27 we have that

$$S_{out,c}(B_1) \parallel_{\{out,c\}} S_{in,c}(B_2) \subseteq_f S_{out,c}(P) \parallel_{\{out,c\}} S_{in,c}(Q)$$

and it follows that $S_{out,c}(P) \parallel_{\{out,c\}} S_{in,c}(Q)$ 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 $B_1 \gg B_2 \subseteq_f P \gg Q$, 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 on output 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 \( B_1 \subseteq_f Q_1 \) and \( B_2 \subseteq_f Q_2 \), so from theorem 6.3.29 we have
\[
S_{out,c}(B_1) \{in,c\} \| \{out,c\} S_{in,c}(B_2) \subseteq_f S_{out,c}(Q_1) \{in,c\} \| \{out,c\} S_{in,c}(Q_2)
\]
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 \( B_1 \gg B_2 \subseteq_f Q_1 \gg Q_2 \). It follows from theorem 5.6.8 that \( Q_1 \gg Q_2 \) is boundedly stable. \( \square \)

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 \( B_1 \subseteq_f Q_1 \) and \( B_2 \subseteq_f Q_2 \), so from theorem 6.3.29 we have \( S_{out,c}(B_1) \{in,c\} \| \{out,c\} S_{in,c}(B_2) \subseteq_f S_{out,c}(Q_1) \{in,c\} \| \{out,c\} S_{in,c}(Q_2) \). 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 \( B_1 \gg B_2 \subseteq_f Q_1 \gg Q_2 \). It follows from theorem 5.6.7 that \( Q_1 \gg Q_2 \) is stable. \( \square \)

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 $P_1 \subseteq_f Q_1$ and $P_2 \subseteq_f Q_2$, 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 $P_1 \triangleright P_2 \subseteq_f Q_1 \triangleright Q_2$.

Proof Every buffer has $\text{out} \leq \text{in}$, and so it is weakly limited on $\text{out}$. Hence
$(S_{\text{out},c}(Q_1))_{(\text{in},c)} \| (S_{\text{out},c}(Q_2))_{(\text{in},c)}$ is weakly limited on $c$. The result follows from
corollary 6.3.34. □

Corollary 7.3.12 If $P_i \subseteq_f Q_i$ and $P_2 \subseteq_f Q_2$ and $Q_1$ and $Q_2$ are prompt, then
$P_i \triangleright \text{COPY} \triangleright P_2 \subseteq_f Q_i \triangleright \text{COPY} \triangleright Q_2$

In order to ensure that $Q_1$ and $Q_2$ communicate despite their possible inability to
synchronise, we insert $\text{COPY}$ 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 $(\text{process, interface})$ pairs $\{(P_i, X_i)\}$,
where an interface is a subset of $\Sigma$, 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:

$$
\begin{align*}
PAR(\{(P_i, X_i)\}) &= (\| P_i) \\
X &= \bigcup\{X_i \cap X_j \mid 1 \leq i < j \leq n\} \\
NET(\{(P_i, X_i)\}) &= PAR(\{(P_i, X_i)\}) \setminus X
\end{align*}
$$

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

- \((PAR(\{(P, X_i)\}))\) is weakly limited on \(\bigcup\{X_i \cap X_j \mid 1 \leq i < j \leq n\}\)
- \(\forall i \cdot P_i \subseteq_f Q_i\)
- \(PAR(\{(Q_i, X_i)\})\) is weakly prompt

then \(PAR(\{(P, X_i)\}) \subseteq_f PAR(\{(Q_i, X_i)\})\).

Proof \((PAR(\{(P, X_i)\}))\) is weakly limited on \(\bigcup\{X_i \cap X_j \mid 1 \leq i < j \leq n\}\), so it follows from corollary 6.2.13 and lemma 6.2.11, and the fact that all TCSP operators preserve \(\subseteq_f\)-refinement, that \(PAR(\{(Q_i, X_i)\})\) is weakly limited on \(\bigcup\{X_i \cap X_j \mid 1 \leq i < j \leq n\}\). The result follows immediately from lemma 6.3.32 □

Corollary A.1.5 states that

\[
Z \cap Y = \emptyset \Rightarrow (P \setminus Z) \parallel_Y Q = (P_{X \cup Z} \parallel_Y Q) \setminus Z
\]

The following consequences are useful:

Corollary 7.4.2

\[
(P \parallel_Y Q) \setminus Z = (P \setminus ((X - Y) \cap Z)) \parallel_Y Q \setminus (Z - (X - Y))
\]

Corollary 7.4.3

\[
(X \cap Y \cap Z) = \emptyset \land Z \subseteq (X \cup Y)
\]

\[
\Rightarrow (P \parallel_Y Q) \setminus Z = (P \setminus (Z \cap X)) \parallel_Y (Q \setminus (Z \cap Y))
\]

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

$$I = \bigcup_{i=1}^{m} \bigcup_{j=m+1}^{n} \{X_i \cap X_j\}$$

$$Y_1 = I \cup \bigcup_{i=1}^{m} (X_i \setminus (\bigcup_{j \neq i} X_j))$$

$$Y_2 = I \cup \bigcup_{i=m+1}^{n} (X_i \setminus (\bigcup_{j \neq i} X_j))$$

then we have

$$\text{NET}((\{(Q_i, X_i) \mid 1 \leq i \leq n\})) \subseteq \text{NET}((\{(Q_i, X_i) \mid 1 \leq i \leq m\})_{Y_1} \big|_{Y_2} \text{NET}((\{(Q_i, X_i) \mid m + 1 \leq i \leq n\}))$$

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. $\text{PAR}((\{(P_i, X_i)\}))$ is weakly limited on $\bigcup\{X_i \cap X_j \mid 1 \leq i < j \leq n\}$

then $\text{NET}((\{(P_i, X_i)\})) \subseteq_f \text{NET}((\{(Q_i, X_i)\}))$

Proof We first prove, by induction on $r$, that

$$\text{PAR}((\{(P_i, X_i) \mid 1 \leq i \leq r\}) \subseteq_f \text{PAR}((\{(Q_i, X_i) \mid 1 \leq i \leq r\}))$$

The base case ($r = 2$) follows immediately from theorem 6.3.27.

For the inductive step, assume

$$\text{PAR}((\{(P_i, X_i) \mid 1 \leq i \leq r\}) \subseteq_f \text{PAR}((\{(Q_i, X_i) \mid 1 \leq i \leq r\}))$$

Now define the set

$$I = \{i \mid Q_{r+1} \text{ is not non-retracting on } X_i \cap X_{r+1}\}$$
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

$$PAR(\{(Q_i, X_i) \mid 1 \leq i \leq r\})$$

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 $\bigcup_{i \in I}(X_i \cap X_{r+1})$. Further, from the promptness conditions we have that both $PAR(\{(Q_i, X_i) \mid 1 \leq i \leq r\})$ and $Q_{r+1}$ are prompt on $\bigcup_{i \leq i \leq r}(X_i \cap X_{r+1})$, and so we are in a position to apply theorem 6.3.27, from which we conclude that

$$PAR(\{(P_i, X_i) \mid 1 \leq i \leq r + 1\}) \subseteq_f PAR(\{(Q_i, X_i) \mid 1 \leq i \leq r + 1\})$$

Thus we may conclude by induction that

$$PAR(\{(P_i, X_i) \mid 1 \leq i \leq n\}) \subseteq_f PAR(\{(Q_i, X_i) \mid 1 \leq i \leq n\})$$

Now, defining

$$A = \bigcup_{i \leq j \leq n} X_i \cap X_j$$

it follows from lemma 6.2.11 that $PAR(\{(Q_i, X_i) \mid 1 \leq i \leq n\})$ 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

$$NET(\{(P_i, X_i) \mid 1 \leq i \leq n\}) \subseteq_f NET(\{(Q_i, X_i) \mid 1 \leq i \leq n\})$$

□
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, the 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 both 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 \(\text{TIME}\) provides times along a \(\text{time}\) channel. Our specification for it is the conjunction of the following conditions:

1. \(\text{TIME}\) is responsive on \(\text{time}\)
2. \(\text{TIME} \text{ sat } \text{ACCURATE}(s, \mathbb{N})\), where
   \[
   \text{ACCURATE}(s, \mathbb{N}) \equiv \{(t, \text{time}.n) \in s \mid 0 \leq t - n < 1
   \]

Our proposed \(\text{TCSP}\) implementation of a timeserver is given as follows:

\[
\text{TIME} \equiv \mu X \cdot (\mu Y \cdot \text{time}.0 \rightarrow Y) \; \downarrow_{\text{timesucc}(X)}
\]

where the alphabet transformation \(\text{timesucc}\) is defined by

\[
\text{timesucc} (\text{time}.n) \equiv \text{time}.(n + 1), \quad \text{timesucc}(a) \equiv a \text{ if } a \neq \text{time}.n \text{ for all } n
\]

Now \(\text{timesucc}\) clearly preserves "initially \((1 - 3\delta, \text{time})\)-responsive", so by theorem 5.2.17 it immediately follows that \(\text{TIME}\) is responsive on \(\text{time}\) (in fact it is \((1, \text{time})\)-responsive).
We will use the proof system to show that \( \text{TIME} \) sat \( \text{ACCURATE}(s, N) \). Abbreviating \( \text{A}(s, N) \) for \( \text{ACCURATE}(s, N) \), it will be sufficient to show that the defining function for \( \text{TIME} \) preserves \( \text{A}(s, N) \). Assuming \( X \) sat \( \text{A}(s, N) \) we require

\[
(\mu Y \cdot \text{time}.0 \to Y) \overset{\triangledown}{\perp}_{s - 3\delta} \text{timesucc}(X) \text{ sat } \text{A}(s, N)
\]

It is immediate that \( \triangledown \not\in \sigma(\mu Y \cdot \text{time}.0 \to Y) \), 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

\[
\begin{align*}
\mu Y \cdot \text{time}.0 \to Y & \text{ sat } S_1 \\
\text{timesucc}(\text{WAIT} \delta; X) & \text{ sat } S_2
\end{align*}
\]

\[
\begin{align*}
S_1(s \uparrow (1 - 3\delta), \delta \uparrow (1 - 3\delta)) \land s \uparrow ((1 - 3\delta), 1 - \delta) = () \\
\land S_2(s \downarrow (1 - \delta), \delta \downarrow (1 - \delta))
\end{align*}
\]

\( \Rightarrow \) \( \text{A}(s, N) \)

Now we have by assumption that \( X \) sat \( \text{A}(s, N) \), so we have

\[
\text{WAIT} \delta; X \text{ sat } \text{A}(s - \delta, \delta \land \delta) \land \text{begin}(s) \geq \delta
\]

Hence it follows from the rule for alphabet transformation that

\[
\text{timesucc}(\text{WAIT} \delta; X) \text{ sat } \text{A}(\text{timesucc}^{-1}(s - \delta), \text{timesucc}^{-1}(\delta \land \delta))
\]

\( \land \text{begin}(s) \geq \delta \land \sigma(s) \subseteq \text{ran}(\text{timesucc}) \)

which provides us with a candidate for \( S_2 \):

\[
S_2(s, \delta) \equiv \text{A}(\text{timesucc}^{-1}(s - \delta), \text{timesucc}^{-1}(\delta \land \delta))
\]

\( \land \text{begin}(s) \geq \delta \land \sigma(s) \subseteq \text{ran}(\text{timesucc}) \)

The relevant property of \( \mu Y \cdot \text{time}.0 \to Y \) is that the value it is prepared to output on its \text{time} channel is always \( 0 \), so we define \( S_1 \) by

\[
S_1(s, \delta) \equiv ((t, \text{time}.n)) \text{ in } s \Rightarrow n = 0
\]

which may be easily established by recursion induction. We only have to establish the logical condition to complete the proof. It reduces to

\[
\begin{align*}
S_1(s \uparrow (1 - 3\delta), \delta \uparrow (1 - 3\delta)) \land s \uparrow ((1 - 3\delta), 1 - \delta) = ()
\land \text{A}(\text{timesucc}^{-1}(s - 1), \text{timesucc}^{-1}(\delta - 1))
\end{align*}
\]

\( \Rightarrow \) \( \text{A}(s, N) \)

This may be established by a case analysis on a typical timed event of the form \( (t, \text{time}.n) \) 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 \( 0 \leq t - n < 1 \).
Case $t \leq 1 - 3\delta$

Then

$$(t, time.n) \in s \uparrow (1 - 3\delta)$$

and we have

$$s \uparrow (1 - 3\delta), \emptyset \uparrow (1 - 3\delta)$$

so we obtain $n = 0$, and so $0 \leq t - n < 1$ as required.

Case $t \in (1 - 3\delta, 1)$

This case cannot arise since it contradicts $s \uparrow ((1 - 3\delta), 1) = \emptyset$.

Case $t \geq 1$

Then we have

$$A(timesucc^{-1}(s - 1), timesucc^{-1}(\emptyset - 1)) \land \sigma(s) \subseteq \text{ran}(timesucc)$$

and also

$$\langle (t - 1, time.(n - 1)) \rangle \in timesucc^{-1}(s - 1)$$

so we obtain $0 \leq (t - 1) - (n - 1) < 1$, so $0 \leq t - n < 1$ 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

$$\bigsqcup_{i=1}^{n} i : P_i X \llbracket_{I \cup R} (CON \llbracket_{R} RES)$$

where

$$R = \sigma(RES)$$

$$X = \bigcup_{i=1}^{n} \sigma(i : P_i)$$

$$I = X \cap R$$

$I$ 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. \( CON \ \text{sat} \ (\langle t', i.c \rangle) \in s \Rightarrow i = [t'/t] \mod n \)

We define \( CON \) as follows:

\[
CON \triangleq \mu X \bullet F(X) \\
F(X) \triangleq (\mu Y \bullet y : 0.\Sigma \rightarrow Y) \downarrow_{t-3\delta} \text{succ}_n(X)
\]

where

\[
\text{succ}_n(m.a) \triangleq ((m + 1) \mod n).a \\
\text{succ}_n(a) \triangleq a \text{ if } a \notin m.\Sigma \text{ for every } m
\]

We have \( (\text{succ}_n)^n \) is the identity function on \( \Sigma \), so by considering \( CON \) as the fixed point of the function \( (F \circ \text{WAIT} \delta)^n \) we have that

\[
\begin{align*}
P_0 & \triangleq (\mu Y \bullet y : 0.\Sigma \rightarrow Y) \\
P_{m+1} & \triangleq P_0 \downarrow_{t-3\delta} \text{succ}_n(\text{WAIT} \delta ; P_m) \\
CON & \triangleq \mu X \bullet P_{n-1} \uparrow_{nt-3\delta} X
\end{align*}
\]

It may be established by induction on \( m \) that

\[0 \leq r \leq m < n \Rightarrow P_m \text{ is immediately } ((m + 1)t - 3\delta, r.\Sigma)-\text{responsive}\]

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 \( (nt - 3\delta, m.\Sigma)-\text{responsive} \). 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 \( CON \ \text{sat} \ A(s, N) \), where

\[
A(s, N) \triangleq (\langle t', i.c \rangle) \in s \Rightarrow i = [t'/t] \mod n
\]

140
Assuming $X$ sat $A(s, \mathbb{N})$ we must find $S_1$ and $S_2$ such that

$$
\mu Y \cdot y : \emptyset \Sigma \rightarrow Y \text{ sat } S_1
$$

$$
succ_n(WAIT \delta; X) \text{ sat } S_2
$$

$$
S_1(s \uparrow (t - 3\delta), \mathbb{N} \uparrow (t - 3\delta)) \land s \uparrow (t - 3\delta, t - \delta) = ()
\right\} \Rightarrow A(s, \mathbb{N})
$$

The relevant property that will suffice for $S_1$ is given by

$$
S_1(s, \mathbb{N}) \equiv ((t', i, c)) \in s \Rightarrow i = 0
$$

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 $A(s, \mathbb{N})$:

$$
S_2(s, \mathbb{N}) \equiv A(s \uparrow \delta), succ_n(s \uparrow (N \uparrow \delta))
\land begin(s) \geq \delta \land \sigma(s) \subseteq ran(succ_n)
$$

The third proof obligation above can easily be established by a case analysis on the time $t'$ of any given $((t', i, c))$ in $s$, the cases being $t' \leq t - 3\delta$, $t' \in (t - 3\delta, t)$, and $t' \geq t$. □

### 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 the 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 $(s \uparrow \text{reset}) \uparrow [t_i, t_i + t] = () \Rightarrow (s \uparrow \text{reset}) \uparrow t_i = ()$
2. $WTIM$ sat $\#\text{alarm} = 0 \Rightarrow \text{reset} \notin \sigma(\emptyset \uparrow [\text{end}(s) + 2\delta, \text{end}(s) + t])$
3. $WTIM$ sat $s = s' \uparrow ((t', \text{alarm})) \Rightarrow t' \geq \text{end}(s') + t$
4. $WTIM$ sat $\#\text{alarm} = 0 \Rightarrow [\text{end}(s), \text{end}(s) + t + T] \times \{\text{alarm}\} \subseteq \mathbb{N}$
5. $WTIM$ is strongly non-retracting on $\text{alarm}$
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 the 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 possible, 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

$$WTIM \equiv WAIT 2\delta ; TIM$$

$$TIM \equiv (\mu X \cdot (reset \rightarrow X)^{t-2\delta} \text{alarm} \rightarrow SKIP$$

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. Since “strongly non-retracting on alarm” is closed we need only prove that it is preserved by

$$F(X) \equiv (\mu X \cdot (reset \rightarrow WAIT \delta ; X)^{t-2\delta} \text{alarm} \rightarrow SKIP$$

If $X$ is strongly non-retracting on alarm then $reset \rightarrow WAIT \delta ; X$ is also strongly non-retracting on alarm and $\text{inits}(reset \rightarrow WAIT \delta ; X) \cap \{\text{alarm}\} = \emptyset$. We also have $\text{alarm} \rightarrow SKIP$ is strongly non-retracting on alarm, so by theorem 5.1.8 we conclude that $(reset \rightarrow WAIT \delta ; X)^{t-2\delta} \text{alarm} \rightarrow SKIP$ is strongly non-retracting on alarm, as required. \( \square \)

In order to prove that $WAIT 2\delta ; TIM$ sat $W_5(s, \kappa)$, the rule for delay yields that it will be sufficient to prove that $TIM$ sat $W_5$

$$W_5 = W_1(s, \kappa) \land ((s \uparrow \text{reset}) \uparrow t - 2\delta = () \Rightarrow s \uparrow \text{reset} = ())$$

We prove this by recursion induction. Assuming $X$ sat $W_5(s, \kappa)$, we wish to prove that

$$(reset \rightarrow WAIT \delta ; X)^{t-2\delta} \text{alarm} \rightarrow SKIP \text{ sat } W_5(s, \kappa)$$

142
We must find $S_1$ and $S_2$ such that

\[
\begin{align*}
\text{reset} &\rightarrow \text{WAIT} \delta; X \quad \text{sat} \quad S_1 \\
\text{alarm} &\rightarrow \text{SKIP} \quad \text{sat} \quad S_2 \\
\begin{cases}
\begin{align*}
\text{begin}(s) &\leq t - 2\delta \land S_1(s, \delta) \\
\lor
\text{begin}(s) &\geq t - \delta \land S_1((), \delta \uparrow t - 2\delta) \\
\land S_2(s \rightarrow (t - \delta), \delta \rightarrow (t - \delta))
\end{align*}
\end{cases}
\Rightarrow W_\delta(s, \delta)
\end{align*}
\]

The property of $\text{alarm} \rightarrow \text{SKIP}$ contributing to the correctness of the construct is that it is unable to perform $\text{reset}$. Hence we may put

\[
S_2(s, \delta) \triangleq s \uparrow \text{reset} = ()
\]

From the rules for prefixing and delay, we derive a suitable $S_1$:

\[
S_1(s, \delta) = \begin{cases}
\begin{align*}
\text{begin}(s) &\leq t - 2\delta \land S_1(s, \delta) \\
\lor
\text{begin}(s) &\geq t - \delta \land S_1((), \delta \uparrow t - 2\delta) \\
\land S_2(s \rightarrow (t - \delta), \delta \rightarrow (t - \delta))
\end{align*}
\end{cases}
\]

Our third proof obligation

\[
\begin{align*}
\text{begin}(s) &\leq t - 2\delta \land S_1(s, \delta) \\
\lor
\text{begin}(s) &\geq t - \delta \land S_1((), \delta \uparrow t - 2\delta) \\
\land S_2(s \rightarrow (t - \delta), \delta \rightarrow (t - \delta))
\Rightarrow W_\delta(s, \delta)
\end{align*}
\]

is now straightforward to establish. □

We now prove that $\text{WAIT} 2\delta; \text{TIM} \text{sat} \ W_\delta(s, \delta)$, the rule for delay yields that it will be sufficient to prove that

\[
\begin{align*}
\text{TIM} &\text{ sat } \ W_\delta \\
W_\delta &\triangleq \begin{cases}
\begin{align*}
s &= () \Rightarrow \text{reset} \notin \sigma(\delta \uparrow (t - 2\delta)) \\
\land (s \notin () \land \#\text{alarm} = 0) \Rightarrow \text{reset} \notin \sigma(\delta \uparrow [\text{end}(s) + 2\delta, \text{end}(s) + t])
\end{align*}
\end{cases}
\end{align*}
\]

We prove this by recursion induction. Assuming $X \text{ sat } W_\delta(s, \delta)$, we wish to prove that

\[
(reset \rightarrow \text{WAIT} \delta; X) \xrightarrow{\text{reset} \leq \delta} \text{alarm} \rightarrow \text{SKIP} \quad \text{sat} \quad W_\delta(s, \delta)
\]

The structure of this proof is identical to that of the previous proof: The property of $\text{alarm} \rightarrow \text{SKIP}$ that will do for $S_2$ is that the first action it can do must be an alarm action. Hence we set

\[
S_2 \triangleq s \neq () \Rightarrow \#\text{alarm} > 0
\]
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

\[
\text{reset} \rightarrow \text{WAIT} \delta ; X \quad \text{sat} \quad W_6
\]

Using the law for prefixing, we need only find \( S_3 \) such that

\[
\begin{align*}
\begin{aligned}
W_6 \delta ; X & \quad \text{sat} \quad S_3(s, \mathbb{N}) \\
S(s, \mathbb{N}) & = \langle \rangle \land \text{reset} \notin \sigma(\mathbb{N}) \\
& \quad \Rightarrow \quad W_6(s, \mathbb{N}) \\
\end{aligned} \\
\begin{aligned}
S(s, \mathbb{N}) & = \langle (t', \text{reset}) \rangle \land S(s', \mathbb{N}) \land \text{reset} \notin \sigma(\mathbb{N} \uparrow t') \land \text{begin}(s') \geq t' + \delta \\
& \quad \land \;
S(s', \mathbb{N}) \land (t' + \delta)) \\
& \quad \Rightarrow \quad W_6(s, \mathbb{N})
\end{aligned}
\end{align*}
\]

The second of these obligations is immediately dischargeable. To find \( S_3 \), we have that \( X \text{ sat } W_6(s, \mathbb{N}) \), so from the delay rule we know that

\[
\text{WAIT} \delta ; X \quad \text{sat} \quad W_6(s - \delta, \mathbb{N} - \delta) \land \text{begin}(s) \geq \delta
\]

so we may define

\[
S_3(s, \mathbb{N}) \equiv W_6(s - \delta, \mathbb{N} - \delta) \land \text{begin}(s) \geq \delta
\]

Hence we have discharged our first proof obligation. The third and final proof obligation is simple to discharge, using elementary set arithmetic. \( \Box \)

The proofs that \( \text{WTIM sat } W_3(s, \mathbb{N}) \) and \( \text{WTIM sat } W_4(s, \mathbb{N}) \) are identical in structure to the previous two proofs.

To prove that \( \text{WTIM sat } W_5(s, \mathbb{N}) \), we reduce the proof obligation, using the rule for delay, to

\[
\begin{align*}
\begin{aligned}
\text{TIM} & \quad \text{sat} \quad W_7 \\
W_7 & = \quad \#s \geq 2 \Rightarrow (s = s' \land ((t', \text{alarm})) \Rightarrow t' \geq \text{end}(s') + t \\
& \quad \land \ \
S(s, \mathbb{N}) \equiv (t', \text{alarm})) \Rightarrow t' \geq t - 2\delta
\end{aligned}
\end{align*}
\]

In this case, the instantiations for \( S_1 \) and \( S_2 \) which are sufficient to establish that the defining equation for \( \text{TIM} \) preserves \( W_7 \) are given by

\[
\begin{align*}
S_1(s, \mathbb{N}) & = W_3(s, \mathbb{N}) \\
S_2(s, \mathbb{N}) & = \exists t, t' : s \leq ((t, \text{alarm}), (t', \checkmark))
\end{align*}
\]

Under the assumption that \( X \text{ sat } W_7 \), it is straightforward to establish that

\[
\begin{align*}
\begin{aligned}
\text{reset} & \rightarrow \text{WAIT} \delta ; X \quad \text{sat} \quad S_1 \\
\text{alarm} & \rightarrow \text{SKIP} \quad \text{sat} \quad S_2
\end{aligned} \\
\begin{aligned}
\begin{cases}
\text{begin}(s) \leq t - 2\delta \land S_1(s, \mathbb{N}) \\
\text{begin}(s) \geq t - \delta \land S_1((t, \mathbb{N} \uparrow t - 2\delta)) \\
\land \ 
S_2(s - (t - \delta), \mathbb{N} - (t - \delta))
\end{cases}
\end{aligned}
\end{align*}
\]

144
and hence that $TIM$ sat $W_7$.

We finally prove that $WTIM$ sat $W_4(s, \mathcal{X})$. The proof obligation is first reduced via the rule for delay to

$$TIM \text{ sat } W_8$$

$$W_8 = s = () \Rightarrow \{ end(s), end(s) + t + T - 2\delta \} \times \{ \text{alarm} \} \not\subseteq \mathcal{X}$$

$$(s \neq () \land \#\text{alarm} = 0) \Rightarrow \{ end(s), end(s) + t + T \} \times \{ \text{alarm} \} \not\subseteq \mathcal{X}$$

Providing $T > \delta$, it is sufficient to instantiate $S_I$ and $S_S$ as follows:

$$S_I(s, \mathcal{X}) = W_4(s, \mathcal{X})$$

$$S_S(s, \mathcal{X}) = s = () \Rightarrow \text{alarm} \notin \sigma(\mathcal{X})$$

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:

![Diagram of protocol](image)

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 $\text{PROT}$ to meet the following timed failures specification:

$$\text{SPEC}(s, N) \equiv \text{last}(s) = \text{in} \Rightarrow \text{out} \not\in \sigma(N \uparrow (\text{end}(s) + 2))$$

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 $\text{in}, \text{lm}, \text{rc}$ 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 $\text{SPEC}_P$:

$$\text{SPEC}_P(s, N) \equiv \text{tstrip}(s) \leq (\text{in}, \text{lm}, \text{rc})^* \land$$

$$\text{last}(s) = \text{in} \Rightarrow \text{lm} \not\in \sigma(N \uparrow (\text{end}(s) + 2\delta)) \land$$

$$\text{last}(s) = \text{lm} \Rightarrow \text{rc} \not\in \sigma(N \uparrow (\text{end}(s) + 2\delta)) \land$$

$$\text{last}(s) = \text{rc} \Rightarrow \text{in} \not\in \sigma(N \uparrow (\text{end}(s) + 2\delta)) \land$$

$$s = () \Rightarrow \text{in} \not\in \sigma(N)$$

146
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 $SPEC_Q$:

\[
SPEC_Q(s,\mathcal{N}) \triangleq tstrip(s) \leq (rm, out, lc)^* \land \\
\text{last}(s) = rm \Rightarrow out \notin \sigma(\mathcal{N} \uparrow (end(s) + 2\delta)) \land \\
\text{last}(s) = out \Rightarrow lc \notin \sigma(\mathcal{N} \uparrow (end(s) + 2\delta)) \land \\
\text{last}(s) = lc \Rightarrow rm \notin \sigma(\mathcal{N} \uparrow (end(s) + 2\delta)) \land \\
s = () \Rightarrow rm \notin \sigma(\mathcal{N})
\]

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

\[
SPEC_{W_1}(s,\mathcal{N}) \triangleq tstrip(s) \leq (lm, rm)^* \land \\
\text{last}(s) = lm \Rightarrow rm \notin \sigma(\mathcal{N} \uparrow (end(s) + 1)) \land \\
\text{last}(s) = rm \Rightarrow lm \notin \sigma(\mathcal{N} \uparrow (end(s) + 2\delta)) \land \\
s = () \Rightarrow lm \notin \sigma(\mathcal{N})
\]

\[
SPEC_{W_2}(s,\mathcal{N}) \triangleq tstrip(s) \leq (lc, rc)^* \land \\
\text{last}(s) = lc \Rightarrow rc \notin \sigma(\mathcal{N} \uparrow (end(s) + 1)) \land \\
\text{last}(s) = rc \Rightarrow lc \notin \sigma(\mathcal{N} \uparrow (end(s) + 2\delta)) \land \\
s = () \Rightarrow lc \notin \sigma(\mathcal{N})
\]

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 in section 7.4. We write it here in full in order to be able to apply the inference rules. If we define the sets

\[
X \triangleq \{in, lm, rc\} \\
Y \triangleq \{out, rm, lc\} \\
C \triangleq \{lc, rc\} \\
M \triangleq \{lm, rm\} \\
A \triangleq M \cup C
\]

then the protocol may be defined:

\[
PROT \triangleq ((P_X \parallel_Y Q)_{X \cup Y \parallel_M \cup C} (W_1 \parallel_M \parallel_C W_2)) \setminus A
\]
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:

\[
PROT \text{ sat } SPEC(s, K)
\]

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:

\[
(P_X \parallel Y Q) \circ \parallel_{M \cup C} (W_1 \parallel C W_2) \text{ sat } \sigma(\mathcal{N}) \subseteq A \land \{(0, \text{end}(s, K)) \times A\} \subseteq \mathcal{N} \Rightarrow SPEC(s \setminus A, K - \mathcal{N})
\]

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:

\[
\begin{align*}
P \parallel_X Y Q & \text{ sat } S_1(s, K) \\
W_1 \parallel_C W_2 & \text{ sat } S_2(s, K) \\
\sigma(s_1, R_1) & \subseteq (X \cup Y) \land \sigma(s_2, R_2) \subseteq (M \cup C) \\
\sigma(R_3) & \subseteq \Sigma - (X \cup Y \cup M \cup C) \\
S_1(s_1, R_1) & \land S_2(s_2, R_2) \land s_3 \in s_1 \circ \parallel_X Y \parallel_{M \cup C} s_2 \\
R & = R_1 \cup R_2 \cup R_3 \\
\sigma(\mathcal{N}) & \subseteq A \land \{(0, \text{end}(s_3, K)) \times A\} \subseteq \mathcal{N} \Rightarrow SPEC(s_3 \setminus A, R - \mathcal{N})
\end{align*}
\]

Before we continue, we note that the specification \( SPEC \) is independent of the hidden set of events \( A \), for consider the definition:

\[
SPEC \triangleq \text{last}(s) = \text{in} \Rightarrow \text{out} \notin \sigma(\Sigma \setminus (\text{end}(s) + 2))
\]

Formally, we can show that

\[
SPEC(s, K \upharpoonright (\Sigma - A)) \Rightarrow SPEC(s, K)
\]

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

\[
\begin{align*}
\sigma(s_1, R_1) & \subseteq (X \cup Y) \land \sigma(s_2, R_2) \subseteq (M \cup C) \\
\sigma(R_3) & \subseteq \Sigma - (X \cup Y) \\
S_1(s_1, R_1) & \land S_2(s_2, R_2) \land s_3 \in s_1 \circ \parallel_X Y \parallel_{M \cup C} s_2 \\
((0, \text{end}(s_3, R_1 \cup R_2 \cup R_3)) \times A) & \subseteq R_1 \cup R_2 \cup R_3 \\
\Rightarrow SPEC(s_3 \setminus A, R_1)
\end{align*}
\]
To identify $S_1$ we apply the parallel rule once again. We are then required to find $S_4$ and $S_5$ such that:

$$
P \text{ sat } S_4(s, R)$$

$$Q \text{ sat } S_5(s, R)$$

$$\sigma(s_4, R_4) \subseteq X \land \sigma(s_5, R_5) \subseteq Y$$

$$\sigma(R_5) \subseteq \Sigma - (X \cup Y)$$

$$S_4(s_4, R_4) \land S_5(s_5, R_5) \land s_6 \in s_4 \parallel_Y s_5 \} \Rightarrow S_1(s_6, R_4 \cup R_5 \cup R_6)$$

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:

$$\{ SPEC_P(s \uparrow X, R \uparrow X) \} \Rightarrow S_1(s, R)$$

$$\{ SPEC_Q(s \uparrow Y, R \uparrow Y) \} \Rightarrow S_1(s, R)$$

$$\sigma(s) \subseteq (X \cup Y)$$

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 $S_2$:

$$SPEC_{W_1}(s \uparrow M, R \uparrow M) \land$$

$$SPEC_{W_2}(s \uparrow C, R \uparrow C) \land$$

$$\sigma(s) \subseteq (M \cup C)$$

Our proof requirement can then be written as follows:

$$\{ \sigma(s_1, R_1) \subseteq (X \cup Y) \land \sigma(s_2, R_2) \subseteq (M \cup C), $$

$$\sigma(R_2) \subseteq \Sigma - (X \cup Y) $$

$$SPEC_P(s_1 \uparrow X, R_1 \uparrow X) \land SPEC_Q(s_1 \uparrow Y, R_1 \uparrow Y) $$

$$SPEC_{W_1}(s_2 \uparrow M, R_2 \uparrow M) \land SPEC_{W_2}(s_2 \uparrow C, R_2 \uparrow C) $$

$$([0, \text{end}(s_3), R_1 \cup R_2 \cup R_3]) \times A) \subseteq R_1 \cup R_2 $$

$$s_3 \in s_1 \parallel_Y \|_{M \cup C} s_2 $$

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 antecedent, we are trying to establish that

$$\text{last}(s_3 \setminus A) = \text{in} \Rightarrow \text{out} \notin \sigma(R_1 \uparrow (\text{end}(s_3 \setminus A) + 2))$$

149
From $SPEC_p, SPEC_q, SPEC_w_1, SPEC_w_2$, and the properties of sequences, we can deduce that

$$s_3 \subseteq \langle in, lm, rm, out, lc, rc \rangle^*$$

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 $SPEC_p$, \hfill $lm \not\in \sigma((N_1 \setminus X) \cup (end(s_1 \setminus X) + 2\delta))$

In this case \hfill $end(s_1) = end(s_1 \setminus X)$

and we know that \hfill $lm \not\in Y$

Hence \hfill $lm \not\in \sigma(N_1 \cup \{end(s_3) + 2\delta\})$

Similarly, as \hfill $s_3 \subseteq s_1 \cup Y \cup M \cup C \cup s_2$

$SPEC_w_1$ implies that \hfill $lm \not\in \sigma(N_2 \cup \{end(s_3) + 2\delta\})$

Hence \hfill $lm \not\in \sigma((N_1 \cup N_2 \cup N_3) \cup (end(s_3) + 2\delta))$

However, \hfill $([0, end(s, N_1 \cup N_2 \cup N_3)) \times A) \subseteq (N_1 \cup N_2 \cup N_3)$

and \hfill $lm \in A$

So \hfill $end(N_1 \cup N_2 \cup N_3) \leq end(s_3) + 2\delta$

But $\delta \ll 1$, so \hfill $(N_1 \cup N_2 \cup N_3) \cap (end(s_3) + 2\delta) = \emptyset$

We conclude that \hfill $out \not\in \sigma((N_1 \cup N_2 \cup N_3) \cup end(s_3 \setminus A + 2))$

Case: $last(s_3) = lm$

We establish that $end(s_3) \leq end(s_3 \setminus A) + 2\delta$: that the $lm$ event occurred within time $2\delta$ of the last input.

Assume otherwise: \hfill $end(s_3) > end(s_3 \setminus A) + 2\delta$

If we let $t$ be the time \hfill $(end(s_3 \setminus A) + end(s_3) + 2\delta)/2$

Then we know that \hfill $last(s_3 \setminus t) = in$

By the previous case \hfill $lm \not\in \sigma(((N_1 \cup N_2 \cup N_3) \setminus t) \cup (end(s_3 \setminus t + 2\delta))$

From our assumptions \hfill $([0, end(s_3, N_1 \cup N_2 \cup N_3)) \times A) \subseteq N_1 \cup N_2$

And \hfill $end(s_3 \setminus t + 2\delta = end(s_3 \setminus A) + 2\delta < t$

Hence \hfill $lm \in \sigma(((N_1 \cup N_2 \cup N_3) \setminus t) \cup (end(s_3 \setminus t + 2\delta))$

Forcing a contradiction.

We can show, with a similar argument to the first case, in which the event $rm$ replaces $lm$, that $end(N_1 \cup N_2 \cup N_3) \leq end(s_3) + 1$. From above, $end(s_3) \leq end(s_3 \setminus A) + 2\delta$: the result follows.

150
Case: last(\(s_3\)) = rm

By a similar argument, we can establish that the event \(rm\) must occur no later than \(I + 2\delta\) 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:

\[
\begin{align*}
\sigma(s_1, \mathcal{R}_1) &\subseteq (X \cup Y) \land \sigma(s_2, \mathcal{R}_2) \subseteq (M \cup C) \\
\sigma(\mathcal{R}_2) &\subseteq \Sigma - (X \cup Y) \\
SPEC_p(s_1 \uparrow X, \mathcal{R}_1 \uparrow X) \land SPEC_Q(s_1 \uparrow Y, \mathcal{R}_1 \uparrow Y) \\
SPEC_{W_1}(s_2 \uparrow M, \mathcal{R}_2 \uparrow M) \land SPEC_{W_2}(s_2 \uparrow C, \mathcal{R}_2 \uparrow C) \\
\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_s \land s_2 \in s_1 \downarrow Y || M \cup C s_2 \\
\sigma(\mathcal{R}') \subseteq A \land (\emptyset, end(s_3, \mathcal{R}) \times A) \subseteq \mathcal{R}
\end{align*}
\]

\(\Rightarrow S(s_3 \setminus A, \mathcal{R} - \mathcal{R}')\)

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 the 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 be 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:

\[
P = \mu X \cdot in \rightarrow lm \rightarrow rc \rightarrow X
\]

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 \textit{out}. It should then send a confirmation signal along wire $W_2$, on channel $lc$, before returning to its initial state. Our proposed solution:

$$Q \equiv \mu Y \cdot rm \rightarrow out \rightarrow lc \rightarrow Y$$

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:

$$\mu X \cdot in \rightarrow lm \rightarrow rc \rightarrow X \text{ sat } SPEC_P(s, \mathcal{K})$$

This is a recursive process; the second recursion rule requires us to find a specification $S(s, \mathcal{K})$ such that:

$$X \text{ sat } S(s, \mathcal{K}) \Rightarrow in \rightarrow lm \rightarrow rc \rightarrow (WAIT \delta ; X) \text{ sat } S(s, \mathcal{K})$$

$$S(s, \mathcal{K}) \Rightarrow SPEC_P(s, \mathcal{K})$$

We will show that the specification $SPEC_P$ is strong enough to be preserved by the recursion. We have to show that:

$$X \text{ sat } SPEC_P(s, \mathcal{K}) \Rightarrow in \rightarrow lm \rightarrow rc \rightarrow (WAIT \delta ; X) \text{ sat } SPEC_P(s, \mathcal{K})$$

Assume that $X$ sat $SPEC_P(s, \mathcal{K})$. We wish to establish that:

$$in \rightarrow lm \rightarrow rc \rightarrow (WAIT \delta ; X) \text{ sat } SPEC_P(s, \mathcal{K})$$

Applying the prefix rule three times transforms this proof obligation to the following requirement: we must find a specification $U(s, \mathcal{K})$ such that:

$$WAIT \delta ; X \text{ sat } U(s, \mathcal{K})$$

$$s = (\emptyset) \land in \notin \sigma(\mathcal{K}) \lor$$

$$s = ((t, in))^{s'} \land in \notin \sigma(\mathcal{K} \uparrow t_1) \land$$

$$s' \vdash (t_1 + \delta) = (\emptyset) \land lm \notin \sigma(\mathcal{K} \uparrow (t_1 + \delta)) \lor$$

$$s' \vdash (t_1 + \delta) = ((t_2, lm))^{s''} \land lm \notin \sigma(\mathcal{K} \uparrow (t_1 + \delta) \uparrow t_2) \land$$

$$s'' \vdash (t_2 + \delta) = (\emptyset) \land rc \notin \sigma(\mathcal{K} \uparrow (t_1 + t_2 + 2\delta)) \lor$$

$$s'' \vdash (t_2 + \delta) = ((t_3, rc))^{s''' \land}$$

$$rc \notin \sigma(\mathcal{K} \uparrow (t_1 + t_2 + 2\delta) \uparrow t_3) \land$$

$$U(s''' \vdash (t_1 + t_2 + t_3 + 3\delta), \mathcal{K} \uparrow (t_1 + t_2 + t_3 + 3\delta))$$

\[152\]
With a suitable choice of \( \tau_1, \tau_2, \tau_3 \), this can be transformed to:

\[
WAIT \delta; X \text{ sat } U(s, \mathcal{N})
\]

\[
s = () \land \text{in } \not\in \sigma(\mathcal{N})
\]

\[
s = ((\tau_1, \text{in})) \land \not\in \sigma(\mathcal{N} \uparrow \tau_1) \land \text{lm } \not\in \sigma(\mathcal{N} \uparrow \tau_1 + \delta)
\]

\[
s = ((\tau_1, \text{in}), (\tau_2, \text{lm})) \land \text{in } \not\in \sigma(\mathcal{N} \uparrow \tau_1)
\]

\[
\land \text{lm } \not\in \sigma(\mathcal{N} \uparrow [\tau_1 + \delta, \tau_2])
\]

\[
\land \text{rc } \not\in \sigma(\mathcal{N} \uparrow \tau_2 + \delta)
\]

\[
\Rightarrow \text{SPEC}_p(s, \mathcal{N})
\]

Applying the second form of the delay rule, we can instantiate \( U \) as follows:

\[
U(s, \mathcal{N}) \equiv \text{SPEC}_p(s - \delta, \mathcal{N} - \delta) \land \text{begin}(s) \geq \delta
\]

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 \( \text{SPEC}_p \):

\[
\text{SPEC}_p \equiv tstrip(s) \leq (\text{in, lm, rc})^* \land
\]

\[
\text{last}(s) = \text{in } \Rightarrow \text{lm } \not\in \sigma(\mathcal{N} \uparrow (\text{end}(s) + 2\delta))
\]

\[
\text{last}(s) = \text{lm } \Rightarrow \text{rc } \not\in \sigma(\mathcal{N} \uparrow (\text{end}(s) + 2\delta))
\]

\[
\text{last}(s) = \text{rc } \Rightarrow \text{in } \not\in \sigma(\mathcal{N} \uparrow (\text{end}(s) + 2\delta))
\]

\[
s = () \Rightarrow \text{in } \not\in \sigma(\mathcal{N})
\]

The only non-trivial case corresponds to \( s = ((\tau_1, \text{in}), (\tau_2, \text{lm}), (\tau_3, \text{rc})) \sim \text{u} \). Here we require two arguments, one for each of the cases: \( u = () \), \( u \neq () \). Expanding the specification \( \text{SPEC}_p \) 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 the 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:

\[
P \text{ sat } S_I(s, \mathcal{N}) \\
\quad \frac{S_I(s, \mathcal{N}) \Rightarrow S(f(s), f(\mathcal{N}))}{f(P) \text{ sat } S(s, \mathcal{N})}
\]

For example, we can use the result of the previous section to establish that \( Q \text{ sat } \text{SPEC}_Q \), by defining injective function \( f \) such that:

\[
\begin{align*}
&f(\text{in}) \equiv \text{rm} \\
&f(\text{lm}) \equiv \text{out} \\
&f(\text{rc}) \equiv \text{lc}
\end{align*}
\]

We then observe that:

\[
\text{SPEC}_P(s, \mathcal{N}) = \text{SPEC}_Q(f(s), f(\mathcal{N})) \\
Q = f(P)
\]

The inference rule allows us to conclude that:

\[
Q \text{ sat } \text{SPEC}_Q(s, \mathcal{N})
\]

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:

\[
\begin{align*}
S & \equiv \mu X \bullet (\text{in?}x \rightarrow \text{lm!}x \rightarrow \text{rc?}y \rightarrow X) \\
A_S & \equiv \text{in} \cup \text{lm} \cup \text{rc} \\
R & \equiv \mu X \bullet (\text{rm?}x \rightarrow \text{out!}x \rightarrow \text{lc!}x \rightarrow X) \\
A_R & \equiv \text{rm} \cup \text{out} \cup \text{lc}
\end{align*}
\]

We also have specifications on the media \( M_1 \) and \( M_2 \): \( M_1 \) is prompt and responsive on \( \text{lm} \) and on \( \text{rm} \), and \( M_1 \) sat \( \text{rm} \leq_I \text{lm} \); and \( M_2 \) is prompt and responsive on \( \text{lc} \) and on \( \text{rc} \), and \( M_2 \) sat \( \text{rc} \leq_I \text{lc} \). We further define \( A_{M_1} \equiv \{\text{lm, rm}\} \), and \( A_{M_2} \equiv \{\text{lc, rc}\} \). Then setting \( I \equiv \{S, R, M_1, M_2\} \) we may define \( SAWP \) as follows:

\[
SAWP \equiv \text{NET}(\{(P, A_P) \mid P \in I\})
\]

154
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 $\Theta(S) \subseteq R$, and $\Theta(R) \subseteq S$. It follows from the specifications of $M_1$ and $M_2$ that $COPY \subseteq M_1$, $COPY \subseteq M_2$, $COPY \subseteq M_1$, and $COPY \subseteq M_2$. 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 $R_S \equiv \Theta(S)$, $R_R \equiv \Theta(R)$, $R_{M_1} \equiv COPY[lm, rm/in, out]$ and $R_{M_2} \equiv COPY[lc, rc/in, out]$. It is proved in [PS88] that any trace of $PAR(\{(R_i, A_i)\})$ has

$$rc \leq lc \leq out \leq rm \leq lm \leq in \quad \text{and} \quad rc \leq i in$$

so the channels in the trace appear in the cyclic sequence $(in, lm, rm, out, lc, rc)^\ast$. Hence $PAR(\{(R_i, A_i)\})$ is limited on $lm \cup rm \cup lc \cup rc$, which is the same as $\cup\{A_i \cap A_j \mid i, j \in I, i \neq j\}$.

Thus we apply theorem 7.4.5 to obtain

$$NET(\{(R_i, A_i)\}) \sqsubseteq NET(\{(P, AP)\})$$

It was proved in [PS88] that

$$\Theta(NET(\{(P, AP)\})) \text{ sat } out \leq i in$$

so from theorem 6.2.12 it follows that

$$NET(\{(P, AP)\}) \text{ sat } out \leq i in$$

Standard algebraic techniques may be used to establish that $\Theta(NET(\{(R_i, A_i)\}))$ is deadlock-free, so $NET(\{(P, AP)\})$ is deadlock-free; it is also a pipe, impartial on in (immediate from the syntax), and satisfies $out \leq i in$, 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, 'O', 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:

\[ S \equiv S_0 \]
\[ S_b \equiv in?x \rightarrow lm!z.b \rightarrow S_{x,b} \quad \text{where } b \in \{0, 1\} \]
\[ S_{x,b} \equiv ((rc?a \rightarrow (\text{if } a = b \text{ then } S_{l,b} \text{ else } S_{x,b})) \]
\[ \square \circ \rightarrow lm!x.b \rightarrow S_{x,b}) \setminus \{\square\} \]

\[ R \equiv R_I \]
\[ R_b \equiv rm?x.c \rightarrow (\text{if } c \neq b \text{ then } out!z \rightarrow lc!c \rightarrow R_c \]
\[ \quad \text{else } lc!b \rightarrow R_b) \quad \text{where } b \in \{0, 1\} \]

The conditions assumed to hold of the wires \( M1 \) and \( M2 \) are the following:

\[ M1 \text{ sat } lm \leq rm \]
\[ M2 \text{ sat } lc \leq rc \]

where \( s_I \subseteq s_R \) means that \( s_I \) is a (not necessarily contiguous) subsequence of \( s_R \). 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:

\[ A_S \equiv \{\text{in, lm, rc}\} \]
\[ A_R \equiv \{\text{out, rm, lc}\} \]
\[ A_{M1} \equiv \{\text{lm, rm}\} \]
\[ A_{M2} \equiv \{\text{lc, rc}\} \]

Then setting \( I = \{S, R, M1, M2\} \) we may define \( ABP \) as follows:

\[ ABP \equiv NET(\{(P, A_P) \mid P \in I\}) \]

The result proved in \( M_T \) is that \( ABP \) sat \( out \leq_I in \).

Pictorially, the \( ABP \) is entirely similar to the \( SAWP \):

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]

\[ \text{We refine the sender by making the timeout explicit. Our timed sender } TS \text{ will} \]
timeout at time $t$. The timed receiver will have the same description as the original receiver.

$$TS \equiv TS_0$$

$$TS_b \equiv \text{in?}x \rightarrow \text{lm!}x.b \rightarrow TS_{x.b} \quad \text{where } b \in \{0, 1\}$$

$$TS_{x.b} \equiv ((\text{rc?}a \rightarrow (\text{if } a = b \text{ then } TS_{1-b} \text{ else } TS_{x.b})))$$

$$\uparrow \text{lm!}x.b \rightarrow TS_{x.b}$$

$$TR \equiv TR_t$$

$$TR_b \equiv \text{rm?}x.c \rightarrow (\text{if } c \neq b \text{ then } \text{out!}x \rightarrow \text{lc!}c \rightarrow Rc \quad \text{else } lc!b \rightarrow TR_b) \quad \text{where } b \in \{0, 1\}$$

Recalling from [Hoa85] the law (for the untimed models)

$$(a \rightarrow P) \square b \rightarrow Q) \setminus b = ((a \rightarrow (P \setminus b)) \square Q \setminus b) \setminus Q \setminus b$$

we obtain that $\Theta(TS) = S$. We also have that $\Theta(TR) = R$. Hence we obtain from corollary 6.2.8 that $S \subseteq_t TS$ and $R \subseteq_t TR$. Then for timed media $TM1$ and $TM2$ we may define the timed alternating bit protocol as follows:

$$TABP \equiv \text{NET}((P, A_P) | P \in I)$$

where $I \equiv \{TS, TR, TM1, TM2\}$, and the $A_P$ are the same sets as for the corresponding untimed processes.

Now if $sr(X \text{ sat } \text{lm} \preceq \text{rm})(TM1)$ and $sr(X \text{ sat } \text{lc} \preceq \text{rc})(TM2)$, then

$$\exists M1 \subseteq_t TM1 \cdot M1 \text{ sat } \text{lm} \preceq \text{rm} \land \exists M2 \subseteq_t TM2 \cdot M2 \text{ sat } \text{lc} \preceq \text{rc}$$

Hence there is an $ABP$ such that $ABP \subseteq_t TABP$ and $ABP \text{ sat } \text{out} \preceq_t \text{in}$, so we obtain that $sr(X \text{ sat } \text{out} \preceq_t \text{in})(TABP)$, which is written

$$TABP \text{ sat } \text{out} \preceq_t \text{in}$$

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

$$TM1 \text{ sat } \text{lm} \preceq \text{rm}$$

$$TM2 \text{ sat } \text{lc} \preceq \text{rc}$$

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 input channels: that $TM1$ is responsive on $lm$, and $TM2$ is responsive 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 within 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:

$$TM1 \text{ sat } (s = u \cdot w \wedge w \uparrow rm = e \wedge \# w \uparrow lm \geq N)$$

$$\Rightarrow [begin(w), end(w)] \times rm \not\subseteq R$$

$$TM2 \text{ sat } (s = u \cdot w \wedge w \uparrow rc = e \wedge \# w \uparrow lc \geq N)$$

$$\Rightarrow [begin(w), end(w)] \times rc \not\subseteq R$$

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 $re$ in order to ensure eventual synchronisation. We will call the modified system $TABP2$.

Liveness of the $ABP$ is not guaranteed in $M_T$, 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 responsive.

The alternating bit protocol $TABP2$ may be described pictorially as follows:

![Diagram of TABP2](image)

We make the following new definitions, and redefine $AMS_C$

$$C \equiv COPY[mc, rc/in, out]$$
We then define

\[ \text{TABP}_2 \quad \triangleq \quad \text{NET}(\{(P, A_{P}) \mid P \in \{S, R, M_1, M_2, C\}\}) \]

\text{TABP}_2 is clearly impartial on \text{in}, and since it is responsive on \text{in} \cup \text{out}, and we have established that it satisfies \text{out} \preceq_{1} \text{in}, it follows from theorem 7.2.4 that it is a buffer, verifying the protocol. It follows as a refinement of \text{COPY}, state that \text{COPY} \subseteq_{1} \text{TABP}_2. It also follows from theorem 5.1.13 that \text{TABP}_2 is non-retracting on both \text{in} and \text{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 distributor and a message collator. Diagrammatically, the structure of the sliding window protocol is as follows:
This is defined in the process algebra as follows:

\[ Q_i \equiv i : (TABP2[in, out/i.send, i.pass]) \]

\[ A_{Q_i} \equiv i.send \cup i.pass \]

\[ DIS \equiv D_0 \]

\[ D_h \equiv in?x \rightarrow h.send!x \rightarrow D_{h@w.t} \]

\[ A_{DIS} \equiv \text{in} \cup \bigcup_{i=0}^{n} i.send \]

\[ COL \equiv COL_0 \]

\[ C_i \equiv l.pass?x \rightarrow out!x \rightarrow C_{l@w.t} \]

\[ A_{COL} \equiv \text{out} \cup \bigcup_{i=0}^{n} i.pass \]

\[ SWP \equiv NET\{(Q, A_Q) | Q \in \{Q_i | 0 \leq i \leq n\} \cup \{COL, DIS\}\} \]

We have from the previous section that \( COPY \subseteq_f TABP2 \), and that \( TABP2 \) is non-retracting on both \( \text{in} \) and \( \text{out} \), so

\[ i : (COPY[in, out/send, pass]) \subseteq_f i : (TABP2[in, out/send, pass]) \]

for any \( i \), and that \( i : (TABP2[in, out/send, pass]) \) is non-retracting on both \( i.send \) and \( i.receive \). It follows from corollary 6.3.24 that

\[ \Theta(COL) \subseteq_f COL \]

\[ \Theta(DIS) \subseteq_f DIS \]

and we also have from theorem 5.3.6 that \( COL \) and \( DIS \) are both prompt. Defining

\[ P_i \equiv i : (COPY[in, out/send, pass]) \]

\[ A_{P_i} \equiv A_{Q_i} \]

we define an untimed sliding window protocol \( SWP \) by

\[ SWP \equiv NET\{(P, A_P) | P \in \{P_i | 0 \leq i \leq n\} \cup \{\Theta(COL), \Theta(DIS)\}\} \]

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

\[ \text{PAR} \left( \{ (P, A_P) \mid P \in \{ P_i \mid 0 \leq i \leq n \} \} \cup \{ (\Theta(COL), \sigma(COL)), (\Theta(DIS), \sigma(DIS)) \} \right) \]

is limited on \( \bigcup_{i=0}^{n} (i.\text{send} \cup i.\text{pass}) \). We therefore obtain that

\[ SWP \subseteq_f TSWP \]

It is proved in [PS88] that \( SWP \) sat out \( \leq_{w+2} \) 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 by 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 number 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, such 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. Such 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. This 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 study, 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 constructors 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 under consideration. Our verification that the sliding window protocol described in chapter 8 met the specification $\text{out} \leq \text{in}$ was much easier in $M_T$ than it would have been in $TM_{FS}$, where any proof would have involved the manipulation of $(\text{trace}, \text{stability}, \text{refusal})$ 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 buffer 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 which 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 the 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 both safety and liveness properties; there now 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 TMFS or TMF.
Temporal logics consist of propositional atoms which are assigned truth values at every instant, together with temporal operators such as ‘eventually’ ($\Diamond$), ‘henceforth’ ($\square$), ‘unless’ ($U$) 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 $r$ holds of a state in which the system is ready to perform an $a$ action, then the assertion that $\Diamond r$ holds at a certain time of a system means that it will eventually be the 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

$$\square(r \Rightarrow rUs)$$

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 observable behaviours, because the denotational semantics of TCSP processes does not characterise state information. For example, the two processes

$$(a \rightarrow STOP) \sqcap \bot$$

and

$$(a \rightarrow STOP(\sqcap ((a \rightarrow STOP) \sqsubseteq \bot)) \sqcap \bot$$

have the same semantics, but they have different possible states. Using the temporal logic definition we would conclude that the 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 distinguish systems only by their observable events, by taking the propositional atoms of the temporal logic to be 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 unit. 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 that \(a\) will eventually be available.

Koymans argues this case in [Koy89], and proposes an extension to the language of temporal logic to include temporal operators which have explicit times associated with them. To do this, he requires that there be a metric function 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 \(F_{\delta}(\phi)\) asserts of its argument that there is a point exactly \(\delta\) in the future at which \(\phi\) will be true. \(F_{\leq\delta}(\phi)\) asserts that \(\phi\) will hold within \(\delta\). Its dual, \(G_{\leq\delta}(\phi)\) asserts that \(\phi\) will hold at all instants within \(\delta\).

The atomic propositions may again access state information, allowing the formulation of more operational style definitions. Responsiveness of \(P\) may be defined as follows:

\[
\Box(F_{\leq t}(r)) \text{ holds of } P
\]

where \(r\) holds of a state in 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:

\[
\Box(G_{\leq t}(\neg r) \Rightarrow (\neg r)\bigwedge s) \text{ holds of } P
\]

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 when the specification language is standard 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 (usually 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 with each transition a minimum and maximum length of time that the transition may be continuously enabled without firing. Hence a transition $t$ may fire at time $\tau$ 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 $\infty$.

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 $\sqsubseteq_r$-refinement.

Liveness properties concern the eventual 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 constitute 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:

\[ \text{Buff} = \text{in}(x).\text{out}(x).\text{Buff} \]

We may think of a verification that a process is a one place buffer as a demonstration that it is \textit{bisimilar} to \text{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 Sussex [Reg89] is investigating the consequences of the addition of a single \textit{WAIT} statement to \textit{EPL} [Hen88]. The treatment of time is discrete, and actions are instantaneous.

Synchronised \textit{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 \textit{CCS} processes may be ‘implemented’ by \textit{SCCS} ones. It also seems possible, for a given bisimilarity relation between \textit{CCS} processes, that there may be a weak bisimilarity relation (that ignores the passage of time) that holds between \textit{SCCS} processes precisely when the corresponding untimed \textit{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 \textit{ATP} [NRSV89] treats time rather differently. Actions are considered to take no time, except for an explicit synchronisation action \( \chi \) 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 \( \chi \) action. Successive instants are identified with successive occurrences of the \( \chi \) 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 \textit{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 \textit{LOTOS} [QF87] and Timed \textit{ACP} [BB89] are both process algebras which
allow continuous time. The syntax of Timed LOTOS allows the hidden action $\tau$ to be treated on the same footing as any visible action. There is also a delay operator $\text{WAIT } t$ (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 be written as Timed LOTOS processes, and verifications again consist of demonstrating a bisimilarity between a candidate process and a process we take to be the specification. Time-wise 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 refinement, unless a weak bisimulation relation is used.

Timed ACP [BB89] is an extension of ACP [BKl84] to include 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] that 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 above, 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 has that property. Any other properties the process may have may are irrelevant to the verification. On the other hand, a proof that two processes are bisimilar will establish that they share every property identified by the bisimilarity relation. Thus a verification often reduces to algebraic 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 be 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 $C[\cdot]$ and $D[\cdot]$ 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

$$B \gg Buff \approx Buff \gg B$$

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), $\diamond$ and $[\cdot]$. The specification $\langle \alpha \rangle G$ asserts of a process $P$ that it may perform $\alpha$ and reach a state of which $G$ holds. The specification $[\alpha]G$ asserts of $P$ that whenever $P$ performs $\alpha$ 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 unable 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 $\subseteq$ NUCLEAR is presented, from which it is concluded that NUCLEAR meets the safety requirement. Specifications expressible in the form $P \subseteq X$ (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 $X \text{ sat } (b \in \mathcal{E}[P])$, where $b$ ranges over behaviours, and $\mathcal{E}[P]$ denotes the semantics of $P$.

An alternative approach 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 which the 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

$$\text{send}(c) \text{ sat } c! U (c \text{ U}_{=1} \text{ done})$$

for the program $\text{send}(c)$. 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 that 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 the projection mappings between them, provide the foundation for timewise (and other) refinement relations. It would 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 equivalence (such as bisimilarity) may be obtainable, and would 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 TMF, yet the (infinite) semantics of a process will not necessarily be the set of all limits of the finite behaviours. In particular, if \( I_T[P] \) 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:

\[ \mathcal{I}_T\left(\bigcap_{i \in I} P(i)\right) = \bigcup_{i \in I} \mathcal{I}_T\left[P(i)\right] \]

\[ \mathcal{I}_T\left[P \setminus A\right] = \{(s \setminus A, \mathbb{R} \mid (s, \mathbb{R} \cup [0, \infty) \times A) \in \mathcal{I}_T\left[P\right]\} \]

\[ \mathcal{I}_T\left[f(P)\right] = \{(f(s), \mathbb{R}) \mid (s, f^{-1}(\mathbb{R}) \in \mathcal{I}_T\left[P\right]\} \]

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

\[ t \in [0, \infty) \Rightarrow \exists n(t) \in \mathbb{N}, \forall s \in \text{traces}(S) \cdot \text{end}(s) \leq t \Rightarrow \#s \leq n(t) \]

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:

\[ \forall t \in [0, \infty), s \in \text{traces}(S) \cdot \#(s \uparrow t) < \infty \]

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

\[ P = \bigcap_{n \in \mathbb{N}} \text{WAIT } n \; ; \; a \rightarrow \text{STOP} \]

\[ Q = P \cap \text{STOP} \]

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 \( \text{STOP} \), 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 \((\varnothing, [0, \infty) \times \{a\})\), 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 failures model will be useful for the construction of a temporal logic based specification language.

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 \(TM_P\). 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 require more rules for algebraically refining process descriptions, to enable us to work towards the timing requirements 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 up of the sender process could result in a timeout occurring 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 enable 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 been 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 establishing 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 \( \alpha, \delta, t, X, \mathbb{N}, \gamma \) be such that

\[ (\alpha \geq \delta \geq \text{end}(t) \land (t, \alpha, \mathbb{N} \cup ([0, \max\{\delta, \text{end}(\mathbb{N})\}] \times X)) \in \mathcal{E}_T[P] \]

Then define the following:

\[
\begin{align*}
  w_0 &= t \\
  w_{n+1} &= \begin{cases} 
    w_n & \text{if } B_n = 0 \\
    w_n \sim (t_{n+1}, a_{n+1}) & \text{if } B_n \neq 0 \\
    \text{where } (t_{n+1}, a_{n+1}) \in B_n
  \end{cases} \\
  t_0 &= \max\{\delta, \text{end}(\mathbb{N})\} \\
  t_{n+1} &= \begin{cases} 
    \text{some } \beta \text{ where } \exists a \bullet (\beta, a) \in B_n & \text{if } B_n \neq 0 \\
    \gamma & \text{if } B_n = 0
  \end{cases} \\
  B_n &= \{(\beta, a) \mid t_n \leq \beta \leq \gamma \land \\
  &\quad a \in X \land (w_n \sim (\beta, a), \mathbb{N} \cup ([0, \beta] \times X)) \in \text{fail}(\mathcal{E}_T[P])\}
\end{align*}
\]

Sublemma A.1.1 \( \forall n \bullet (w_n, \mathbb{N} \cup [0, t_n] \times X) \in \text{fail}(\mathcal{E}_T[P]) \),

We prove the sublemma inductively as follows:

**Base Case:** \( n = 0 \), the result follows from the definitions.

**Inductive Step:** Assume \( (w_n, \mathbb{N} \cup ([0, t_n] \times X)) \in \text{fail}(\mathcal{E}_T[P]) \).

**Case** \( B_n = \emptyset \)

\[ \vdash (w_n, \mathbb{N} \cup ([0, \gamma] \times X)) \in \text{fail}(\mathcal{E}_T[P]) \quad a_4 \]

\[ \vdash (w_{n+1}, \mathbb{N} \cup ([0, t_{n+1}] \times X)) \in \text{fail}(\mathcal{E}_T[P]) \quad \Box \]

**Case** \( B_n \neq \emptyset \)

Trivial

which establishes the sublemma. \( \square \)
We are now in a position to prove

**Lemma A.1.2**

\[ \forall \alpha, \delta, t, a, \mathcal{R}, \gamma \cdot \]
\[ (\alpha \geq \delta \geq \text{end}(t) \land (t, \alpha, \mathcal{R} \cup ([0, \text{max}\{\delta, \text{end}(\mathcal{R})\}] \times X)) \in \mathcal{E}_T[P] \]
\[ \Rightarrow \exists w, \alpha' \cdot w \setminus X = t \setminus X \land \alpha' \geq \delta \land (w, \alpha', \mathcal{R} \cup ([0, \gamma] \times X)) \in \mathcal{E}_T[P] \]

**Proof.**

Case

\[ \gamma \leq \text{max}\{\delta, \text{end}(\mathcal{R})\} \]
\[ \vdash \exists \alpha' \geq \alpha \cdot (t, \alpha', \mathcal{R} \cup ([0, \gamma] \times X)) \in \mathcal{E}_T[P] \quad a10 \]

Case

\[ \gamma > \text{max}\{\delta, \text{end}(\mathcal{R})\} \]
\[ \vdash \exists N \cdot \forall s \in \text{traces}(\mathcal{E}_T[P]) \cdot (\text{end}(s) \leq \gamma \Rightarrow \#s \leq N) \]
\[ \land (w_r = w_{r+1} \Rightarrow \forall k \geq r \cdot (w_k = w_{k+1}) \]
\[ \vdash w_{N+1} \neq w_N \Rightarrow \forall i \leq N \cdot (w_i \neq w_{i+1}) \]
\[ \land (w_i \neq w_{i+1} \Rightarrow \#w_{i+1} = \#w_i + 1) \quad \text{defn } w_i, t_i, B_i \]
\[ \vdash (w_{N+1} \neq w_N \Rightarrow \#w_{N+1} = \#w_0 + N + 1 \geq N + 1) \]

But \[ \text{end}(w_{N+1}) = t_{N+1} \leq \gamma \]
\[ \vdash \#w_{N+1} \leq N \quad \text{defn} \]
\[ \vdash w_{N+1} = w_N \]
\[ \vdash B_N = \emptyset \]
\[ \vdash t_{N+1} = \gamma \]
\[ \vdash \exists \alpha' \cdot (w_{N+1}, \alpha', \mathcal{R} \cup ([0, \gamma] \times X)) \in \mathcal{E}_T[P] \]

To complete the proof, it is sufficient to show that \( \alpha' \geq \delta \):

Case \[ w_0 = w_I \]
\[ \vdash w_{N+1} = w_0 \land B_0 = \emptyset \]
\[ \vdash \alpha' = \alpha \quad a11 \]
\[ \vdash \alpha' \geq \delta \]

Case \[ w_0 < w_I \]
\[ \vdash w_I = w_0 \cdot ((t_I, a)) \quad \text{defn} \]
\[ \vdash \text{end}(w_I) \geq t_I \geq t_0 \]
\[ \vdash \alpha' \geq \text{end}(w_{N+1}) \geq \text{end}(w_I) \geq t_0 \geq \delta \]

179
Corollary A.1.3 \((s, \mathcal{R}) \in \mathcal{F}_T[P \setminus A] \Rightarrow \forall t \exists w \bullet w \setminus A = s \land (w, \mathcal{R}[0, t) \times A) \in \mathcal{F}_T[P]\)

Lemma A.1.4 \(Z \subseteq (X \setminus Y) \Rightarrow (P \setminus Z) \|_Y Q = (P \|_Y Q) \setminus Z\)

Recall the definitions of hiding and of the parallel operator in \(TMFS\) given in [Ree88]:

\[
\begin{align*}
P \setminus X &= \sup \{ (s \setminus X, \beta, \mathcal{R}) \mid \exists \alpha \geq \beta \geq \text{end}(s) \bullet \\
&\quad (s, \alpha, \mathcal{R} \cup ([0, \max\{\beta, \text{end}(\mathcal{R})\}) \times X)) \in \mathcal{E}_T[P] \} \\
P \|_Y Q &= \sup \{ (s, \max\{\alpha_P, \alpha_Q\}, \mathcal{R}_P \cup \mathcal{R}_Q \cup \mathcal{R}_Z) \mid \exists (s_P, \alpha_P, \mathcal{R}_P) \in \mathcal{E}_T[P], \\
&\quad (s_Q, \alpha_Q, \mathcal{R}_Q) \in \mathcal{E}_T[Q] \bullet \alpha(\mathcal{R}_P) \subseteq X \land \alpha(\mathcal{R}_Q) \subseteq Y \\
&\quad \land s \in (s_P \|_Y s_Q) \land \alpha(\mathcal{R}_Z) \subseteq (\Sigma - (X \cup Y)) \}
\end{align*}
\]

where \(v \|_Y w = \{ s \in (T \Sigma)^Z \mid (s \uparrow (X \cup Y) = s) \land (s \uparrow X = v) \land (s \uparrow Y = w) \}\)

The following definitions will be useful:

\[
\begin{align*}
S(\alpha_P, \alpha_Q) &\leftrightarrow \exists \mathcal{R}_P, \mathcal{R}_Q \bullet \sigma(\mathcal{R}_P) \subseteq X \land \sigma(\mathcal{R}_Q) \subseteq Y \\
&\quad (t \uparrow X, \alpha_P, \mathcal{R}_P) \in \mathcal{E}_T[P] \\
&\quad \land (t \uparrow Y, \alpha_Q, \mathcal{R}_Q) \in \mathcal{E}_T[Q] \\
&\quad \land (\mathcal{R}_P \cup \mathcal{R}_Q) = (\mathcal{R} \cup ([0, \max\{\beta, \text{end}(\mathcal{R})\}) \times Z)) \uparrow X \cup Y
\end{align*}
\]

\[
\begin{align*}
D_{P,Q}(t, \beta, \mathcal{R}) &= \{ \max\{\alpha_P, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \}
\end{align*}
\]

\[
\begin{align*}
C_{P,Q}(s, \mathcal{R}) &= \{ \beta \mid \exists t \bullet t \setminus Z = s \\
&\quad \land \sup D_{P,Q}(t, \beta, \mathcal{R}) \geq \beta \geq \text{end}(t) \}
\end{align*}
\]

\[
\begin{align*}
S_P^Z(s, \mathcal{R}) &= \{ \beta \mid \exists t, \alpha \bullet \alpha \geq \beta \geq \text{end}(t) \land t \setminus Z = s \\
&\quad \land (t, \alpha, \mathcal{R} \cup ([0, \max\{\beta, \text{end}(\mathcal{R})\}) \times Z)) \in \mathcal{E}_T[P] \}
\end{align*}
\]

\[
\begin{align*}
T_{P,Q}(s, \mathcal{R}) &= \{ \max\{\alpha_P, \alpha_Q\} \mid \exists \mathcal{R}_P, \mathcal{R}_Q \bullet \alpha_P = \sup S_P^Z(s \uparrow X, \mathcal{R}_P) \\
&\quad \land \alpha_P \geq 0 \\
&\quad \land (s \uparrow Y, \alpha_Q, \mathcal{R}_Q) \in \mathcal{E}_T[Q] \\
&\quad \land \mathcal{R}_P \cup \mathcal{R}_Q = \mathcal{R} \cup X \cup Y \\
&\quad \land \sigma(\mathcal{R}_P) \subseteq X \land \sigma(\mathcal{R}_Q) \subseteq Y \}
\end{align*}
\]

180
The definitions of these sets allow us to obtain the following equivalences (adopting the convention that $\sup \emptyset = -\infty$):

\[
\begin{align*}
(s, \alpha, \mathcal{R}) \in \mathcal{E}_T[(P \setminus Z)] & \iff \alpha = \sup \beta_T(s, \mathcal{R}) \\
(s, \alpha, \mathcal{R}) \in \mathcal{E}_T[(P \setminus Z) \parallel Y] & \iff \alpha = \sup \beta_T(s, \mathcal{R}) \\
(s, \alpha, \mathcal{R}) \in \mathcal{E}_T[(P \parallel Y) \setminus Z] & \iff \alpha = \sup \beta_T(s, \mathcal{R})
\end{align*}
\]

Hence it is sufficient to prove that, for arbitrary $s, \mathcal{R}$, and any processes $P$ and $Q$, that $\sup \beta_T(s, \mathcal{R}) = \sup \beta_T(s, \mathcal{R})$. This will be done by first showing that

\[
\sup \beta_T(s, \mathcal{R}) \geq \sup \beta_T(s, \mathcal{R})
\]

and then by showing that

\[
\sup \beta_T(s, \mathcal{R}) \geq \sup \beta_T(s, \mathcal{R})
\]

We first aim to show that $\sup \beta_T(s, \mathcal{R}) \geq \sup \beta_T(s, \mathcal{R})$:

Assume $\beta_T(s, \mathcal{R}) \neq \emptyset$. Then let $\beta \in \beta_T(s, \mathcal{R})$. Take $\delta \in \mathcal{D}_{P, Q}(t, \beta, \mathcal{R})$, where $t \setminus Z = s$. Then we have

\[
\begin{align*}
(t \uparrow X, \alpha, \mathcal{R}) & \in \mathcal{E}_T[P] \land \sigma(\mathcal{R}) \subseteq X \\
(t \uparrow Y, \alpha, \mathcal{R}) & \in \mathcal{E}_T[Q] \land \sigma(\mathcal{R}) \subseteq Y \\
(\mathcal{R} \cup \mathcal{R}) & = (\mathcal{R} \cup ([0, \max\{\beta, \text{end}(\mathcal{R})) \times Z)) \uparrow X \cup Y
\end{align*}
\]

for some $\alpha, \mathcal{R}, \alpha, \mathcal{Q}$ such that $\max\{\alpha, \mathcal{Q}\} = \delta$. Let $\mathcal{R} = \mathcal{R} \setminus ([0, \infty) \times Z)$. Then

\[
\begin{align*}
(t \uparrow X) \setminus Z & = s \uparrow X \\
(t \uparrow Y, \alpha, \mathcal{R} \cup ([0, \max\{\beta, \text{end}(\mathcal{R})) \times Z)) & \in \mathcal{E}_T[P] \\
(t \uparrow Y, \alpha, \mathcal{R}) & \in \mathcal{E}_T[Q] \\
\mathcal{R} \cup \mathcal{R} & = \mathcal{R} \uparrow X \cup Y \\
\alpha & \geq \text{end}(t \uparrow X)
\end{align*}
\]

Case $\beta > \alpha$

$\vdash (t \uparrow X, \alpha, \mathcal{R} \cup ([0, \max\{\alpha, \text{end}(\mathcal{R})) \times Z)) \in \mathcal{E}_T[P]$

$\vdash \alpha \in \beta_T(s \uparrow X, \mathcal{R})$

Case $\beta \leq \alpha$

$\vdash \beta \in \beta_T(s \uparrow X, \mathcal{R})$

since $\beta \geq \text{end}(t) \geq \text{end}(t \uparrow X)$
So we deduce that $\min\{\beta, \alpha_p\} \in S^k_x(s \uparrow X, \mathcal{R}_P)$.

We have so far established

$$
\sup D_{P,Q}(t, \beta, \mathcal{R}) \geq \beta \\
\wedge S(\alpha_P, \alpha_Q) \Rightarrow (\exists \mathcal{R}_P \bullet \min\{\beta, \alpha_p\} \in S^k_x(s \uparrow X, \mathcal{R}_P) \\
\vdash \sup T_{P,Q}(s, \mathcal{R}) \geq \sup \left\{ \max\{\min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \right\}
$$

We again have two cases to consider.

**Case**

$$
\forall \alpha_P, \alpha_Q \bullet (S(\alpha_P, \alpha_Q) \Rightarrow \beta \geq \alpha_P) \\
\vdash \{ \max\{\min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \} = \{ \max\{\alpha_P, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \} \\
\vdash \sup \left\{ \max\{\min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \right\} = \sup D_{P,Q}(t, \beta, \mathcal{R}) \geq \beta
$$

**Case**

$$
\exists \alpha_P, \alpha_Q \bullet S(\alpha_P, \alpha_Q) \wedge \alpha_P \geq \beta \\
\vdash \sup \left\{ \max\{\min\{\alpha_P, \beta\}, \alpha_Q\} \mid S(\alpha_P, \alpha_Q) \right\} \geq \beta
$$

In either case we conclude that $\sup T_{P,Q}(s, \mathcal{R}) \geq \beta$

(And that $T_{P,Q}(s, \mathcal{R}) \neq \emptyset$).

We now aim to show that $\sup C_{P,Q}(s, \mathcal{R}) \geq \sup T_{P,Q}(s, \mathcal{R})$:

Assume $T_{P,Q}(s, \mathcal{R}) \neq \emptyset$. Then $\sup T_{P,Q}(s, \mathcal{R}) = \sup U_{P,Q}(s, \mathcal{R})$.

Let $\beta \in U_{P,Q}(s, \mathcal{R})$

$$
\vdash \beta = \max\{\delta, \alpha_Q\} \\
\wedge \alpha \geq \delta \geq \text{end}(t) \\
\wedge t \uparrow Z = s \uparrow X \\
\wedge (t, \alpha, \mathcal{R}_P \cup ([0, \max\{\delta, \text{end}(\mathcal{R}_P)\}) \times Z)) \in \mathcal{E}_T[P] \\
\wedge (s \uparrow Y, \alpha_Q, \mathcal{R}_Q) \in \mathcal{E}_T[Q] \\
\wedge \sigma(\mathcal{R}_P) \subseteq X \wedge \sigma(\mathcal{R}_Q) \subseteq Y \\
\wedge \mathcal{R}_P \cup \mathcal{R}_Q = \mathcal{R} \uparrow X \cup Y \\
\text{for some } \delta, \alpha, \alpha_Q, \mathcal{R}_P, \mathcal{R}_Q, t
$$

$$
\vdash \beta = \max\{\delta, \alpha_Q\} \\
\wedge w \uparrow X = t \uparrow X \\
\wedge \alpha' \geq \delta \\
\wedge (w, \alpha', \mathcal{R}_P \cup [0, \max\{\beta, \text{end}(\mathcal{R})\}) \in \mathcal{E}_T[P] \\
\wedge t \uparrow X = s \uparrow X \\
\wedge (s \uparrow Y, \alpha_Q, \mathcal{R}_Q) \in \mathcal{E}_T[Q] \\
\wedge \mathcal{R}_P \cup \mathcal{R}_Q = \mathcal{R} \uparrow X \cup Y \text{ for some } w, \alpha' \\
$$

(Lemma A.1.2)
Let $u \in (w_X \|_Y (s \uparrow Y))$

\[
\vdash (u \uparrow \alpha_p, \alpha', \mathbb{R}_p) \in \mathcal{E}_T [P] \\
\wedge (u \uparrow \alpha_q, \alpha_q, \mathbb{R}_q) \in \mathcal{E}_T [Q] \\
\wedge \mathbb{R}_p \cup \mathbb{R}_q = (\mathbb{R} \cup \emptyset, \max\{\beta, \text{end}(\mathbb{R})\}) \times Z \uparrow X \cup Y
\]

(Defining $\mathbb{R}_p = \mathbb{R}_p \cup \emptyset, \max\{\beta, \text{end}(\mathbb{R})\}) \times Z$)

Now $\text{end}(w) \leq \max\{\beta, \text{end}(\mathbb{R})\}$

\[
\vdash \text{end}(u \uparrow X) \leq \max\{\beta, \text{end}(\mathbb{R})\}
\]

Also $\text{end}(u \uparrow Y) \leq \beta \leq \max\{\beta, \text{end}(\mathbb{R})\}$

There are two cases to consider here:

**Case**

$max\{\alpha', \alpha_q\} \leq \text{end}(\mathbb{R})$

\[
\vdash D_{P,Q}(t, \beta, \mathbb{R}) = D_{P,Q}(t, \max\{\alpha', \alpha_q\}, \mathbb{R})
\]

$\vdash \max\{\alpha', \alpha_q\} \in C_{P,Q}(s, \mathbb{R})$

(since $\text{end}(u) \leq \max\{\alpha', \alpha_q\}$)

**Case**

$max\{\alpha', \alpha_q\} > \text{end}(\mathbb{R})$

**subcase** $\beta \geq \text{end}(u)$

$\vdash \beta \in C_{P,Q}(s, \mathbb{R})$

**subcase** $\beta < \text{end}(u)$

$\vdash \max\{\beta, \text{end}(\mathbb{R})\} = \text{end}(\mathbb{R}) \geq \text{end}(u)$

(since $\text{end}(u \uparrow X) \leq \max\{\beta, \text{end}(\mathbb{R})\}$, $\text{end}(u \uparrow Y) \leq \beta$, so $\beta < \text{end}(u) \leq \max\{\beta, \text{end}(\mathbb{R})\}$)

$\vdash \alpha' \geq \text{end}(\mathbb{R}) \geq \delta$

\[\wedge \alpha' \geq \text{end}(\mathbb{R}) \geq \alpha_q\]

$\vdash \max\{\alpha', \alpha_q\} \in D_{P,Q}(u, \text{end}(\mathbb{R}), \mathbb{R})$

$\wedge \max\{\alpha', \alpha_q\} \geq \text{end}(u)$

$\vdash \max\{\alpha', \alpha_q\} \in C_{P,Q}(s, \mathbb{R})$

Also $\max\{\alpha', \alpha_q\} \geq \text{end}(u) > \beta$.

Therefore $\sup C_{P,Q}(s, \mathbb{R}) \geq \sup U_{P,Q}(s, \mathbb{R}) = \sup T_{P,Q}(s, \mathbb{R}) \quad \square$

**Corollary A.1.5** $Z \cap Y = \emptyset \Rightarrow (P \setminus Z) \|_Y Q = (P \cup Z \|_Y Q) \setminus Z$

183
B Semantic Models and Mappings

B.1 Reed's Hierarchy

The semantic models are arranged in a hierarchical structure as follows:

\[ \begin{align*}
    TM_{FS} \\
    \downarrow & \\
    M_{FS} & \rightarrow TM_{FS}^* \\
    \downarrow & \\
    TM_F & \rightarrow M_F \quad M_S \rightarrow TM_S \\
    \downarrow & \\
    M_T & \\
    \quad \uparrow & \\
    TM_T
\end{align*} \]

Some Projection Mappings

\( \Pi : TM_{FS} \rightarrow TM_F \) is defined on processes \( Q \in TM_{FS} \) by

\[ \Pi(Q) = \text{fail}(Q) \]

The projection mapping \( \Pi : TM_{FS} \rightarrow TM_{FS}^* \) may be expressed using our notion of \( s \) reflected in \( \mathcal{N} \). It is defined as follows:
Let \( P \in T M_{FS} \). For each \( s \in T \Sigma^* \), let
\[
P[s] = \{(\bar{s}, \beta, \kappa) \in P \mid \text{s is reflected in } \kappa\}
\]
\[
\Pi(P) = \{(s, \alpha, X) \in T \bar{\Sigma}^* \times TSTAB \times P(\Sigma) \mid P[s] \neq \emptyset \land \alpha = \sup\{\beta \mid (\bar{s}, \beta, \kappa) \in P[s]\} \land (\alpha < \infty \land \exists(\bar{s}, \beta, \kappa) \in P[s] \mid X = \sigma(\kappa \uparrow \alpha) \lor \alpha = \infty \land X \in P(\Sigma))\}
\]
\[
\Pi : T M_{FS} \rightarrow M_F \text{ is defined on processes } Q \in T M_{FS} \text{ by}
\]
\[
\Pi(P) = \{(tstrip(s), X) \mid (s, \alpha, \kappa) \in P \land \exists(s, \kappa) \in fail(P) \land \alpha \leq \text{begin}(\kappa) < \infty \land \sigma(\kappa) = X \lor \alpha = \infty \land X \in P(\Sigma)\}
\]

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 \( \Sigma^* \) satisfying:

1. \( \emptyset \in S \)
2. \( s \omega \in S \Rightarrow s \in S \).

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

$$S(n) \doteq \{ s \in S \mid \#s \leq n \}.$$  

The complete metric on $M_T$ is defined:

$$d(S_1, S_2) \doteq \inf \{ 2^{-n} \mid S_1(n) = S_2(n) \}.$$  

The Semantic Function $T$.

We now define a semantic function $T : CSP \rightarrow M_T$.

\begin{align*}
T[\bot] & \doteq \{ \} \\
T[STOP] & \doteq \{ \} \\
T[SKIP] & \doteq \{ (), (\lor) \} \\
T[a \rightarrow P] & \doteq \{ () \} \cup \{ (a) \cap s \mid s \in T[P] \} \\
T[P \cap Q] & \doteq T[P] \cup T[Q] \\
T[P \cap Q] & \doteq T[P] \cup T[Q] \\
T[P \parallel Q] & \doteq T[P] \cap T[Q] \\
T[P \chi_Y Q] & \doteq \{ s \mid s \uparrow X \in T[P] \land s \uparrow Y \in T[Q] \\
& \quad \land s \uparrow (X \cup Y) = s \} \\
T[P ||| Q] & \doteq \{ s \mid \exists u \in T[P], v \in T[Q] : s \in \text{Merge}(u, v) \} \\
T[P; Q] & \doteq \{ s \mid s \in T[P] \land \lor \notin s \} \\
& \quad \cup \{ s \cap t \mid s \cap (\lor) \in T[P] \land t \in T[Q] \} \\
T[P \setminus X] & \doteq \{ s \setminus X \mid s \in T[P] \} \\
T[f^{-1}(P)] & \doteq \{ s \mid f(s) \in T[P] \} \\
T[\mu P.F(P)] & \doteq \text{The least fixed point of the continuous mapping with respect to the complete partial order } \sqsubseteq \text{ on } M_T \text{ represented by } F. \\
or \quad T[\mu P.F(P)] & \doteq \text{The unique fixed point of the contraction mapping with respect to the complete metric } d \text{ on } M_T \text{ represented by } F.
\end{align*}
We may also add the equations

\[ T[a : A \rightarrow P(a)] \equiv \{()\} \cup \{(a)^s | a \in A \land s \in T[P(a)]\} \]

\[ T[\bigcap P(a)] \equiv \bigcup T[P(a)] \]

The failures model

The Evaluation Domain \( M_F \).

We formally define \( M_F \) to be those subsets \( S \) of \( \Sigma^* \times P(\Sigma) \), satisfying:

1. \( () \in \text{Traces}(S) \)
2. \( s \cap w \in \text{Traces}(S) \Rightarrow s \in \text{Traces}(S) \)
3. \( (s, X) \in S \land Y \subseteq X \Rightarrow (s, Y) \in S \)
4. \( (s, X) \in S \land (\forall a \in Y, ((s, (a), \emptyset) \notin S)) \Rightarrow (s, X \cup Y) \in S \)

The Complete Metric on \( M_F \).

If \( S \in M_F \), we define

\[ S(n) \equiv \{(s, X) \in S \mid \#s < n\} \]

\[ \cup \{(s, X) \mid \#s = n \land s \in \text{traces}(S)\}. \]

The complete metric on \( M_F \) is defined:

\[ d(S_1, S_2) \equiv \inf \{2^{-n} \mid S_1(n) = S_2(n)\} \]

The Semantic Function \( \mathcal{F} \).

We now define the semantic function \( \mathcal{F} : CSP \rightarrow M_F \).

\[ \mathcal{F}[\bot] \equiv \{(), X) \mid X \in P(\Sigma)\} \]

\[ \mathcal{F}[\text{STOP}] \equiv \{(), X) \mid X \in P(\Sigma)\} \]

\[ \mathcal{F}[\text{SKIP}] \equiv \{(), X) \mid \forall \notin X\} \cup \{((\forall), X) \mid X \in P(\Sigma)\} \]

187
\[ F[a \to P] \equiv \{((),X) \mid a \notin X\} \]
\[ \cup \{((a)^{\sim}s,X) \mid (s,X) \in F[P]\} \]
\[ F[P^\square Q] \equiv \{(((),X) \mid (((),X) \in F[P] \cap F[Q]\}
\[ \cup \{(s,X) \in F[P] \cup F[Q] \mid s \neq ()\} \]
\[ F[P \cap Q] \equiv F[P] \cup F[Q] \]
\[ F[P \parallel Q] \equiv \{(s,X \cup Y) \mid (s,X) \in F[P] \]
\[ \wedge (s,Y) \in F[Q]\} \]
\[ F[P \times Y Q] \equiv \{(s,Z_p \cup Z_Q \cup Z) \mid s \uparrow (X \cup Y) = s \wedge \]
\[ Z_p \subseteq X \wedge Z_Q \subseteq Y \wedge Z \subseteq (\Sigma - (X \cup Y)) \]
\[ \wedge (s \uparrow X,Z_p) \in F[P] \]
\[ \wedge (s \uparrow Y,Z_Q) \in F[Q]\} \]
\[ F[P \\ Q] \equiv \{(s,X) \mid \exists u,v \in \Sigma^* \cdot (u,X) \in F[P]\]
\[ \wedge (v,X) \in F[Q] \wedge s \in \text{Merge}(u,v)\} \]
\[ F[P;Q] \equiv \{(s,X) \mid \nabla s \wedge (s,X \cup \{\nabla\}) \in F[P]\}
\[ \cup \{(s^\sim w,X) \mid \nabla s \wedge (s^\sim \nabla),\emptyset) \in F[P]\]
\[ \wedge (w,X) \in F[Q]\} \]
\[ F[P \setminus X] \equiv \{s \setminus X, Y) \mid (s,X \cup Y) \in F[P]\}
\[ \cup \{(s,Y) \mid \forall n \geq \#s, \exists w_n \in \text{Traces}(F[P]) \cdot \]
\[ w_n < w_{n+1} \wedge s = w_n \setminus X\} \]
\[ F[f^{-1}(P)] \equiv \{(s,X) \mid (f(s),f(X)) \in F[P]\} \]
\[ F[f(P)] \equiv \{(f(s),X) \mid (s,f^{-1}(X)) \in F[P]\} \]
\[ F[\mu P.F(P)] \equiv \text{The unique fixed point of the contraction mapping}
\[ C(Q) = C(WAIT \delta ; Q), \text{where } C \text{ is the mapping on}
\[ M_F \text{ represented by } F. \]

We may also add the equations
\[ F[a : A \to P(a)] \equiv \{(((),X) \mid A \cap X = \emptyset\}
\[ \cup \{((a)^{\sim}s,X) \mid a \in A \wedge (s,X) \in F[P]\} \]
\[ F[\bigcap P(a)] \equiv \bigcup F[P(a)] \]

188
The (Untimed Failures)-(Timed Stability) Model

The Evaluation Domain $TMFS$

We formally define $TMFS$ to be those subsets $S$ of $TΣx[0,∞] × Σ$ satisfying:

1. $\emptyset \in \text{traces}(S)$
2. $s^w \in \text{traces}(S) \Rightarrow s \in \text{traces}(S)$
3. $(s, α, X) ∈ S \Rightarrow (s, α, X) ∈ S$
4. $(s, α, X) ∈ S \land s \equiv w \Rightarrow (w, α, X) ∈ S$
5. $s^w((t, a)) \in \text{traces}(S) \Rightarrow \exists t' ≤ t \bullet (s^t \bullet (t', a)) \in \text{traces}(S) \land$
   $(t' ≤ t'' < t \Rightarrow (s^t \bullet (t'', a)) \in \text{traces}(S))$
6. $∀ t ∈ [0, ∞), \exists n(t) ∈ N$ such that $∀ s ∈ \text{traces}(S), (\text{end}(s) ≤ t \Rightarrow \#s ≤ n(t))$
7. $(s, α), (s, α') ∈ \text{stab}(S) \Rightarrow α = α'$
8. $(s, α) ∈ \text{stab}(S) \Rightarrow \text{end}(s) ≤ α$
9. $(s, α) ∈ \text{stab}(S) \land s^w((t, a)) \in \text{traces}(S) \Rightarrow t ≤ α$
10. $(s, α) ∈ \text{stab}(S) \Rightarrow \text{if } t > α, t' ≥ α, a ∈ Σ \text{ and}$
    
    $w ∈ TΣx[0,∞]$ is such that $w = ((t, a))^w'$,
    
    then $(s^w, α', X) ∈ S \Rightarrow (s^w + (t' - t), γ, X) ∈ S,$
    
    where $γ ≥ α' + (t' - t)$$
11. $(s, α, X) ∈ S \land Y ⊆ X \Rightarrow (s, α, Y) ∈ S$
12. $(s, α, X) ∈ S \land \exists Y ∈ P(Σ) \bullet (\forall a ∈ Y,$
    
    $\exists t ≥ α \bullet (s^w((t, a)), 0) \notin \text{fail}(S)) \Rightarrow (s, α, X \cup Y) ∈ S$
13. $(s, ∞) ∈ \text{stab}(S) \land X ∈ P(Σ) \Rightarrow (s, ∞, X) ∈ S$

The Complete Metric on $TMFS$

If $S ∈ TMFS$ and $t ∈ [0, ∞)$, we define

$$S(t) ≜ \{(s, α, X) ∈ S | α < t\} \cup \{(s, ∞, X) | \text{end}(s) < t \land \exists α ≥ t \text{ such that } (s, α) ∈ \text{stab}(S) \land X ∈ P(Σ)\}.$$
The complete metric on $TM^*_S$ is defined:

$$d(S_t, S_{t'}) \equiv \inf \{2^{-t} \mid S_t(t) = S_{t'}(t)\}$$

The Semantic Function $\mathcal{E}_T^*$

We now define the semantic function $\mathcal{E}_T^* : TCSP \rightarrow TM^*_S$.

$$\mathcal{E}_T^*[\bot] \equiv \{((t), \infty, X) \mid X \in \mathcal{P}(\Sigma)\}$$

$$\mathcal{E}_T^*[STOP] \equiv \{((t), 0, X) \mid X \in \mathcal{P}(\Sigma)\}$$

$$\mathcal{E}_T^*[SKIP] \equiv \{((t), 0, X) \mid \not\exists X\}$$

$$\cup\{(\{(0, \not\exists\}), 0, X) \mid X \in \mathcal{P}(\Sigma)\}$$

$$\cup\{(((t, \not\exists)), t, X) \mid t \geq 0 \land \not\exists X\}$$

$$\mathcal{E}_T^*[WAIT t] \equiv \{((t), t, X) \mid \not\exists X\}$$

$$\cup\{(((t, \not\exists)), t, X) \mid X \in \mathcal{P}(\Sigma)\}$$

$$\cup\{(((t', \not\exists)), t', X) \mid t' \geq t \land X \in \mathcal{P}(\Sigma)\}$$

$$\mathcal{E}_T^*[a \rightarrow P] \equiv \{((t), 0, X) \mid a \not\in X\}$$

$$\cup\{(((0, \not\exists)), (s + \delta, \alpha + \delta, X) \mid (s, \alpha, X) \in \mathcal{E}_T^*[P]\}$$

$$\cup\{(((t, \not\exists)), (s + (t + \delta), \alpha + t + \delta, X) \mid t \geq 0$$

$$\land (s, \alpha, X) \in \mathcal{E}_T^*[P]\}$$

$$\mathcal{E}_T^*[P \sqcap Q] \equiv SUP(\{((t), \alpha, X) \mid ((t), \alpha) \in \text{stab}(\mathcal{E}_T^*[P] \cup \mathcal{E}_T^*[Q])$$

$$\land ((t), X) \in \text{fail}(\mathcal{E}_T^*[P]) \cap \text{fail}(\mathcal{E}_T^*[Q])\}$$

$$\cup\{(s, \alpha, X) \in \mathcal{E}_T^*[P] \cup \mathcal{E}_T^*[Q] \mid s \neq (t)\}$$

$$\mathcal{E}_T^*[P \sqcup Q] \equiv SUP(\mathcal{E}_T^*[P] \cup \mathcal{E}_T^*[Q])$$

$$\mathcal{E}_T^*[P \parallel Q] \equiv SUP(\{(s, \max\{\alpha_P, \alpha_Q\}, X_P \cup X_Q) \mid (s_P, \alpha_P, X_P) \in \mathcal{E}_T^*[P]$$

$$\land (s_Q, \alpha_Q, X_Q) \in \mathcal{E}_T^*[Q] \land \delta_P = \delta_Q\})$$

$$\land (s, \alpha, X) \in \mathcal{E}_T^*[P] \land (s, \alpha, X) \in \mathcal{E}_T^*[Q] \land s \in Tmerge(u, v))$$

$$\mathcal{E}_T^*[P; Q] \equiv CL(\mathcal{E}_T^*[P] \land \not\exists (\sigma(s))$$

$$\cup\{(s \sim (w + t), \alpha + t, X) \mid s \sim ((t, \not\exists)) \in \text{traces}(\mathcal{E}_T^*[P])$$

$$\land \not\exists \sigma(s) \land (w, \alpha, X) \in \mathcal{E}_T^*[Q]\})}$$

Note: The above definitions involve complex expressions that include constructs such as $\cup$, $\land$, $\not\exists$, $\land$, and $\max\{\alpha_P, \alpha_Q\}$, which are typical in formal semantics and temporal logic to describe the behavior of systems under different conditions.
\[ \mathcal{E}^{*}_T[P \setminus X] \cong \{ (s \setminus X, \alpha, Y) \mid s \text{ is } X\text{-active in } \mathcal{E}^{*}_T[P] \land \alpha = \sup \{ \beta \mid \exists (w, \beta) \in \mathcal{E}^{*}_T[P] \} \cdot w \text{ is } X\text{-active} \land w \setminus X = s \setminus X \land (\alpha < \infty \land (s, X \cup Y) \in \text{fail}(\mathcal{E}^{*}_T[P]) \lor (\alpha = \infty \land Y \in \mathcal{P}(\Sigma)) \} \]

where \( s \) is \( X \)-active provided \( s \) contains no element of the form \((t, a)\) for \( a \in X \) (all communications in \( X \) are in the form \( \dot{a} \)).

\[ \mathcal{E}^{*}_T[f^{-1}(P)] \cong \{ (s, \alpha, X) \mid (f(s), \alpha, f(X)) \in \mathcal{E}^{*}_T[P] \} \]

\[ \mathcal{E}^{*}_T[f(P)] \cong \text{SUP}(\{(f(s), \alpha, X) \mid (s, \alpha, f^{-1}(X)) \in \mathcal{E}^{*}_T[P]\}) \]

\[ \mathcal{E}^{*}_T[\mu P.F(P)] \cong \text{The unique fixed point of the contraction mapping } \hat{C}(Q) = C(WAIT \delta; Q), \text{ where } C \text{ is the mapping on } TM^{*}_T \text{ represented by } F. \]

We may add the following equations

\[ \mathcal{E}^{*}_T[\bigcap_{a \in A} P(a)] \cong \text{SUP}(\bigcup_{a \in A} \mathcal{E}^{*}_T[P(a)]) \]

\[ \mathcal{E}^{*}_T[a : A \rightarrow P(a)] \cong \{ (\emptyset, 0, X) \mid A \cap X = \emptyset \}

\cup \{ (((0, \dot{a})) \cap (s + \delta), \alpha + \delta, X) \mid a \in A \land (s, \alpha, X) \in \mathcal{E}^{*}_T[P(a)] \} \]

\cup \{ (((t, a)) \cap (s + t + \delta), \alpha + t + \delta, X) \mid a \in A \land t \geq 0 \land (s, \alpha, X) \in \mathcal{E}^{*}_T[P(a)] \} \]

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 $T\Sigma_\leq \times RSET$ satisfying:

1. $\emptyset \in Traces(S)$
2. $(s \triangleleft w, R) \in S \Rightarrow (s, R \uparrow begin(w)) \in S$
3. $(s, R) \in S \land s \equiv w \Rightarrow (w, R) \in S$
4. $(s, R) \in fail(S) \Rightarrow \exists N' \in RSET \bullet \\
   \land t \geq 0 \Rightarrow R \subseteq N' \land (s, N') \in fail(S) \land \\
   (t' \leq t \land (t', a) \notin N') \Rightarrow (s \uparrow t' \triangleleft ((t', a)), N' \uparrow t') \in fail(S)$
5. $\forall t \in [0, \infty), \exists n(t) \in \mathbb{N} \bullet \forall s \in traces(S), (end(s) \leq t \Rightarrow \#s \leq n(t))$
6. $(s, R) \in S \land R' \in RSET \land R' \supseteq R \Rightarrow (s, R') \in S$
7. $(s \triangleleft w, R) \in S \land R' \in RSET \bullet end(s) \leq begin(R') \land end(R') \leq begin(w) \land \\
   (\forall (t, a) \in R', (s, ((t, a)), R \uparrow t) \notin S) \Rightarrow (s, R \cup R') \in S$

The Complete Metric on $TM_F$.

If $S \in TM_F$ and $t \in [0, \infty)$, we define

$$S(t) \equiv \{(s, R) \in S \mid end(s) < t \land end(R) < t\}.$$

The complete metric on $TM_F$ is defined:

$$d(S_1, S_2) \equiv \inf\{2^{-t} \mid S_1(t) = S_2(t)\}$$

The Semantic Function $\mathcal{F}_T$.

We now define the semantic function $\mathcal{F}_T : TCSP \rightarrow TM_F$. 

192
\[ \mathcal{F}_T[\bot] = \{ (\emptyset, N) \mid N \in \mathbb{R} \} \]
\[ \mathcal{F}_T[\text{STOP}] = \{ (\emptyset, N) \mid N \in \mathbb{R} \} \]
\[ \mathcal{F}_T[\text{SKIP}] = \{ (\emptyset, N) \mid \emptyset \not\in \sigma(N) \} \]
\[ \mathcal{F}_T[\text{WAIT} t] = \{ (\emptyset, N) \mid N \in \{ t, \infty \} \} \]
\[ \mathcal{F}_T[a \rightarrow P] = \{ (\emptyset, N) \mid a \not\in \sigma(N) \} \]
\[ \mathcal{F}_T[P ; Q] = \{ (s, N) \mid \exists (s, R) \in \mathcal{F}_T[P] \land (s, R) \in \mathcal{F}_T[Q] \} \]
\[ \mathcal{F}_T[P \setminus X] = \{ (s \setminus X, N) \mid (s, N) \in \mathcal{F}_T[P] \} \]
The unique fixed point of the contraction mapping

\[ \hat{C}(Q) = C(WAIT_\delta; Q), \]

where \( C \) is the mapping on \( T \mathcal{M}_F \) represented by \( F \).

We may also add the following equations:

\[
\mathcal{F}_T\left[\bigcap_{a \in A} P(a)\right] \equiv \bigcup_{a \in A} \mathcal{F}_T\left[P(a)\right]
\]

\[
\mathcal{F}_T\left[a : A \rightarrow P_\ast\right] \equiv \{((\ell, \mathcal{R}) | A \cap \sigma(\mathcal{R}) = \emptyset) \cup\{(\ell, \mathcal{R}) | A \cap \sigma(\mathcal{R} \cap t) = \emptyset \}
\]

provided the set of processes \( \{P(a) | a \in A\} \) is uniformly bounded.

The timed failures stability model

The Evaluation Domain \( T \mathcal{M}_F \).

We formally define \( T \mathcal{M}_F \) to be those subsets \( S \) of \( (T \Sigma)_\xi^\ast \times [0, \infty) \times RSET \) satisfying the following 12 axioms:

1. \( \langle \rangle \in \text{Traces}(S) \)

2. \( (s, w, \mathcal{R}) \in \text{fail}(S) \Rightarrow \langle s, \mathcal{R} \uparrow \text{begin}(w) \rangle \in \text{fail}(S) \)

3. \( (s, \alpha, \mathcal{R}) \in S \land s \equiv w \Rightarrow \langle w, \alpha, \mathcal{R} \rangle \in S \)

4. \( (s, \mathcal{R}) \in \text{fail}(S) \Rightarrow \exists \mathcal{R}' \in RSET \bullet \frac{\mathcal{R} \subseteq \mathcal{R}' \land (s, \mathcal{R}') \in \text{fail}(S) \land (t' \leq t \land (t', a) \notin \mathcal{R}') \Rightarrow (s \uparrow t' \cap (t', a), \mathcal{R}' \cap t') \in \text{fail}(S)}{t \in [0, \infty) \Rightarrow \exists n(t) \in N \text{ such that } \forall s \in \text{Traces}(S), \end(s) \leq t \Rightarrow \#s \leq n(t)} \)

5. \( t \in [0, \infty) \Rightarrow \exists n(t) \in N \) such that \( \forall s \in \text{Traces}(S), \)

6. \( (s, \alpha, \mathcal{R}), (s, \beta, \mathcal{R}) \in S \Rightarrow \alpha = \beta \)
7. \((s, \alpha, R) \in S \Rightarrow end(s) \leq \alpha\)

8. \((s, \alpha, R) \in S \wedge (s, ((t, a)), R) \in fail(S) \wedge t > t' \geq \alpha \wedge t \geq end(R) \Rightarrow (t', a) \notin R\)

9. \((s, \alpha, R) \in S \Rightarrow if t > \alpha, t' \geq \alpha, a \in \Sigma and w \in (\Sigma^\infty)^2 is such that w = ((t, a)).w', then (s.w, \alpha', R') \in S \wedge R \subseteq R' \uparrow t \Rightarrow \exists \gamma \geq \alpha' + (t' - t) \cdot (s.(w + (t' - t)), \gamma, R'_1 \cup R'_2 \cup (R'_3 + (t' - t))) \in S, where R_1 = R' \uparrow a, R_2 = \{\alpha, t'\} \times \Sigma (R' \cap ([\alpha, t) \times \Sigma))\}

\[R_3 = R' \cap [t, \infty) \times \Sigma\].

10. \((s, \alpha, R) \in S \wedge R' \in RSET such that R' \subseteq R \Rightarrow \exists \alpha' \geq \alpha such that (s, \alpha', R') \in S\)

11. \((s.w, \alpha, R) \in S \wedge R' \in RSET is such that end(s) \leq begin(R') \wedge end(R') \leq begin(w) \wedge (\forall (t, a) \in R', (s, ((t, a)), R \uparrow t) \notin fail(S)) \Rightarrow (s.w, \alpha, R \cup R') \in S\)

12. \((s, \alpha, R) \in S \Rightarrow (\forall I \in TINT, I \subseteq [\alpha, \infty) \Rightarrow (s, \alpha, R \cup (I \times \Sigma (R \cap ([\alpha, \infty) \times \Sigma)))) \in S\)

The Complete Metric on \(TMFS\).

If \(S \in TMFS\) an \(t \in [0, \infty)\), we define

\[S(t) \triangleq \{(s, \alpha, R) \in S \mid \alpha < t \wedge end(R) < t\} \cup \{(s, \infty, R) \mid end(s) < t \wedge end(R) < t \wedge \exists \alpha \geq t \cdot (s, \alpha, R) \in S\}\].

The complete metric on \(TMFS\) is defined:

\[d(S_t, S_s) \triangleq \inf \{2^{-t} \mid S_t(t) = S_s(t)\}\]

The Semantic Function \(E_T\).

We now define the semantic function \(E_T : TCSP \rightarrow TMFS\).

\[E_T[\bot] \triangleq \{(\bot, \infty, R) \mid R \in RSET\}\]

\[E_T[STOP] \triangleq \{(\bot, 0, R) \mid R \in RSET\}\]

195
\[ E_T[Skip] \equiv \{((), 0, \mathbb{R}) \mid \emptyset \notin \Sigma(\mathbb{R})\} \]
\[ \cup \{((t, \top), t, \mathbb{R}_1 \cup \mathbb{R}_2) \mid t \geq 0 \land (I(\mathbb{R}_1) \subseteq [0, t) \land \top \notin \Sigma(\mathbb{R}_1)) \land I(\mathbb{R}_2) \subseteq [t, \infty)\} \]

\[ E_T[Wait t] \equiv \{(((), t, \mathbb{R}) \mid \mathbb{R} \cap ([t, \infty) \times \{\top\}) = \emptyset\} \]
\[ \cup \{(((t', \top)), t', \mathbb{R}_1 \cup \mathbb{R}_2 \cup \mathbb{R}_3) \mid t' \geq t \land (I(\mathbb{R}_1) \subseteq [0, t) \land I(\mathbb{R}_2) \subseteq [t, t') \land \top \notin \Sigma(\mathbb{R}_3)) \land I(\mathbb{R}_3) \subseteq [t', \infty)\} \]

\[ E_T[a \rightarrow P] \equiv \{(((), 0, \mathbb{R}) \mid a \notin \Sigma(\mathbb{R})\} \]
\[ \cup \{(((t, a)) \sim (s + (t + \delta)), a + t + \delta, \mathbb{R}) \mid t \geq 0 \land a \notin \Sigma(\mathbb{R} \uparrow t)) \land (s, a, \mathbb{R} \downarrow (t + \delta)) \in E_T[P]\} \]

\[ E_T[P \sqcap Q] \equiv SUP\{(((), \max\{\alpha_P, \alpha_Q\}, \mathbb{R}) \mid (((), \alpha_P, \mathbb{R}) \in E_T[P] \land ((), \alpha_Q, \mathbb{R}) \in E_T[Q]\}
\[ \cup \{(s, \alpha, \mathbb{R}) \mid s \neq () \land (s, \alpha, \mathbb{R}) \in E_T[P] \cup E_T[Q] \land ((), \mathbb{R} \uparrow begin(s)) \notin fail(E_T[P]) \land fail(E_T[Q]))\} \]

\[ E_T[P \sqcup Q] \equiv SUP(E_T[P] \cup E_T[Q]) \]

\[ E_T[P \parallel Q] \equiv SUP\{(s, \max\{\alpha_P, \alpha_Q\}, \mathbb{R}_P \cup \mathbb{R}_Q) \mid (s, \alpha_P, \mathbb{R}_P) \in E_T[P] \land (s, \alpha_Q, \mathbb{R}_Q) \in E_T[Q]\} \]

\[ E_T[P \parallel Y Q] \equiv SUP\{(s, \max\{\alpha_P, \alpha_Q\}, \mathbb{R}_P \cup \mathbb{R}_Q \cup \mathbb{R}_X) \mid \exists (s_P, \alpha_P, \mathbb{R}_P) \in E_T[P], (s_Q, \alpha_Q, \mathbb{R}_Q) \in E_T[Q] \land \sigma(\mathbb{R}_P) \subseteq X \land \sigma(\mathbb{R}_Q) \subseteq Y \land 
\[ s \in (s_P \parallel_Y s_Q) \land \sigma(\mathbb{R}_X) \subseteq (\Sigma - (X \cup Y))\} \]

where
\[ v_X \parallel_Y w = \{s \in (T\Sigma)_c \mid s \in (X \cup Y)^* \land 
\[ s \uparrow X = v \land s \uparrow Y = w\} \]

\[ E_T[P || Q] \equiv SUP\{((s, \max\{\alpha_P, \alpha_Q\}, \mathbb{R}) \mid \exists (u, \alpha_P, \mathbb{R}) \in E_T[P] \land (v, \alpha_Q, \mathbb{R}) \in E_T[Q] \land s \in T_{merge}(u, v)\} \]

196
\[ \mathcal{E}_T[P; Q] \equiv CL_{\mathcal{E}}(SUP\{(s, \alpha, \mathcal{N}) \mid \forall I \in TINT \cdot (s, \alpha, \mathcal{N} \cup (I \times \{\cdot\})) \in \mathcal{E}_T[P]\} \]
\[ \cup \{(s^\prec(w+t), \alpha+t, \mathcal{N}_t \cup (\mathcal{N}_s+t)) \mid
\forall \in \sigma(s) \land \text{end}(\mathcal{N}_t) \leq t \land (w, \alpha, \mathcal{N}_s) \in \mathcal{E}_T[Q]\}
\land (s^\prec((t, \cdot)), \mathcal{N}_t \cup (\{0, t\} \times \{\cdot\}) \in \text{fail}(\mathcal{E}_T[P]))\})\}
\]
\[ \mathcal{E}_T[P \setminus X] \equiv SUP\{(s \setminus X, \beta, \mathcal{N}) \mid \exists \alpha \geq \beta \geq \text{end}(s) \cdot
(s, \alpha, \mathcal{N} \cup (\{0, \max\{\beta, \text{end}(\mathcal{N})\}\} \times X) \in \mathcal{E}_T[P]\}\}
\]
\[ \mathcal{E}_T[f^{-1}(P)] \equiv \{(s, \alpha, \mathcal{N}) \mid (f(s), \alpha, \mathcal{N}) \in \mathcal{E}_T[P]\}\]
\[ \mathcal{E}_T[f(P)] \equiv SUP\{((f(s), \alpha, \mathcal{N}) \mid (s, \alpha, f^{-1}(\mathcal{N})) \in \mathcal{E}_T[P]\}\}
\]
\[ \mathcal{E}_T[\mu P.F(p)] \equiv \text{The unique fixed point of the contraction mapping } \hat{C}(Q) = C(WAIT \delta; Q), \text{ where } C \text{ is the mapping on } TM_{FS} \text{ represented by } F.\]

We may also add:
\[ \mathcal{E}_T[a : A \rightarrow P(a)] \equiv \{(\emptyset, 0, \emptyset) \mid A \cap \Sigma(\emptyset) = \emptyset\}
\cup\{((t, a)) \cup (s + (t + \delta)), \alpha + t + \delta, \emptyset) \mid
\left( t \geq 0 \land A \cap \Sigma(\emptyset \cup t) = 0 \right) \land (s, \alpha, \emptyset \cup (t + \delta)) \in \mathcal{E}_T[P]\}\}
\text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}
\]
\[ \mathcal{E}_T[\bigcap P(a)] \equiv SUP(\bigcup \mathcal{E}_T[P(a)])
\text{when } \{P(a) \mid a \in A\} \text{ is uniformly bounded}
\]
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

$$ P ::= \bot \mid STOP \mid SKIP \mid a \rightarrow P \mid a : A \rightarrow P(a) \mid P \cap Q \mid \cap P(a) \mid P \sqcap Q \mid P \parallel Q \mid P_{x \parallel y} Q \mid P \parallel Q \mid \parallel P ; Q \mid P \backslash A \mid f(P) \mid f^{-1}(P) \mid X \mid \mu X \bullet P $$

$TCSP$ with variables

$$ P ::= \bot \mid STOP \mid SKIP \mid WAIT t \mid WAIT t ; P \mid a \rightarrow P \mid a : A \rightarrow P(a) \mid P \cap Q \mid \cap P(a) \mid P \sqcap Q \mid P \parallel Q \mid \parallel P ; Q \mid P \backslash A \mid f(P) \mid f^{-1}(P) \mid X \mid \mu X \bullet P $$

The Semantic Function $T$.

The environment $\sigma$ is of type $\text{var} \rightarrow M_T$.

We now define a semantic function $T : CSP \rightarrow M_T$. 


\[ T[\bot]_\sigma = \{(\\)} \]
\[ T[\text{STOP}]_\sigma = \{(\\)} \]
\[ T[\text{SKIP}]_\sigma = \{(\\), (\checkmark)} \]
\[ T[a \rightarrow P]_\sigma = \{(\\)} \cup \{(a) \checkmark s \mid s \in T[P]_\sigma \} \]
\[ T[a : A \rightarrow P(a)]_\sigma = \{(\\)} \cup \{(a) \checkmark s \mid a \in A \land s \in T[P(a)]_\sigma \} \]
\[ T[P \sqcap Q]_\sigma = T[P]_\sigma \sqcap T[Q]_\sigma \]
\[ T[P \sqcup Q]_\sigma = T[P]_\sigma \sqcup T[Q]_\sigma \]
\[ T[\neg P(a)]_\sigma = \sqcup T[P(a)]_\sigma \]
\[ T[P \parallel Q]_\sigma = T[P]_\sigma \sqcap T[Q]_\sigma \]
\[ T[P_X \parallel_Y Q]_\sigma = \{s \mid s \uparrow X \in T[P]_\sigma \land s \uparrow Y \in T[Q]_\sigma \land s \uparrow (X \cup Y) = s \} \]
\[ T[P \parallel Q]_\sigma = \{s \mid \exists u \in T[P]_\sigma, v \in T[Q]_\sigma \bullet \]
\[ s \in \text{Merge}(u, v) \} \]
\[ T[P ; Q]_\sigma = \{s \mid s \in T[P]_\sigma \land \checkmark \notin s \}
\[ \cup \{s \checkmark t \mid s \checkmark (\checkmark) \in T[P]_\sigma \land t \in T[Q]_\sigma \} \]
\[ T[P \setminus X]_\sigma = \{s \setminus X \mid s \in T[P]_\sigma \} \]
\[ T[f^{-1}(P)]_\sigma = \{s \mid f(s) \in T[P]_\sigma \} \]
\[ T[X]_\sigma = \sigma(X) \]
\[ T[\mu X \bullet P]_\sigma = \text{The least fixed point of the continuous mapping } \lambda Y \bullet T[P](\sigma[Y/X]) \text{ with respect to the complete partial order } \subseteq \text{ on } M_T. \]
\[ T[\mu X \bullet P]_\sigma = \text{The unique fixed point of the contraction mapping } \lambda Y \bullet T[P](\sigma[Y/X]) \text{ with respect to the complete metric } d \text{ on } M_T. \]
The Semantic Function $\mathcal{F}$.

We now define the semantic function $\mathcal{F} : \text{CSP} \rightarrow M_F$.

\begin{align*}
\mathcal{F}[\bot]_\sigma & \equiv \{()_X \mid X \in P(\Sigma)\} \\
\mathcal{F}[\text{STOP}]_\sigma & \equiv \{()_X \mid X \in P(\Sigma)\} \\
\mathcal{F}[\text{SKIP}]_\sigma & \equiv \{()_X \mid \diamond \notin X\} \cup \{()_X \mid X \in P(\Sigma)\} \\
\mathcal{F}[a \rightarrow P]_\sigma & \equiv \{()_X \mid a \notin X\} \\
& \quad \cup \{(a)^s_0, (s, X) \in \mathcal{F}[P]_\sigma\} \\
\mathcal{F}[a : A \rightarrow P(a)]_\sigma & \equiv \{()_X \mid A \cap X = \emptyset\} \\
& \quad \cup \{(a)^s_0, (s, X) \in \mathcal{F}[P]_\sigma\} \\
\mathcal{F}[P \sqcap Q]_\sigma & \equiv \{()_X \mid ()_X \in \mathcal{F}[P]_\sigma \cap \mathcal{F}[Q]_\sigma\} \\
\mathcal{F}[P \cap Q]_\sigma & \equiv \mathcal{F}[P]_\sigma \cup \mathcal{F}[Q]_\sigma \\
\mathcal{F}[\bigcap P(a)]_\sigma & \equiv \bigcup \mathcal{F}[P(a)]_\sigma \\
\mathcal{F}[P \parallel Q]_\sigma & \equiv \{()_X \mid (s, X, Y) \in F[P]_\sigma\} \\
& \quad \wedge (s, Y) \in F[Q]_\sigma \\
\mathcal{F}[P X \parallel Y Q]_\sigma & \equiv \{(s, Z_P \cup Z_Q) \mid (X \cup Y) = s \wedge Z_P \subseteq X \wedge Z_Q \subseteq Y \wedge (\Sigma - (X \cup Y)) \\
& \quad \wedge (s) \in F[P]_\sigma \\
& \quad \wedge (s, Y, Z_Q) \in F[Q]_\sigma\} \\
\mathcal{F}[P ; Q]_\sigma & \equiv \{()_X \mid (u, X) \in F[P]_\sigma\} \\
& \quad \wedge (u, X) \in F[Q]_\sigma \\
\mathcal{F}[P \setminus X]_\sigma & \equiv \{()_X \mid (s, X, Y) \in F[P]_\sigma\} \\
& \quad \wedge (s, Y) \in F[Q]_\sigma \\
\mathcal{F}[f^{-1}(P)]_\sigma & \equiv \{()_X \mid (f(s), f(X)) \in F[P]_\sigma\} \\
\mathcal{F}[f(P)]_\sigma & \equiv \{(f(s), X) \mid (s, f^{-1}(X)) \in F[P]_\sigma\} \\
\mathcal{F}[X]_\sigma & \equiv \sigma(X) \\
\mathcal{F}[\mu X \cdot P]_\sigma & \equiv \text{The unique fixed point of the contraction mapping } \lambda Y \cdot \mathcal{F}[P](\sigma[Y/X]) \text{ on } M_F.
\end{align*}
The Semantic Function $E^*_T$.

We now define the semantic function $E^*_T : TCSP \rightarrow TM_{FS}$. 

$$E^*_T[\bot] \rho \equiv \{((\emptyset, \infty, X) | X \in P(\Sigma)\}$$

$$E^*_T[STOP] \rho \equiv \{((\emptyset, 0, X) | X \in P(\Sigma)\}$$

$$E^*_T[SKIP] \rho \equiv \{((\emptyset, 0, X) | \sqrt{\notin X}\}$$

$$\cup \{((\emptyset, t'), t, X) | t \geq 0 \wedge \sqrt{\notin X}\}$$

$$E^*_T[WAIT t] \rho \equiv \{((\emptyset, t, X) | \sqrt{\notin X}\}$$

$$\cup \{((\emptyset, t'), t, X) | X \in P(\Sigma)\}$$

$$\cup \{((\emptyset, t'), t', X) | t' \geq t \wedge X \in P(\Sigma)\}$$

$$E^*_T[WAIT t; P] \rho \equiv \{((\emptyset, t, X) | \sqrt{\notin X}\}$$

$$\cup \{(s + t, \alpha + t, X) | (s, \alpha, X) \in E^*_T[P] \rho\}$$

$$E^*_T[a \rightarrow P] \rho \equiv \{((\emptyset, 0, X) | a \notin X\}$$

$$\cup \{((s + \delta, \alpha + \delta, X) | (s, \alpha, X) \in E^*_T[P] \rho\}$$

$$\cup \{((s + t + \delta, \alpha + t + \delta, X) | t \geq 0 \wedge (s, \alpha, X) \in E^*_T[P] \rho\}$$

$$E^*_T[a : A \rightarrow P(a)] \rho \equiv \{((\emptyset, 0, X) | A \cap X = \emptyset\}$$

$$\cup \{((s + \delta, \alpha + \delta, X) | a \in A \wedge (s, \alpha, X) \in E^*_T[P(a)] \rho\}$$

$$\cup \{((s + t + \delta, \alpha + t + \delta, X) | a \in A \wedge t \geq 0 \wedge (s, \alpha, X) \in E^*_T[P(a)] \rho\}$$

when \(\{P(a) | a \in A\}\) is uniformly bounded

$$E^*_T[P \cap Q] \rho \equiv sup\{((\emptyset, \alpha, X) | ((\emptyset, \alpha) \in stab(E^*_T[P] \rho \cup E^*_T[Q] \rho)\}$$

$$\wedge ((\emptyset, X) \in fail(E^*_T[P] \rho) \cap fail(E^*_T[Q] \rho)\}$$

$$\cup \{((s, \alpha, X) \in E^*_T[P] \rho \cup E^*_T[Q] \rho \wedge \delta \neq (s)\}}$$

$$E^*_T[\cup P(a)] \rho \equiv sup(E^*_T[P] \rho \cup E^*_T[Q] \rho)$$

when \(\{P(a) | a \in A\}\) is uniformly bounded

$$E^*_T[P \cup Q] \rho \equiv sup\{((sp \cup sq) \max\{\alpha P, \alpha Q\}, X P \cup X Q) | (sp, \alpha P, X P) \in E^*_T[P] \rho \wedge (sq, \alpha Q, X Q) \in E^*_T[Q] \rho\}$$

$$\wedge \delta P = \delta Q\}$$

201
\( \mathcal{E}_T \{ P \parallel Y \mid Q \} \rho \equiv \{(s, \max\{\alpha_p, \alpha_Q\}, Z_p \cup Z_Q \cup Z) \mid \\
\exists (sp, \alpha_p, Z_p) \in \mathcal{E}_T \{ P \} \rho \land (sq, \alpha_Q, Z_Q) \in \mathcal{E}_T \{ Q \} \rho \land \\
Z_p \subseteq X \land Z_Q \subseteq Y \land \\
s \in (sp_x \parallel y sq) \land Z \subseteq (\Sigma - (X \cup Y))) \}

\( \mathcal{E}_T \{ P \parallel Q \} \rho \equiv \text{SUP} (\{(s, \max\{\alpha_p, \alpha_Q\}, X) \mid \exists (u, \alpha_p, X) \in \mathcal{E}_T \{ P \} \rho, \\
(v, \alpha_Q, X) \in \mathcal{E}_T \{ Q \} \rho \land s \in \text{Tmerge}(u, v)\}) \)

\( \mathcal{E}_T \{ P ; Q \} \rho \equiv \text{CL} (\text{SUP} (\{(s, \alpha, X) \mid (s, \alpha, X \cup \{\gamma\}) \in \mathcal{E}_T \{ P \} \rho \land \gamma \notin \Sigma(s) \land \\
\cup (s^\gamma(w + t), \alpha + t, X) \mid s^\gamma((t, \gamma)) \in \text{Traces}(\mathcal{E}_T \{ P \} \rho), \\
\land \gamma \notin \Sigma(s) \land \gamma \in \mathcal{E}_T \{ Q \} \rho)) \}

\( \mathcal{E}_T \{ P \setminus X \} \rho \equiv \{(s \setminus X, \alpha, Y) \mid s \text{ is } X\text{-active in } \mathcal{E}_T \{ P \} \rho \\
\land \alpha = \text{sup} (\beta \mid \exists (w, \beta) \in \mathcal{E}_T \{ P \} \rho \text{ such that } \\
w \text{ is } X\text{-active } \land w \setminus X = s \setminus X) \\
\land (\alpha < \infty \land (s, X \cup Y) \in \text{fail}(\mathcal{E}_T \{ P \} \rho) \\
\lor (\alpha = \infty \land Y \in \mathcal{P}(\Sigma)))\} \}

where \( s \) is \( X\)-active provided
\( s \) contains no element of the form \((t, a)\) for \( a \in X \)
(all communications in \( X \) are in the form \( \hat{a} \)).

\( \mathcal{E}_T \{ f^{-1}(P) \} \rho \equiv \{(s, \alpha, X) \mid (f(s), \alpha, f(X)) \in \mathcal{E}_T \{ P \} \rho\} \)

\( \mathcal{E}_T \{ f(P) \} \rho \equiv \text{SUP} (\{(f(s), \alpha, X) \mid (s, \alpha, f^{-1}(X)) \in \mathcal{E}_T \{ P \} \rho\}) \)

\( \mathcal{E}_T \{ X \} \rho \equiv \rho(X) \)

\( \mathcal{E}_T \{ \mu X \cdot P \} \rho \equiv \text{The unique fixed point of the contraction mapping } \lambda Y \cdot \mathcal{E}_T \{ P \} (\rho(W_\delta(Y)/X)), \text{ where } \\
W_\delta \text{ is the mapping on } TM_{\delta} \text{ corresponding to } \\
\lambda Y \cdot \text{WAIT } \delta ; Y. \)
The Semantic Function $\mathcal{E}$.

We now define the semantic function $\mathcal{E} : TCSP \rightarrow TMFS$.

\[
\begin{align*}
\mathcal{E}[\bot] &\triangleq \{(((), \infty, \emptyset) | \emptyset \in RSET) \\
\mathcal{E}[\text{STOP}] &\triangleq \{(((), 0, \emptyset) | \emptyset \in RSET) \\
\mathcal{E}[\text{SKIP}] &\triangleq \{(((), 0, \emptyset) | \emptyset \notin \Sigma(\emptyset)) \\
\cup \{(((t, \vee)), t, R_1 \cup R_2) | t \geq 0 \land (I(R_1) \subseteq [0, t) \land \emptyset \notin \Sigma(R_2)) \land I(R_2) \subseteq [t, \infty)\} \\
\mathcal{E}[\text{WAIT} \ t] &\triangleq \{(((), t, R) | R \cap ([t, \infty) \times \{\vee\}) = \emptyset) \\
\cup \{(((t', \vee)), t', R_1 \cup R_2 \cup R_3) | t' \geq t \land I(R_1) \subseteq [0, t) \land I(R_2) \subseteq [t, t') \land \emptyset \notin \Sigma(R_3)) \land I(R_3) \subseteq [t', \infty)\} \\
\mathcal{E}[\text{WAIT} \ t; \ P] &\triangleq \{((s, \alpha, R) | (s - t, \alpha - t, R - t) \in \mathcal{E}[P] \rho) \\
\mathcal{E}[a \rightarrow P] &\triangleq \{(((), 0, \emptyset) | a \notin \Sigma(\emptyset)) \\
\cup \{(((t, a)), s + (t + \delta), \alpha + t + \delta, R) | t \geq 0 \land a \notin \Sigma(R) \land (s, \alpha, R - (t + \delta)) \in \mathcal{E}[P] \rho\} \\
\mathcal{E}[a : A \rightarrow P(a)] &\triangleq \{(((), 0, \emptyset) | A \cap \Sigma(\emptyset) = \emptyset) \\
\cup \{(((t, a)), s + (t + \delta), \alpha + t + \delta, R) | t \geq 0 \land A \cap \Sigma(R) \land (0) \land (s, \alpha, R - (t + \delta)) \in \mathcal{E}[P] \rho\} \\
\text{when } \{P(a) | a \in A\} \text{ is uniformly bounded} \\
\mathcal{E}[P \square Q] &\triangleq \text{SUP}(\{(((), \max \{\alpha_P, \alpha_Q\}, \emptyset) | (((), \alpha_P, \emptyset) \in \mathcal{E}[P] \rho \land (((), \alpha_Q, \emptyset) \in \mathcal{E}[Q] \rho) \\
\cup \{(s, \alpha, R) | s \neq () \land (s, \alpha, R) \in \mathcal{E}[P] \rho \cup \mathcal{E}[Q] \rho) \\
\land ((), R \uparrow \text{begin}(s)) \in \text{fail}((\mathcal{E}[P] \rho) \cap \text{fail}((\mathcal{E}[Q] \rho)))\}) \\
\mathcal{E}[P \cap Q] &\triangleq \text{SUP}(\mathcal{E}[P] \rho \cup \mathcal{E}[Q] \rho) \\
\mathcal{E}[\bigcap P(a)] &\triangleq \text{SUP}(\bigcup \mathcal{E}[P(a)] \rho) \\
\text{when } \{P(a) | a \in A\} \text{ is uniformly bounded} \\
\mathcal{E}[P \parallel Q] &\triangleq \text{SUP}(\{((s, \max \{\alpha_P, \alpha_Q\}, R_P \cup R_Q) | (s, \alpha_P, R_P) \in \mathcal{E}[P] \rho \land (s, \alpha_Q, R_Q) \in \mathcal{E}[Q] \rho)\})
\end{align*}
\]
\[ \mathcal{E}_T[P \parallel Y \parallel Q] \rho \equiv \sup \{ (s, \alpha, \beta, \rho) | \exists (s', \alpha', \beta', \rho') \in \mathcal{E}_T[P] \rho \land (s, \alpha, \rho) \in \mathcal{E}_T[Q] \rho \land s \in \{ s' \parallel \rho \} \} \]

where

\[ s \parallel \rho \equiv \{ s \in \Sigma^* | s \in (X \cup Y)^* \land s' \parallel X = v \lor s \parallel Y = w \} \]

\[ \mathcal{E}_T[P \parallel Q] \rho \equiv \sup \{ (s, \alpha, \beta, \rho) | \exists (s', \alpha', \beta', \rho') \in \mathcal{E}_T[P] \rho \land (s, \alpha, \beta, \rho) \in \mathcal{E}_T[Q] \rho \land s \in \text{merge}(u, v) \} \]

\[ \mathcal{E}_T[P; Q] \rho \equiv \text{CL}_{\mathcal{E}_T}(\sup \{ (s, \alpha, \beta, \rho) | \exists (s', \alpha', \beta', \rho') \in \mathcal{E}_T[P] \rho \land (s', \alpha', \rho' \cup \{ (s, \alpha, \beta, \rho) \}) \} \]

\[ \mathcal{E}_T[P \setminus X] \rho \equiv \sup \{ (s \setminus X, \alpha, \beta, \rho) | \exists (s', \alpha', \beta', \rho') \in \mathcal{E}_T[P] \rho \land (s, \alpha, \rho) \in \{ 0, \max(\beta, \text{end}(\rho)) \} \times X \} \}

\[ \mathcal{E}_T[f^{-1}(P)] \rho \equiv \{ (s, \alpha, \beta, \rho) | (s, \alpha, f(\beta)) \in \mathcal{E}_T[P] \rho \}
\]

\[ \mathcal{E}_T[f(P)] \rho \equiv \sup \{ (f(s), \alpha, \rho) | (s, \alpha, f^{-1}(\beta)) \in \mathcal{E}_T[P] \rho \}
\]

\[ \mathcal{E}_T[X] \rho \equiv \rho(X) \]

\[ \mathcal{E}_T[\mu X \cdot P] \rho \equiv \text{The unique fixed point of the contraction mapping } \lambda Y \cdot \mathcal{E}_T[P] \rho(W_{\delta}(Y)/X), \text{ where } W_{\delta} \text{ is the mapping on } TM_{FS} \text{ corresponding to } \lambda Y \cdot \text{WAIT } \delta; Y. \]
Bibliography


the International Workshop on Timed Petri Nets, Torino, Italy, July 1985


### Index of Notation

<table>
<thead>
<tr>
<th>Notation on traces</th>
<th>Page</th>
<th>Notation on failures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T\Sigma_t$</td>
<td>5</td>
<td>$begin$</td>
<td>8</td>
</tr>
<tr>
<td>$\neg$</td>
<td>5</td>
<td>$end$</td>
<td>8</td>
</tr>
<tr>
<td>#</td>
<td>5</td>
<td>$\uparrow t$</td>
<td>8</td>
</tr>
<tr>
<td>in</td>
<td>5</td>
<td>$\uparrow I$</td>
<td>8</td>
</tr>
<tr>
<td>first</td>
<td>5</td>
<td>$\uparrow A$</td>
<td>8</td>
</tr>
<tr>
<td>last</td>
<td>5</td>
<td>$\setminus t$</td>
<td>9</td>
</tr>
<tr>
<td>begin</td>
<td>5</td>
<td>$\sigma$</td>
<td>9</td>
</tr>
<tr>
<td>end</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@</td>
<td>6</td>
<td>$begin$</td>
<td>9</td>
</tr>
<tr>
<td>$\uparrow I$</td>
<td>6</td>
<td>$end$</td>
<td>9</td>
</tr>
<tr>
<td>$\uparrow t$</td>
<td>6</td>
<td>$\uparrow I$</td>
<td>9</td>
</tr>
<tr>
<td>$\setminus t$</td>
<td>9</td>
<td>$\uparrow A$ (on events)</td>
<td>9</td>
</tr>
<tr>
<td>$\setminus A$</td>
<td>9</td>
<td>$\widehat{A}$ (A-active)</td>
<td>9</td>
</tr>
<tr>
<td>$\setminus X/Y$</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x \parallel y$</td>
<td>7</td>
<td>$traces$</td>
<td>10</td>
</tr>
<tr>
<td>$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\triangledown$</td>
<td>7</td>
<td>$stab$</td>
<td>10</td>
</tr>
<tr>
<td>$Tmerge$</td>
<td>7</td>
<td>$SUP$</td>
<td>10</td>
</tr>
<tr>
<td>$\vee$</td>
<td>7</td>
<td>$CL_\infty$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sigma$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$T$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$S$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mathcal{F}$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\mathcal{E}$</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notation on refusal sets</th>
<th>Page</th>
<th>Notation on processes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TINT$</td>
<td>8</td>
<td>$traces$</td>
<td>10</td>
</tr>
<tr>
<td>$RTOK$</td>
<td>8</td>
<td>$fail$</td>
<td>10</td>
</tr>
<tr>
<td>$RSET$</td>
<td>8</td>
<td>$stab$</td>
<td>10</td>
</tr>
<tr>
<td>$I(\mathbb{N})$</td>
<td>8</td>
<td>$SUP$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$CL_\infty$</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sigma$</td>
<td>10</td>
</tr>
</tbody>
</table>

210
<table>
<thead>
<tr>
<th>Notation</th>
<th>Page</th>
<th>Notation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_T$</td>
<td>10</td>
<td>$ST_T[P]$</td>
<td>29</td>
</tr>
<tr>
<td>$S_T$</td>
<td>10</td>
<td>sat</td>
<td>31</td>
</tr>
<tr>
<td>$F_T$</td>
<td>10</td>
<td>$TF$</td>
<td>42</td>
</tr>
<tr>
<td>$E_T^*$</td>
<td>10</td>
<td>$SS$</td>
<td>43</td>
</tr>
<tr>
<td>$E_T$</td>
<td>10</td>
<td>$I(P_1, P_2)$</td>
<td>49</td>
</tr>
<tr>
<td>$\equiv$</td>
<td>10</td>
<td>$\emptyset$</td>
<td>89</td>
</tr>
<tr>
<td>inits</td>
<td>10</td>
<td>$\sqsubseteq_i$ (between processes)</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqsubseteq_i$ (between predicates)</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqsubseteq_{ts}$ (between processes)</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$sr$ (on $MT$ specifications)</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$wc$ (to $MT$ specifications)</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqsubseteq_f$ (between processes)</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqsubseteq_{ts}$ (between processes)</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sqsubseteq_f$ (between predicates)</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$sr$ (on $MF$ specifications)</td>
<td>119</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$wc$ (to $MF$ specifications)</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c.v$</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c.V$</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c?z$</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$c!z$</td>
<td>124</td>
</tr>
<tr>
<td>$i :$</td>
<td>10</td>
<td>$\leq$</td>
<td>124</td>
</tr>
<tr>
<td>$\text{rem}_i$</td>
<td>11</td>
<td>$\leq_m$</td>
<td>124</td>
</tr>
<tr>
<td>$\parallel_I$</td>
<td>11</td>
<td>$\gg$</td>
<td>125</td>
</tr>
<tr>
<td>$;^i$</td>
<td>12</td>
<td>$S_{a,b}$</td>
<td>125</td>
</tr>
<tr>
<td>$\prod_I$</td>
<td>14</td>
<td>$IRSET$</td>
<td>28</td>
</tr>
<tr>
<td>$a : A \rightarrow P_a$</td>
<td>16</td>
<td>$PAR$</td>
<td>133</td>
</tr>
<tr>
<td>$^i$</td>
<td>19</td>
<td>$I(P)$</td>
<td>28</td>
</tr>
<tr>
<td>$^i$</td>
<td>21</td>
<td>$NET$</td>
<td>133</td>
</tr>
<tr>
<td>$\nabla^i$</td>
<td>24</td>
<td>$\triangleright$</td>
<td>156</td>
</tr>
<tr>
<td>$\nabla^i$</td>
<td>24</td>
<td>$\lessdot$</td>
<td>156</td>
</tr>
<tr>
<td>$i \in I$</td>
<td>156</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>