Specification and Proof in Real-Time Systems

by

Jim Davies

Technical Monograph PRG-93
ISBN 0-902928-71-6

April 1991

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

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

Electronic mail: `jdavies@uk.ac.oxford.prg`

Specification and Proof in Real-time Systems

Jim Davies

Thesis submitted for the degree of Doctor of Philosophy
at the University of Oxford, January 1991

Abstract

This thesis shows how the mathematical theory of Timed Communicating Sequential Processes (Timed CSP) developed by Reed and Roscoe may be applied to the specification and proof of complex real-time systems. A number of substantial additions are made to the theory, producing a powerful tool for the analysis and implementation of timing requirements and concurrency.

The syntax and semantics of Timed CSP are extended to include new primitive operators for timing and recursion. A language of behavioural specifications is formulated, together with a complete, compositional proof system. A significant case study is used to illustrate these developments. The language is then extended to include an element of broadcast concurrency.

Acknowledgments

I am indebted to my supervisor, Jim Woodcock, for his guidance, encouragement, and good taste. Also to Steve Schneider, whose concurrent approach to the subject has lead to many enjoyable discussions and collaborative results. The research described in this thesis is founded upon earlier work by Mike Reed and Bill Roscoe; I am very grateful to them for their insight and friendship.

I would like to thank Jeremy Gibbons, Michael Goldsmith, Tony Hoare, Dave Jackson, Alan Jeffrey, Geraint Jones, Mathai Joseph, Andrew Kay, and Joy Reed for their friendly advice and invaluable suggestions. Finally, I wish to thank BP International and the Science and Engineering Research Council for financial support during the course of this research.

Dedication

To my parents, for their love and support

and

to Alice, for everything

Time present and time past
Are both perhaps present in time future
And time future contained in time past.
If all time is eternally present
All time is unredeemable.
What might have been is an abstraction
Remaining a perpetual possibility
Only in a world of speculation

T.S. ELIOT

Contents

Chapter 0

Introduction

As computing devices become faster and more powerful, we find ourselves ever more dependent upon systems which are difficult to nnderstand and prone to failure. The failure of a commercial banking system, or a company database, may be expensive and inconvenient. If the system is prescribing medicine, or landing an aircraft, then the results might be fatal. As the consequences of system failure become more severe, we must find ways to make applications of computer technology safer and more reliable.

Over the past twenty years, mathematical techniques have been developed for the specification and implementation of transformational systems, which compute results from a given set of inputs. However, most of the systems in which safety is a primary concern are reactive systems, whicb maintain a continuous interaction with the environment, and are often subject to complex timing constraints. These systems cannot be viewed in a transformational setting.

Real-time systems are reactive, and are often required to perform several tasks concurrently. To reason about such systems, we require a mathematical formalism that includes timing information as well as an effective treatment of concurrency. In this thesis, we explore and extend one such formalism: Timed Communicating Sequential Processes, first presented in [Reed & Roscoe 86]. Timed CSP is an extension of Hoare's CSP [Hoare 85] that allows us to reason about time-dependent aspects of concurrent behaviour.

The aim of this thesis is to present a formal development method for real-time systems, based upon the semantic models proposed by Reed and Roscoe. Tbis method should support both formal and rigorous reasoning at every stage of system development—from initial specification to final implementation—and be applicable to systems of a realistic size. It is our hope that the results of the research described in this thesis may be used to improve the safety and reliability of real-time distributed systems.

The thesis begins with an introduction to the language of Timed CSP, and the semantic models presented in [Reed 88]. In chapter two, we extend this language to include primitive operators for modelling timeouts and interrupts, as well as alternative forms of parallel and sequential composition. These operators are given a semantics in Reed's Timed Failures model, and an intuitive explanation is provided for each operator in the extended language.

In Reed and Roscoe's treatment of recursion, a strictly positive delay δ is associated with each recursive call. In chapter three, we investigate the consequences of dispensing with this delay, and give a sufficient condition for the validity of a recursive definition. The theory is then extended to permit mutual recursion: processes may be defined by sets of mutually recursive equations. Semantics-preserving rules are established for manipulating these sets, which may be arbitrarily large.

In the Timed Failures model, each language construct is identified with a set of possible behaviours. In chapter four, we show how the informal requirements upon a system may be captnred as behavioural specifications—predicates upon an arbitrary system behaviour—and demonstrate that the notation of the semantic model gives rise to a simple specification language. We investigate the form of safety and liveness conditions in this language, and show that it may be used to formalise assumptions about the environment of a system.

The notation introduced in chapter four may be used to produce a formal specification of a system, and the extended language described in chapters two and three may be used to suggest possible implementations. In chapter five, we present a complete proof system for relating specifications and implementations, using the notion of satisfaction introduced in [Hoare 85]. This system is compositional, in the sense of [Hooman & de Roever 89]:

> *Properties of a compound programming language construct (such as sequential composition and parallel composition) can be deduced from specifications for its constituent parts without any further information about the internal structure of these parts.*

This is essential if the proof system is to be employed in the development of large, complex systems.

A formal specification of a real-time system will include many requirements that can be established without timing information. In this case, we may use the untimed models for CSP presented in [Reed 88] to simplify our proof obligations. A substantial part of chapter five is devoted to a simple theory of timewise refinement, which relates uutimed safety conditions in the Timed Failures model to behavioural specifications in tbe untimed Traces model. The research described in this chapter is a continuation of research carried out jointly with Steve Schneider, some of which is reported in [Davies & Schneider 89] and [Schneider 89].

If we wish to produce a readable specification of a large system, then we must take care to present our description in a clear, structured fashion. In chapter six, we show how the hiding operator may be used to structure specifications, and present a simple proof rule for abstraction which allows us to separate the concerns of concealment and scheduling. The chapter continues with the introduction of a macro specification language—a first-order logic with time—which may be used to simplify the process of requirements captnre.

Chapter seven presents a case study in the application of Timed CSP to real-time distributed systems. It begins with a detailed method for the specification and development of hierarchical protocols, based upon the proof system of chapter five. This method is then applied to the development of a local area network protocol. The specification language of chapter six is used to describe the behaviour of the protocol at different levels of abstraction, and the system description language of chapters two and three is used to suggest an implementation.

In a description of a real-time process, it is sometimes convenient to include observable events that are not synchronisations: this can make it easier to describe and analyse certain aspects of behaviour. In chapter eight, we show how the Timed Failures model may be extended to include a treatment of broadcast concurrency, in which output events may occur without the cooperation of the environment. The resulting semantic model is then used to complete the implementation of the communications protocol presented in chapter seven.

In the final chapter of the thesis, we discuss the results of the research presented in the preceding chapters. We consider alternative approaches to the specification and development of real-time systems, and outline directions for future work. The thesis ends with an appendix of mathematical proofs, and a glossary of symbols.

Chapter 1

The Language of Timed CSP

1.1 Communicating Sequential Processes

In [Hoare 85], Hoare nses the word *process* to denote the behaviour pattern of an object, viewed through the occurrence or availability of certain atomic actions, or *events*. These processes may be seen as entities which evolve and communicate with an environment by synchronising upon a set of such actions. An observable event is thus an atomic communication between a process and its environment.

The syntax of CSP is a process algebra; the terms representing processes may be rewritten in accordance with certain algebraic laws. These laws are justified by a number of semantic models for the language, in which each CSP term is associated with a set of possible behaviours. In the simplest of these models, each process is associated with a set of *traces*: sequences of observable events. The other models include more information in the semantic set, and allow us to draw finer distinctions between processes.

The syntax inclndes primitive operators for parallel composition, nondeterministic choice, and hiding. This makes for an elegant notation in which the problems of concurrency, nondeterminism, and abstraction can be addressed separately. The syntax also provides constructs for modelling deadlock, recursion, and process relabelling:

$$\begin{aligned} P \ ::= \ & STOP \mid SKIP \mid a \to P \mid P \Box P \mid P \sqcap P \mid P \,;\, P \mid P \parallel P \mid \\ & P \mathbin{|||} P \mid f(P) \mid f^{-1}(P) \mid P \setminus A \mid \mu X \bullet F(X) \end{aligned}$$

Tbe variety of operators in CSP is in contrast to other algebraic approaches to concurrency, in which much emphasis is placed upon obtaining a minimal set of operators for the syntax.

The semantic models can be used to specify the intended behaviour of a process. As each process is associated with a set of behaviours in the semantic model, a predicate on the semantic set corresponds to a requirement upon the process. For example, in the Traces model of CSP, we may capture the requirement that process P never performs a visible action with the predicate

$$\forall\, tr \in traces(P) \bullet tr = \langle\rangle$$

In this model, the process $STOP$ is associated with the singleton set $\{\langle\rangle\}$, containing only the empty trace. We may conclude that $STOP$ is a process that meets this requirement.

The Traces model M_T is sufficient if we wish to analyse untimed *safety* requirements; these are constraints that proscribe certain events or sequences of events in the history of a process. However, if we wish to ensure that a synchronisation event is offered to the environment, we must include either readiness or *refusal* information in our semantic model. In the Failures model M_F we associate each trace of a process with the set of events that may be refused afterwards. If the failure (tr, X) is present in the semantic set of process P, then P may perform trace tr and then refuse to engage in any event from X.

In Hoare's book, a third aspect of behaviour is considered: the *divergences* of a process. A trace of process P is a divergence if it may be followed by an unbounded sequence of internal events, during which P may refuse to communicate with its environment. Reed's thesis [Reed 88] contains an alternative treatment of divergence. In his Stability model M_S, each trace of a process is associated with a stability value of ∞ or 0, depending on whether or not the process may diverge after engaging in that trace. These models form a simple hierarchy:

$$\begin{tikzcd}
& M_{FS} \arrow[dl] \arrow[dr] & \\
M_F \arrow[dr] & & M_S \arrow[dl] \\
& M_T &
\end{tikzcd}$$

Figure 1.1: Reed's models for CSP

The Failures-Stability model M_{FS} corresponds to the Failures-Divergences model used in [Hoare 85]. In Reed's model, processes are associated with sets of triples

(tr, α, X). A stability value α is attached to each failure; if the value is zero, then the process is stable after performing trace tr: it does not diverge. An infinite stability value indicates that internal activity may continue indefinitely. The arrows in the diagram correspond to projection mappings between the models; behavioural results established in one model remain valid in models lower in the hierarchy.

1.2 Timed Models

The models of CSP presented in [Roscoe 82, Brookes 83, Hoare 85] do not inclnde timing information. By considering only the sequence of observable events, and the subsequent refusal sets, we obtain simplified semantic models with a nnmber of convenient algehraic laws. However, if the logical correctness of a design is dependent upon the precise timing of certain events, we cannot complete our reasoning within the formalism of nntimed CSP.

If we wish to use CSP to describe a real-time system, in which the precise timing of events is important, we must employ timed models for the language. The first timed model for CSP, presented in [Jones 82], proved unsatisfactory for a number of technical reasons. The author suggested that a better model could be obtained by recording the events refnsed *during* the observation of a trace; this is a feature of the later, more successful attempt made by Reed and Roscoe.

Since Jones's attempt, a number of other timed models have been postulated for CSP-like languages, notably in [Zwarico 86, Boucher & Gerth 87]. However, the timed models presented by Reed and Roscoe in [Reed & Roscoe 86, Reed & Roscoe 87, Reed 88] have the following advantages:

* the models are compatible with the existing untimed models of CSP
* infinite hiding and infinite alphabet transformations are possible
* deadlock and divergence may be distinguished
* divergence may be distingnished from the possibility of divergence
* the models are arranged in a hierarchy

The last consideration is an important one. In reasoning about complex systems, we may use the simplest semantic model that is sufficient to express the current requirement, safe in the knowledge that the argnment remains valid in the other models of the hierarchy.

In his thesis [Reed 88], Reed presents five timed models for CSP. In each model, a process is associated with a set of possible timed behaviours. A typical element of a semantic set is a tuple, the elements of which represent different aspects of a possible behaviour. Just as the untimed models recorded trace, refusal and stability information, the timed models record timed traces, timed refusals, and timed stabilities.

The hierarchy of models is ordered by the information content of the semantic sets. The models are linked by projection mappings, represented by arrows in the diagram below; the nature of these mappings ensures that results established in one model remain true as we move downwards.

$$\begin{tikzcd}
 & & TM_{FS} \arrow[ddll] \arrow[d] \arrow[dr] & & \\
 & & M_{FS} \arrow[dl] \arrow[dr] & TM^*_{FS} \arrow[l] \arrow[dr] & \\
TM_F \arrow[r] \arrow[ddrr] & M_F \arrow[dr] & & M_S \arrow[dl] & TM_S \arrow[l] \arrow[ddll] \\
 & & M_T & & \\
 & & TM_T \arrow[u] & &
\end{tikzcd}$$

Figure 1.2: Reed's models for Timed CSP

The untimed models of CSP occupy the lowest positions in the hierarchy, with the untimed Traces model M_T at the very bottom. The simplest of the timed models, TM_T, associates a process with a set of timed traces. The Timed Failures model TM_F, and the Timed Failures-Stability model TM_{FS} record the events refused by a process during and after the observation of each timed trace.

The timed stability models (TM_S, TM^*_{FS} and TM_{FS}), include information about the presence of internal activity. The stability value of a behaviour is the earliest time by which all internal activity is *guaranteed* to have ceased. In the Timed Failures-Stability model, each failure $(s, \aleph)$ of a process is associated with a single stability value α between *0* and ∞, inclusive. If the process exhibits the external behaviour described by $(s, \aleph)$, then all internal activity must cease at or before time α.

The untimed failures-timed stability model TM^*_{FS} records the set of events refused after a timed trace, once the process has stabilised. This model bridges the gap between the Timed Stability model TM_S and the Timed Failures-Stability model TM_{FS}. These models are used in the theory of timewise refinement presented in [Schneider 89]; simple processes may be refined by the introduction of timing information and results established in the lower models give rise to corresponding results in models further up the hierarchy.

In the specification of a real-time system, internal activity is usually of only secondary importance. The correctness of a design will be expressed as a set of constraints upon the occurrence and availability of observable events or external synchronisations. This is precisely the information that may be obtained from the Timed Failures model TM_F. Furthermore, the timed models without timed refusals are complicated by the need to record the times at which events first become available, in order to give a satisfactory semantics to the hiding operator. For these reasons, we will restrict our attention to the Timed Failures model of Timed CSP.

1.3 A Model of Computation

The models presented in [Reed 88] are compatible with the earlier models of CSP given in [Roscoe 82, Brookes 83]; as such, they share the same model of computation: processes communicate by *handshaking*, observable events require the cooperation of the environment, and any behaviour of a process appears the same to all observers. To introduce timing information into this model of computation, several assumptions are required:

Real Time With the non-negative real numbers as our time domain, we have no lower bound on the interval between consecutive events. This allows us to model asynchronous processes in a satisfactory fashion, without artificial constraints upon the times at which independent events may be observed.

Global Clock All observations are recorded with reference to an imaginary global clock, but this clock cannot be accessed by any part of the system being modelled. If a system clock is required, it can be modelled as a simple Timed CSP process. Separate clocks may be modelled as separate processes, and need not keep the same time.

Instantaneous Events All events have zero duration. If a system action takes a significant amount of time to perform, we use two events in our representation: one corresponding to the start of the action, another to the end. Similarly, we consider communications between processes to be instantaneous: delays in transmission, reception, and synchronisation are made explicit.

Termination There is a single *termination* event, ✓, whose occurrence signals the successful termination of a construct. If this construct is followed immediately by a sequential composition operator, then the ✓ event is hidden from the environment, and termination occurs as soon as possible.

Finite Speed We assume that no process can engage in infinitely many events in a finite time. This assumption is enforced by the axioms of our semantic model, and leads to constraints upon the application of certain operators, e.g. indexed nondeterministic choice.

Hiding and Control Observable events cannot occur without the cooperation of the environment. Further, if a process and its environment are both prepared to engage in an event at a particular time, then it occurs at that time. Hidden events do not require the cooperation of the environment, and occur as soon as they become available.

Delay Constant We choose a strictly positive delay constant δ as a lower bound between consecutive events in a sequential process. This ensures that cause precedes effect in any observation of a process: if the occurrence of event a makes another event b possible, then b cannot occur at the same time as a. The existence of such a delay greatly simplifies the analysis of sequential processes.

1.4 Timed CSP

In [Reed & Roscoe 86], Reed and Roscoe present the following syntax for the language of Timed CSP:

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

This is identical to the syntax for untimed CSP presented in [Brookes 83], but for the inclusion of the *WAIT* construct. The addition of this operator allows us to model most forms of timed interaction.

To facilitate a treatment of mutual recursion, we will consider the syntax of *TCSP* terms, rather than processes. In chapter 2, we will add a clause (X) to the syntax to introduce variables from a set *VAR*, and write recursive terms in the form $\mu X \bullet P$. A process will be a *TCSP* term with no free variables: its meaning will be independent of the values of variables from VAR. However, the body of a recursive process *will* contain free variables, and it is necessary to consider the bindings of these variables while reasoning about the process.

The terms $\perp$ and $STOP$ correspond to the divergent and deadlocked processes, respectively. The process $SKIP$ signals successful termination, and $WAIT\ t$ does nothing but terminate successfully after a delay of time t. The event prefix operator $a \rightarrow P$ prefixes a term P with a single event a. A constant delay is associated with this operation: control is not passed to P until δ after a is observed. No such delay is associated with the external and internal choice operators, nor with the sequential composition operator.

The alphabet parallel operator, ${}_A\|_B$, provides a means of synchronisation between processes. In the parallel combination $P\ {}_A\|_B\ Q$, process P may perform events from set A and process Q may perform events from set B; the two processes must cooperate on events drawn from the intersection of the two sets. In a simple parallel combination ($\|$), the two processes must agree on all events, while interleaved processes ($|||$) run asynchronously.

The hiding operator ($\backslash$) allows us to abstract from internal events, concealing them from the environment of a process, and the renaming operators (f and f^{-1}) allow us to relabel the events of a process. All of these operators will be discussed in greater detail in chapter 2, which presents a complete semantics for the language of Timed CSP.

1.5 Example

We consider the user interface of a simple timed vending machine VMS. Users of this machine may insert a coin and, after a short delay, press a button to release a drink. The machine then returns to its original state.

The insertion of a coin is modelled by the event *coin*, and we allow a time t_{drop} for the coin to drop, before the event *button* is made available. If the user then presses the button, the machine will offer a drink: this corresponds to the availability of the event *coke* after a short delay of time t_{coke}.

$$\begin{aligned} VMS \ \widehat{=}\ & coin \rightarrow WAIT\,(t_{drop} - \delta)\,; \\ & \qquad button \rightarrow WAIT\,(t_{coke} - \delta)\,; \\ & \qquad\qquad coke \rightarrow WAIT\,(t_{reset} - \delta)\,;\ VMS \end{aligned}$$

The machine takes time t_{reset} to prepare for another transaction.

The machine VMS presents the user with no choice of product, so the button is an unnecessary feature of the interface. We may use the hiding operator to conceal the event *button* from the user. Hidden events occur as soon as they become available, so the machine $VMS \setminus button$ will behave as follows:

$$\begin{aligned} VMS \setminus button \;\equiv\; & coin \rightarrow WAIT\,(t_{drop} + t_{coke} - \delta)\,; \\ & \qquad coke \rightarrow WAIT\,(t_{reset} - \delta)\,;\,(VMS \setminus button) \end{aligned}$$

The delays before and after the *button* event are unaffected by the hiding operator.

The process relabelling operator may be used to rename the events of a process while retaining the control structure. Suppose that the vending machine is used to dispense a different product; we may model this change with any function f such that

$$\begin{aligned} f(coin) &\;\widehat{=}\; coin \\ f(coke) &\;\widehat{=}\; pepsi \end{aligned}$$

which transforms the vending machine:

$$\begin{aligned} f(VMS \setminus button) \;\equiv\; & coin \rightarrow WAIT\,(t_{drop} + t_{coke} - \delta)\,; \\ & \qquad pepsi \rightarrow WAIT\,(t_{reset} - \delta)\,;\,f(VMS \setminus button) \end{aligned}$$

Again, the delay times are unaffected.

In the above example, the arguments of the delay operator $WAIT$ are adjusted to take account of the constant delay of δ tbat is associated with the event prefix operator. A more elegant description may be obtained using the delayed form of the prefix operator, introduced in the next chapter.

Chapter 2

The Timed Failures Model

A timed event is a pair (t, a), where time t is a non-negative real number and a is drawn from Σ, the set of all events. A timed trace is a finite sequence of timed events arranged in chronological order. For example,

$$s \mathrel{\widehat{=}} \langle (1, a), (3, b) \rangle$$

defines a timed trace in which event a is observed at time 1, and event b is observed at time 3. The order of events in a trace depends only on the time at which they occur. If more than one event is observed at the same time, then these events may appear in any order in the trace.

Timed refusals are sets of timed events. The presence of a timed event (t, a) in a refusal set corresponds to the refusal of a process to participate in event a at time t. One of the assumptions of our computational model, that processes can evolve only at a finite rate, allows us to place the following constraint upon the construction of timed refusals: they are formed by a finite union of product sets, called refusal tokens. A refusal token is a cross product $I \times A$, where I is a half-open finite interval within $[0, \infty)$ and A is a set of events. For example, the timed refusal defined by

$$\aleph \mathrel{\widehat{=}} [1, 2) \times \{a, b\}$$

consists of a single refusal token, and corresponds to the refusal of a process to participate in events a and b between time 1 and time 2.

Timed failures are timed $(\mathit{trace}, \mathit{refusal})$ pairs. The presence of a timed failure $(s, \aleph)$ in the semantic set of a process indicates that the process may perform s while refusing the timed events in $\aleph$. There is no reason why the same timed event (t, a) should not be present in both components of a timed failure. This will occur whenever a process performs as many copies of event a as it can at time t, and thus refuses to perform a further copy of a at that time. For example, the failure

$$(\langle (1, a), (3, b) \rangle, [1, 2) \times \{a, b\})$$

describes a behaviour in which a process engages in event a at time 1, and refuses to perform a second a from this time onwards.

When considering the interaction of a process with its environment, we may view a timed trace as a result of an experiment performed upon a process: the environment offers timed events to the process, which the process may or may not accept. The refusal set represents a *partial* record of these offers: our knowledge of the experiment. The presence of a pair (t, a) in the refusal set indicates that the environment offered more copies of the event a at time t than the process was willing to perform.

2.1 Notation

We use $T\Sigma$ to denote the set of all timed events, and $T\Sigma^*_\leqslant$ to denote the set of all timed traces. $TINT$ is the set of all finite intervals within the time domain $TIME$, which is $[0, \infty)$. $RTOK$ is the set of all possible refusal tokens, $RSET$ denotes the set of all timed refnsals, and TF is the set of all timed failures:

$$\begin{array}{rcl}
T\Sigma & \widehat{=} & TIME \times \Sigma \\
T\Sigma^*_\leqslant & \widehat{=} & \{s \in \text{seq}\, T\Sigma \mid (t, a) \text{ precedes } (t', a') \text{ in } s \Rightarrow t \leqslant t'\} \\
TINT & \widehat{=} & \{[b, e) \mid 0 \leqslant b < e < \infty\} \\
RTOK & \widehat{=} & \{I \times A \mid I \in TINT \wedge A \in \mathbb{P}\,\Sigma\} \\
RSET & \widehat{=} & \{\bigcup C \mid C \in \mathbb{F}\, RTOK\} \\
TF & \widehat{=} & T\Sigma^*_\leqslant \times RSET \\
TS_F & \widehat{=} & \mathbb{P}\, TF
\end{array}$$

In the Timed Failures model, processes are represented by elements of TS_F, the space of sets of timed failures. To reason abont the possible behaviours of a process, we will use the language of set and sequence theory. We inherit the following notation from [Hoare 85]:

$\langle\rangle$	the empty trace
$\frown$	concatenation of traces
in	contiguous subsequence

The predicate $s_1 \text{ in } s_2$ holds precisely when trace s_1 is a contiguous subsequence of s_2. To give a semantics to our language, and to simplify the process of reasoning about it, we define a variety of simple operators on timed traces, timed refusals, and timed failures.

First and Last

The *first* and *last* operators are defined for all timed traces, returning the first and last events in a trace, if non-empty. If the trace is empty, they return the special non-event ε. All that is required of ε is that $\varepsilon \notin \Sigma$. We also define *begin* and *end* operators, which yield the times of the first and last events:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
first(\langle\rangle) & \mathrel{\widehat=} & \varepsilon & begin(\langle\rangle) & \mathrel{\widehat=} & \infty \\
first(\langle(t,a)\rangle^\frown s) & \mathrel{\widehat=} & a & begin(\langle(t,a)\rangle^\frown s) & \mathrel{\widehat=} & t \\
last(\langle\rangle) & \mathrel{\widehat=} & \varepsilon & end(\langle\rangle) & \mathrel{\widehat=} & 0 \\
last(s^\frown\langle(t,a)\rangle) & \mathrel{\widehat=} & a & end(s^\frown\langle(t,a)\rangle) & \mathrel{\widehat=} & t
\end{array}$$

The values chosen for the empty trace are the most convenient for the subsequent mathematics: the possibility of a trace being empty will not require special consideration in our specifications and proofs. It proves convenient to define *head* and *foot* operators on traces:

$$\begin{array}{rcl}
head(s) & \mathrel{\widehat=} & (begin(s), first(s)) \\
foot(s) & \mathrel{\widehat=} & (end(s), last(s))
\end{array}$$

Times

The *times* operator returns the set of time values that appear in a refusal set:

$$times(\aleph) \mathrel{\widehat=} \{t \mid \exists\, a \bullet (t,a) \in \aleph\}$$

We may use this operator to define *begin* and *end* operators on refusal sets:

$$\begin{array}{rcll}
begin(\aleph) & \mathrel{\widehat=} & inf(times(\aleph)) & \text{if } \aleph \neq \{\} \\
begin(\{\}) & \mathrel{\widehat=} & \infty & \\
end(\aleph) & \mathrel{\widehat=} & sup(times(\aleph)) & \text{if } \aleph \neq \{\} \\
end(\{\}) & \mathrel{\widehat=} & 0 &
\end{array}$$

For convenience, we extend the above definitions to timed failures:

$$\begin{array}{rcl}
begin(s,\aleph) & \mathrel{\widehat=} & min\{begin(s), begin(\aleph)\} \\
end(s,\aleph) & \mathrel{\widehat=} & max\{end(s), end(\aleph)\}
\end{array}$$

During, Before and After

We define the *during* ($\uparrow$), *before* ($\upharpoonleft$), and *after* ($\upharpoonright$) operators on timed traces. The first returns the maximal subsequence of the trace with times drawn from set I. The others return the parts of the trace before and after the specified time.

$$\begin{array}{rcll} \langle\rangle \uparrow I & \widehat{=} & \langle\rangle & \\ (\langle(t,a)\rangle^\frown s) \uparrow I & \widehat{=} & \langle(t,a)\rangle^\frown(s \uparrow I) & \text{if } t \in I \\ & & (s \uparrow I) & \text{otherwise} \\ s \upharpoonleft t & \widehat{=} & s \uparrow [0,t] & \\ s \upharpoonright t & \widehat{=} & s \uparrow (t,\infty) & \end{array}$$

where I is a set of real numbers. In the case that $I = \{t\}$ for some time t, we may omit the set brackets. These operators may also be applied to timed refusals, with the following interpretations:

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

Recalling that Σ denotes the set of all events, we see that these restrict a refusal set to events that may be refused before, during, and after the specified times. The definitions of *before* and *after* on refusal sets differ from those on timed traces. For traces, $s \upharpoonleft t$ includes events at t; in the case of refusals, such events are excluded. The opposite is true of the *after* operator. This choice of definitions is the most convenient for timed failures specifications.

Restriction

We use the $\downharpoonright$ symbol to denote the restriction of a timed trace or refusal to a set of events A.

$$\begin{array}{rcll} \langle\rangle \downharpoonright A & \widehat{=} & \langle\rangle & \\ \langle(t,a)\rangle^\frown s \downharpoonright A & \widehat{=} & \langle(t,a)\rangle^\frown(s \downharpoonright A) & \text{if } a \in A \\ & & s \downharpoonright A & \text{otherwise} \\ & & & \\ \aleph \downharpoonright A & \widehat{=} & \aleph \cap ([0,\infty) \times A) & \end{array}$$

The hiding operator on traces may be defined as a restriction:

$$s \setminus A \;\widehat{=}\; s \downharpoonright (\Sigma - A)$$

Alphabets

We define an alphabet (or *event set*) operator on traces and refusals, yielding the set of events present:

$$\begin{array}{rcl}\sigma(s) & \mathrel{\widehat{=}} & \{a \in \Sigma \mid \exists\, t \bullet \langle(t,a)\rangle \ \mathbf{in}\ s\} \\ \sigma(\aleph) & \mathrel{\widehat{=}} & \{a \in \Sigma \mid \exists\, t \bullet (t,a) \in \aleph\}\end{array}$$

The event set of a term P is the set of all observable events that may be performed by the corresponding process:

$$\sigma(P) \ \mathrel{\widehat{=}}\ \{a \in \Sigma \mid \exists (s,\aleph) \in \mathcal{F}_T[\![P]\!] \bullet a \in \sigma(s)\}$$

where $\mathcal{F}_T[\![P]\!]$ denotes the semantic set of P.

Subtraction

To reason about any form of sequential composition or delay, we require a subtraction operator that shifts timed traces and refusals through time:

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

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

It proves economical to define a subtraction operator on timed failures:

$$(s,\aleph) - t \ \mathrel{\widehat{=}}\ (s \dot{-} t, \aleph \dot{-} t)$$

Equivalence and Closure

We define an equivalence relation upon the set of timed traces:

$$u \cong v \ \Leftrightarrow\ u \text{ is a permutation of } v$$

Note that, as timed traces are chronologically ordered sequences, equivalent traces may differ only in the order of appearance of simultaneous events. We use this equivalence to define a closure operator on sets of timed failures:

$$CL_{\cong}(S) \ \mathrel{\widehat{=}}\ \{(s,\aleph) \in TF \mid \exists (w,\aleph) \in S \bullet s \cong w\}$$

2.2 The Timed Failures Model

The Timed Failures Model TM_F is defined to be those elements S of TS_F which satisfy the following axioms:

1. $(\langle\rangle, \{\}) \in S$

2. $(s^\frown w, \aleph) \in S \Rightarrow (s, \aleph \upharpoonright begin(w)) \in S$

3. $(s, \aleph) \in S \wedge s \cong w \Rightarrow (w, \aleph) \in S$

4. $(s, \aleph) \in S \wedge t \geqslant 0 \Rightarrow \exists \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s, \aleph') \in S \wedge$
   $((t' \leqslant t \wedge (t', a) \notin \aleph') \Rightarrow (s \upharpoonright t'^\frown\langle(t', a)\rangle, \aleph' \upharpoonright t') \in S)$

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

6. $\forall \aleph' : RSET \bullet (s, \aleph) \in S \wedge \aleph' \subseteq \aleph \Rightarrow (s, \aleph') \in S$

The first axiom requires that the empty failure $(\langle\rangle, \{\})$ is a possible behaviour of any process. The second is a prefix closure condition: if a process may perform a trace $s^\frown w$ while refusing $\aleph$, then it should be able to perform the prefix s, with the refusal set truncated accordingly. The third axiom insists that the order of events in a timed trace depends only upon the times at which they are observed, no additional information about causal relationships is available.

The fourth axiom enforces our assumption that processes may undergo only a finite number of state changes in a finite time. For any failure $(s, \aleph)$ and time value t, there will always be a maximal refusal set $\aleph'$ that captures all of the refusal information for the current trace, at least until time t. Given any time $t' \leqslant t$, every timed event (t', a) not in $\aleph'$ is a possible extension of $s \upharpoonright t'$. As $\aleph'$ is a refusal set, it must be a *finite* union of refusal tokens, and hence represents only finitely many changes of state.

The fifth axiom places a similar condition upon traces. For any process S, we can exhibit a function n that places a bound upon the number of events observed before a given time. If trace s ends at or before time t, then the length of s must be no greater than $n(t)$. This bounded speed condition leads to constraints upon the application of infinitary operators such as prefix choice and indexed nondeterminism. The final axiom states that if a process may refuse the whole of $\aleph$, then it may refuse any subset of $\aleph$. A similar condition holds in the untimed failures model described in [Brookes 83].

We define a distance metric d on TS_F by considering the first time at which the elements of two sets may be distinguished. If $(s, \aleph)$ is a timed failure, we define a projection function on elements of TS_F:

$$S \upharpoonright t \;\; \mathrel{\widehat{=}} \;\; \{(s, \aleph) \mid (s, \aleph) \in S \wedge end(s, \aleph) \leqslant t\}$$

If S is a element of TS_F then $S \upharpoonright t$ is the set of elements of S which do not extend beyond time t. We may now define the metric:

$$d(S, T) \;\; \mathrel{\widehat{=}} \;\; inf(\{2^{-t} \mid S \upharpoonright t = T \upharpoonright t\} \cup \{1\})$$

This definition is equivalent to the one used in [Reed 88], although the definition of $S \upharpoonright t$ differs slightly. The metric will be needed when we give a semantics to recursive process definitions.

2.3 A Semantic Function

We will give a semantics to a language $TCSP$ of Timed CSP terms, defined by

$$\begin{array}{rcll}
P & ::= & \bot \mid STOP \mid SKIP \mid WAIT\ t \mid X \mid & \text{atoms} \\
 & & a \rightarrow P \mid a \xrightarrow{t} P \mid P \,;\, P \mid P \fatsemi P \mid & \text{sequential composition} \\
 & & P \Box P \mid P \sqcap P \mid a : A \xrightarrow{t_a} P_a \mid & \text{alternation} \\
 & & P \parallel P \mid P \;{}_A\|_B\; P \mid P \mid\mid\mid P \mid P \underset{A}{\parallel} P \mid & \text{parallel composition} \\
 & & P \setminus A \mid f(P) \mid f^{-1}(P) \mid & \text{abstraction and renaming} \\
 & & P \underset{t}{\lightning} P \mid P \underset{t}{\nabla} P \mid P \overset{t}{\rhd} P \mid & \text{timing} \\
 & & \mu X \bullet P \mid \mu X \circ P \mid \langle X_i = P_i \rangle_j & \text{recursion}
\end{array}$$

In the above syntax, clause X introduces variables from a set VAR; these are required for the treatment of mntual recnrsion presented in chapter 3. To give a semantics to this language, we require a formal treatment of variable bindings.

We define a domain of environments, ENV, consisting of all mappings from variables VAR to the space of all sets of timed failures TS_F, and thus a semantic function for terms:

$$\begin{aligned} ENV &\mathrel{\widehat{=}} VAR \rightarrow TS_F \\ \mathcal{F}_T &\in TCSP \rightarrow ENV \rightarrow TS_F \end{aligned}$$

We write $\mathcal{F}_T[\![P]\!]\rho$ to denote the semantics of a term P in an environment ρ. This may be evaluated by associating each free variable X with its value $\rho[\![X]\!]$ in the current environment.

We give the following semantics to syntactic substitution:

$$\mathcal{F}_T[\![P[Q/X]]\!]\rho \;\hat{=}\; \mathcal{F}_T[\![P]\!]\rho[\mathcal{F}_T[\![Q]\!]\rho/X]$$

where $\rho[Y/X]$ is a new environment, defined as follows:

$$\rho[Y/X][\![Z]\!] \;\hat{=}\; \begin{array}{ll} Y & \text{if } Z = X \\ \rho[\![Z]\!] & \text{otherwise} \end{array}$$

A Timed CSP process will be represented by a *TCSP* term with no free variables: its meaning will be independent of the current environment. If P is a process then we may infer that

$$\forall \rho, \rho' : ENV \;\bullet\; \mathcal{F}_T[\![P]\!]\rho = \mathcal{F}_T[\![P]\!]\rho'$$

In this case, we may sensibly omit the environment parameter.

2.4 Sequential Processes

Atoms

The divergent process $\perp$ can perform no observable actions, but internal activity may continue indefinitely. In the Timed Failures model we do not record the possibility of internal activity, and so $\perp$ is identified with the deadlocked process *STOP*. The only trace of either process is $\langle\rangle$, the empty trace.

$$\begin{aligned} \mathcal{F}_T[\![\perp]\!]\rho &\;\hat{=}\; \{(\langle\rangle, \aleph) \mid \aleph \in RSET\} \\ \mathcal{F}_T[\![STOP]\!]\rho &\;\hat{=}\; \{(\langle\rangle, \aleph) \mid \aleph \in RSET\} \end{aligned}$$

Both processes are capable of refusing any event from Σ at any time.

The process *SKIP* models successful termination in Timed CSP. This is signalled by an occurrence of the special event $\checkmark$, the only action that this process may perform:

$$\begin{aligned} \mathcal{F}_T[\![SKIP]\!]\rho \;\hat{=}\; & \{(\langle\rangle, \aleph) \mid \checkmark \notin \sigma(\aleph)\} \\ & \cup \\ & \{(\langle(t, \checkmark)\rangle, \aleph) \mid t \geqslant 0 \wedge \checkmark \notin \sigma(\aleph \upharpoonright t)\} \end{aligned}$$

Either no events have been observed and the event $\checkmark$ is available, or $\checkmark$ has been observed (at some time t) and was continuously available beforehand.

The delay process *WAIT t* represents delayed successful termination, with the event ✓ becoming available at time t. It can be used to introduce an additional delay into a sequential process, or combined with other operators to produce timeout and interrupt constructs.

$$\begin{aligned}\mathcal{F}_T[\![WAIT\ t]\!]\rho \ \widehat{=}\ & \{(\langle\rangle, \aleph) \mid \checkmark \notin \sigma(\aleph \restriction t)\} \\ & \cup \\ & \{(\langle(t', \checkmark)\rangle, \aleph) \mid t' \geqslant t \wedge \checkmark \notin \sigma(\aleph \uparrow [t, t'))\}\end{aligned}$$

If no events have been observed then ✓ must be available continuously from time t onwards. Otherwise, ✓ is observed at a time $t' \geqslant t$ and made available at all times between t and t'. what after t'?

Prefix

The event prefix operator is used to introduce an observable event into a process description; the expression $a \rightarrow P$ denotes a process that is prepared initially to engage only in event a, and then behave as process P. There is a non-zero delay associated with this operation, corresponding to the time taken to change from a state in which event a is available, to one in which it has been performed. The undecorated prefix operator is associated with a constant delay δ.

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

If no events have been observed in a history of $a \rightarrow P$, then event a cannot be refused. Otherwise, a is the first event to be observed and the subsequent behaviour, following a delay of δ, is due to P.

The above operator will be used only when the minimum delay following an event is unimportant. If we are interested in the delay following the observation of an event a then we decorate the prefix operator with a time value: the expression

$$a \xrightarrow{t} P$$

denotes a process which is willing to perform an event a. If a occurs, the process will then behave as process P, once a delay of time t has elapsed. During the time delay, the process behaves as *WAIT*, refusing to participate in any external activity. This is illustrated by the equivalence:

$$a \xrightarrow{t} P \ \equiv\ a \rightarrow WAIT\,(t - \delta)\,; P$$

The semantics of this operator may be derived from the equations for the delay, prefix and sequential composition operators.

We retain the δ constant as a lower bound on the delay associated with the prefix operation. This is necessary if we wish to avoid the possibility of causally-related simultaneous events. To see why this is a problem, postulate the existence of an instantaneous prefix operator $\rightarrow$ with the following interpretation: the expression $a \rightarrow P$ denotes a process that is initially prepared to engage in an event a; once a is observed, the process immediately behaves as P. If P is ready to perform an event immediately, then that event may be observed at the same time as a. Consider the process

$$a \rightarrow b \rightarrow STOP$$

This process may perform b at any time t, providing that it performs a at (or before) that time. Now consider the parallel combination

$$a \rightarrow b \rightarrow STOP \quad \| \quad b \rightarrow a \rightarrow STOP$$

We expect this combination to deadlock immediately. However, both components may perform a and b together at any time t. Simultaneous events may appear in any order in a timed trace, so the parallel combination may perform traces from the following set:

$$\{\langle\rangle\} \cup \{s \mid t \geqslant 0 \land s \cong \langle(t,a),(t,b)\rangle\}$$

This clashes with our intuition about processes and observable events. Events a and b are inseparable, yet they appear separately in traces of a process. As we might expect, this situation is proscribed by an axiom of the semantic model:

$$(s^\frown w, \aleph) \in S \quad \Rightarrow \quad (s, \aleph \upharpoonright begin(w)) \in S$$

We do not allow an effect to precede its cause in a trace.

Sequential Composition

The expression $P \,;\, Q$ denotes the sequential composition of processes P and Q. No delay is associated with this operator; the last event of process P may occur at the same time as the first event from process Q. This need not conflict with our intuition about causal relationships and delay, as the initial state of Q is independent of the final state of P. A behaviour $(s, \aleph)$ of $P \,;\, Q$ may be either

1. a behaviour of P which does not correspond to successful termination, or

2. a terminating behaviour of P, followed by some behaviour of Q

In the first case, s is a trace of P in which $\checkmark$ is not observed, and would be refnsed if offered; this corresponds to the first component of the semantic set. In the second case, the trace s is obtaiued from two traces, s_P and s_Q, performed by P and Q respectively:

$$\begin{aligned}\mathcal{F}_T[\![P\,;Q]\!]\rho \;\hat{=}\; & \{(s,\aleph) \mid \checkmark \notin \sigma(s) \wedge \\ & \qquad \forall I : \mathit{TINT} \bullet (s, \aleph \cup (I \times \{\checkmark\})) \in \mathcal{F}_T[\![P]\!]\rho\} \\ & \cup \\ & CL_{\cong}\{(s_P{}^\frown s_Q, \aleph) \mid \checkmark \notin \sigma(s_P) \wedge (s_Q, \aleph) - t \in \mathcal{F}_T[\![Q]\!]\rho \wedge \\ & \qquad (s_P{}^\frown\langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup ([0,t) \times \{\checkmark\})) \in \mathcal{F}_T[\![P]\!]\rho\}\end{aligned}$$

If control has been transferred at time t, then the trace s_P could have been extended with a $\checkmark$ event at that time. This event is hidden from the environment by the sequential composition operator, and occurs as soon as it becomes available; it mnst be possible for P to refuse $\checkmark$ up until time t while performing trace s_P. The subseqnent behaviour is due to process Q.

The above equation is complicated by the fact that both processes are able to perform actions at time t. Simultaneous events may appear in any order in a timed trace, so we must ensure that our semantic set is closed under trace equivalence. The resulting definition is unsuitable for some applications; in chapter 5 we will see that it fails to preserve timewise refinements. Because of this, we introduce a delayed seqnential composition operator:

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

This defines a process that behaves as P until successful termination is signalled, then waits for an interval length δ before behaving as Q. This delay allows us to separate the events of the first process from those of the second.

Nondeterministic Choice

The expressiou $P \sqcap Q$ denotes the nondeterministic choice between processes P and Q. This operator is sometimes called *internal choice*, as there is no way for the environment to influence the flow of coutrol at this point:

$$\mathcal{F}_T[\![P \sqcap Q]\!]\rho \;\hat{=}\; \mathcal{F}_T[\![P]\!]\rho \cup \mathcal{F}_T[\![Q]\!]\rho$$

We require only that every behaviour of a nondeterministic choice is a possible behaviour of at least one component.

If wish to model arbitrary nondeterministic choice, then we must verify that there is a uniform bound upon the speed of the alternatives. This will ensure that the resulting process can perform only a bounded number of events before any finite time t, in accordance with axiom 5 of section 2.2. We say that a set of processes $\{P_i \mid i \in I\}$ is *uniformly bounded* if there exists a function $n : TIME \longrightarrow \mathbb{N}$ such that for all environments ρ

$$\forall i : I \,;\, t : TIME \bullet (s, \aleph) \in \mathcal{F}_T[\![P_i]\!]\rho \wedge end(s) \leqslant t \Rightarrow \#(s \upharpoonright t) \leqslant n(t)$$

This definition is due to Steve Schneider, and provides a necessary and sufficient condition for the following semantics to be well-defined:

$$\mathcal{F}_T[\![\bigsqcap_{i \in I} P_i]\!]\rho \;\hat{=}\; \bigcup_{i \in I} \mathcal{F}_T[\![P_i]\!]\rho$$

This operator may be used to model nondeterministic delays in sequential processes. We overload the delay operator

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

to define a process that is prepared to terminate after some time t, where t is drawn from the set T.

Deterministic Choice

The expression $P \,\Box\, Q$ denotes a deterministic choice between processes P and Q. This operator is sometimes called *general choice* (or external choice) as the environment may select either P or Q by offering to engage in events which are initially possible for just one of the two processes. The choice is resolved by the first observable event that occurs.

$$\begin{aligned}
\mathcal{F}_T[\![P \,\Box\, Q]\!]\rho \;\hat{=}\; & \{(\langle\rangle, \aleph) \mid (\langle\rangle, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \cap \mathcal{F}_T[\![Q]\!]\rho\} \\
& \cup \\
& \{(s, \aleph) \mid s \neq \langle\rangle \wedge (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \cup \mathcal{F}_T[\![Q]\!]\rho \\
& \qquad\qquad \wedge \\
& \qquad\qquad (\langle\rangle, \aleph \upharpoonright begin(s)) \in \mathcal{F}_T[\![P]\!]\rho \cap \mathcal{F}_T[\![Q]\!]\rho\}
\end{aligned}$$

Any behaviour must be a behaviour of at least one component, and any event refnsed before the first observable event must be refused by both processes.

We know from [Reed 88] that it is not possible to define a deterministic choice operator for Timed CSP which offers a choice over an infinite set of processes. However, [Schneider 89] shows that we may offer a choice over an infinite set of events. As an example, suppose that we wish to define a process that is ready to accept any natural number value on a channel c. Such a process may be modelled as an infinite prefix choice. The expression

$$c.n : c.\mathbb{N} \xrightarrow{t_n} P_n$$

denotes a process that is ready initially to engage in auy event $c.n$ for $n \in \mathbb{N}$. If $c.n$ is observed, the process delays for time t_n and then behaves as P_n.

$$\begin{aligned}\mathcal{F}_T\left[\!\left[a : A \xrightarrow{t_a} P_a\right]\!\right]\rho \;\; \widehat{=} \;\; & \{(\langle\rangle, \aleph) \mid A \cap \sigma(\aleph) = \{\}\} \\ & \cup \\ & \{(\langle(t, a)\rangle^\frown s, \aleph) \mid a \in A \wedge t \geqslant 0 \wedge \\ & \qquad\qquad begin(s) \geqslant t + t_a \\ & \qquad\qquad A \cap \sigma(\aleph \upharpoonright t) = \{\} \wedge \\ & \qquad\qquad (s, \aleph) - (t + t_a) \in \mathcal{F}_T[\![P_a]\!]\rho\}\end{aligned}$$

If no events have been observed then all of the events in set A are available. Otherwise some event a from A has occurred, and the subsequent behaviour is that of P_a. As in the case of indexed nondeterministic choice, this semantics is well-defined if and only if the set of alternative processes is uniformly bounded.

Relabelling

We use process renaming functions to systematically rename the observable events of a process while retaining the control structure. There are two syntactic clauses for relabelling processes, allowing the use of many-to-one or one-to-many relations, providing that either the relation or its inverse is a function f on Σ. Suppose that a and b are events such that $b = f(a)$.

The *inverse image* of P may perform a whenever P may perform b:

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

and refuse a whenever P may refuse b.

The *direct image* of P may perform b whenever P may perform a. As f may be many-to-one, the refusal of an event by process P corresponds to the refusal of a set of events by the image process.

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

In the above equation, the expression $f^{-1}(\aleph)$ denotes the set

$$\{(t, a) \mid (t, f(a)) \in \aleph\}$$

This is the inverse image of refusal set $\aleph$ under function f.

Abstraction

The hiding operator allows us to conceal those events in the history of a system which do not require the cooperation of the environment. Such a structuring mechanism is necessary if we wish to produce readable descriptions of large, complex systems. The expression $P \setminus A$ denotes a process that behaves as P, except that

* events from A happen as soon as they become available
* only events outside A may appear in a trace

In our model of computation, an event occurs as soon as all of the processes involved are willing to cooperate. A hidden event does not require the cooperation of the environment, and will occur as soon as it becomes available.

Events which occur as soon as they are made available may be continuously refused by the process in question: if $(s, \aleph)$ is a behaviour of a process P in which every instance of event a occurs as soon as possible, then

$$(s, \aleph \cup [0, end(s, \aleph)) \times \{a\})$$

is also a behaviour of P. This is a consequence of the fourth axiom of our semantic model, which asserts the existence of a maximal refusal set containing $\aleph$:

$$\begin{aligned}(s,\aleph) \in S \wedge t \geqslant 0 \;\Rightarrow\; & \exists \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s,\aleph') \in S \wedge \\ & ((t' \leqslant t \wedge (t',a) \notin \aleph') \Rightarrow (s \upharpoonright t' \frown \langle (t',a) \rangle, \aleph' \upharpoonright t') \in S)\end{aligned}$$

Now suppose that there exists a time $t < end(s, \aleph)$ such that $(t, a) \notin \aleph'$. By our choice of $\aleph'$, we may infer that (t, a) is a possible extension of the trace $s \upharpoonright t$. This conflicts with our assumption that $(s, \aleph)$ is a behaviour in which every copy of a occurred as soon as it becomes available: $s \upharpoonright t$ already contains as many copies of a as P was able to perform up to and including that time. Hence

$$(\aleph \cup [0, end(s,\aleph)) \times \{a\}) \subseteq \aleph' \quad \wedge \quad (s, \aleph') \in \mathcal{F}_T[\![P]\!]$$

The result follows by the sixth axiom of the semantic model: the refusal sets corresponding to a trace s are closed under the subset relation.

The behaviours of $P \setminus A$ may be obtained from those failures of P in which events from A are continuously refused:

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

Any events from A which appear in trace s are removed by the trace concealment operator, defined in section 2.1 by

$$s \setminus A \;\;\widehat{=}\;\; s \downharpoonright (\Sigma - A)$$

where $\downharpoonright$ denotes set restriction.

2.5 Parallel Processes

Alphabet Parallel

In [Hoare 85], each process P is associated with a set of events αP, the process alphabet. If P appears in a synchronised parallel combination, events from αP require the cooperation of P. In Timed CSP, the need for process alphabets is removed by the introduction of an alphabet parallel operator. This operator is parametrised by two sets of events; in the parallel combination

$$P \mathbin{{}_A\|_B} Q$$

process P may perform only those events in A, process Q may perform only those events in B, and the two processes must cooperate on events drawn from the intersection of A and B. Events that are in neither A nor B are proscribed.

If s is a trace of this parallel combination, the restriction of s to events from set A yields the trace of events performed by process P. Similarly, restricting s to the set B yields the trace of events performed by Q. If these traces are s_P and s_Q respectively, then s is an element of the set

$$s_P \mathbin{{}_A\|_B} s_Q \;\hat{=}\; \{s \in T\Sigma_{\leqslant}^{*} \mid s \downharpoonright A = s_P \wedge s \downharpoonright B = s_Q \wedge s \downharpoonright (A \cup B) = s\}$$

For an alphabet parallel combination to refuse an event, that event must be refused by one of the component processes. A typical refusal set is thus the union of refusal sets from P and Q, together with any set of events from outside $A \cup B$.

$$\begin{aligned}\mathcal{F}_T[\![P \mathbin{{}_A\|_B} Q]\!]\rho \;\hat{=}\; \{(s, \aleph_P \cup \aleph_Q \cup \aleph_R) \mid \exists\, & s_P, s_Q \bullet \\ & \sigma(\aleph_P) \subseteq A \wedge \sigma(\aleph_Q) \subseteq B \;\wedge \\ & \sigma(\aleph_R) \subseteq \Sigma - (A \cup B) \wedge s \in (s_P \mathbin{{}_A\|_B} s_Q) \;\wedge \\ & (s_P, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \wedge (s_Q, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho\;\}\end{aligned}$$

Simple Parallel

The synchronised parallel operator places two processes in *lockstep*. In the parallel combination $P \parallel Q$ processes P and Q must cooperate on every action that is performed. This operator is thus a special case of alphabet parallelism

$$P \parallel Q \;\equiv\; P \mathbin{{}_\Sigma\|_\Sigma} Q$$

with a simple derived semantics

$$\mathcal{F}_T[\![P \parallel Q]\!]\rho \;\hat{=}\; \{(s, \aleph_P \cup \aleph_Q) \mid (s, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \wedge (s, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho\}$$

Interleaving

The interleaving parallel operator allows two processes to evolve asynchronously. In the parallel combination

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

the two processes are independent of each other; no cooperation is required on any action. As a result, any trace of the process $P \mathbin{|||} Q$ will be an interleaving of two traces, one from each component. The set of possible interleavings of two timed traces u and v is given by

$$u \mathbin{|||} v \;\; \widehat{=} \;\; \{s : T\Sigma^*_{\leq} \mid \forall t : TIME \bullet s \upharpoonright t \cong u \upharpoonright t ^\frown v \upharpoonright t\}$$

Trace s is a possible interleaving of traces u and v if, for all t, an event is in s at time t iff it is in u or v at that time. The equivalence operator $\cong$ is required, as the order of simultaneous events in s may differ from the order of the same events in u or v. Note that we cannot simply require that $s \cong u ^\frown v$, as $u ^\frown v$ need not be a valid timed trace.

$$\begin{aligned} \mathcal{F}_T[\![P \mathbin{|||} Q]\!]\rho \;\; \widehat{=} \;\; \{(s, \aleph) \mid \exists\, s_P, s_Q \bullet \;& s \in (s_P \mathbin{|||} s_Q) \wedge \\ & (s_P, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \wedge \\ & (s_Q, \aleph) \in \mathcal{F}_T[\![Q]\!]\rho\} \end{aligned}$$

An interleaving of two processes will refuse a timed event exactly when both components are unwilling to participate; any refusal set of the parallel combination must be common to both processes.

Communicating Parallel

We can define a hybrid parallel operator which allows processes to interleave on all but a given set of events; in the parallel combination

$$P \underset{C}{\|} Q$$

processes P and Q must cooperate on actions from set C. Other actions may be freely performed by either component, with no need for synchronisation:

$$\begin{aligned} \mathcal{F}_T[\![P \underset{C}{\|} Q]\!] \;\; \widehat{=} \;\; \{(s, \aleph) \mid \exists\, s_P, \aleph_P, s_Q, \aleph_Q \bullet \;& s \in s_P \underset{C}{\|} s_Q \wedge \\ & \aleph \restriction C = (\aleph_P \cup \aleph_Q) \restriction C \wedge \\ & \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C \wedge \\ & (s_P, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \wedge \\ & (s_Q, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho \;\; \} \end{aligned}$$

Events from the interface set C must be performed by both components, while other events are interleaved:

$$u \underset{C}{\|} v \;\widehat{=}\; \{s \mid s \upharpoonright C = u \upharpoonright C = v \upharpoonright C \wedge s \setminus C \in (u \setminus C \;|||\; v \setminus C)\}$$

Event from C are refused if they are refused by at least one of the components; other events must be refused by both components.

The semantic set of this operator is well-defined: it satisfies the axioms of TM_F and may be used in recursive definitions. This is a consequence of the following syntactic equivalence:

$$P \underset{C}{\|} Q \;\equiv\; c\,(l(P)\;{}_A\|_B\; r(Q))$$

where the process relabelling functions l, r, and c are given by:

$$\begin{array}{rclllrcll} l(a) & \widehat{=} & a & \text{if } a \in C & & & & & \\ & & l.a & \text{otherwise} & & c(a) & \widehat{=} & a & \text{if } a \in C \\ r(a) & \widehat{=} & a & \text{if } a \in C & & c(l.a) & \widehat{=} & a & \text{if } a \notin C \\ & & r.a & \text{otherwise} & & c(r.a) & \widehat{=} & a & \text{if } a \notin C \end{array}$$

and

$$\begin{array}{rcl} A & \widehat{=} & l(\Sigma - C) \cup C \\ B & \widehat{=} & r(\Sigma - C) \cup C \end{array}$$

This equivalence is demonstrated in section A.1 of the appendix.

Indexed Parallel

An indexed form of the alphabet parallel operator can be used to define networks of communicating processes:

$$\begin{array}{lll} (\|_{A_i} P_i) \quad i \in 1 \,..\, 2 & \widehat{=} & P_1 \;{}_{A_1}\|_{A_2}\; P_2 \\ (\|_{A_i} P_i) \quad i \in 1 \,..\, n & \widehat{=} & P_n \;{}_{A_n}\|_{\bigcup A_i}\; (\|_{A_i} P_i) \qquad i \in 1 \,..\, n-1 \end{array}$$

Each component P_i may perform only those events which lie in the corresponding interface set A_i. If an event a is present in more than one of these sets, then every process in the set $\{P_i \mid a \in A_i\}$ must cooperate on every occurrence of a.

2.6 Timeouts and Interrupts

Timeout

The expression $P \overset{t}{\triangleright} Q$ denotes a timeout construct, in which control is passed to process Q if P fails to perform any external actions before time t. A delay of δ is associated with the transfer of control:

$$\begin{array}{rcl} \mathcal{F}_T\llbracket P \overset{t}{\triangleright} Q \rrbracket \rho & \mathrel{\widehat{=}} & \{(s, \aleph) \mid begin(s) \leqslant t \wedge (s, \aleph) \in \mathcal{F}_T\llbracket P \rrbracket \rho\} \\ & & \cup \\ & & \{(s, \aleph) \mid begin(s) \geqslant t + \delta \wedge (\langle\rangle, \aleph \upharpoonright t) \in \mathcal{F}_T\llbracket P \rrbracket \rho \\ & & \qquad \wedge \\ & & \qquad (s, \aleph) - (t + \delta) \in \mathcal{F}_T\llbracket Q \rrbracket \rho \; \} \end{array}$$

A trace s is a trace of P if an event is observed before time t, and a trace of Q otherwise. Any event refused before time t must be refused by P. The δ delay may be removed without affecting the validity of the semantic definition, although its presence gives rise to a syntactic equivalence:

$$P \overset{t}{\triangleright} Q \quad \equiv \quad g\,((f(P) \,\Box\, WAIT\; t\,;\, e \to f(Q)) \setminus e)$$

where the process relabelling functions f and g are defined by

$$\begin{array}{rcl} f(a) & \mathrel{\widehat{=}} & f.a \\ g(f.a) & \mathrel{\widehat{=}} & a \end{array}$$

and event e is chosen such that $e \neq f.a$ for any a in Σ. As in the case of the communicating parallel operator, we may use this equivalence to show that the semantics of $P \overset{t}{\triangleright} Q$ is well-defined.

Timed Interrupt

Another useful timing construct is the timed interrupt operator. The expression

$$P \underset{t}{\not\,\,} Q$$

denotes a process in which control is passed from P to Q at time t, regardless of the progress made by P. A delay of δ is associated with the transfer of control.

$$\begin{array}{rcl} \mathcal{F}_T\llbracket P \underset{t}{\not\,\,} Q \rrbracket \rho & \mathrel{\widehat{=}} & \{(s, \aleph) \mid begin(s \upharpoonright t) \geqslant t + \delta \; \wedge (s \upharpoonright t, \aleph \upharpoonright t) \in \mathcal{F}_T\llbracket P \rrbracket \rho \\ & & \qquad \wedge \; (s, \aleph) - (t + \delta) \in \mathcal{F}_T\llbracket Q \rrbracket \rho \; \} \end{array}$$

Any behaviour of this process may be decomposed into behaviours of P and Q by considering the parts of the behaviour that occur before and after time t.

Event Interrupt

Although not strictly a timing construct, the event interrupt operator is easily modelled within a timed context. The expression

$$P \underset{e}{\nabla} Q$$

denotes a process that behaves as P until the first occurrence of event e. Once e is observed, control is passed to process Q, following a small delay of δ. The delay is required by our intnition concerning cause and effect; an initial event from process Q may be enabled by e, and so cannot occur at the same time.

$$\begin{array}{lcl} \mathcal{F}_T[\![P \underset{e}{\nabla} Q]\!]\rho & \mathrel{\widehat=} & \{(s,\aleph) \mid e \notin \sigma(s,\aleph) \wedge (s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho\} \\ & & \cup \\ & & \{(s,\aleph) \mid \exists\, t \bullet \; \begin{array}[t]{l} s \upharpoonright t \downharpoonright e = \langle(t,e)\rangle \;\wedge \\ e \notin \sigma(\aleph \upharpoonright t) \;\wedge \\ begin(s \uparrow t) \geqslant t+\delta \;\wedge \\ (s \upharpoonright t \setminus e, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!]\rho \;\wedge \\ (s,\aleph) - (t+\delta) \in \mathcal{F}_T[\![Q]\!]\rho \;\} \end{array} \end{array}$$

Any behaviour in which e has not been observed must be a behaviour of P; in this case, e must be available. Otherwise, e mnst be observed first at some time t; we may then decompose the behavionr to obtain behaviours of P and Q. A sensible requirement is that $e \notin \sigma(P)$, to avoid the possibility of P interrupting itself.

A more general form of this construct allows interrupts from a set E, with a corresponding choice of consequent processes:

$$\begin{array}{lcl} \mathcal{F}_T[\![P \underset{e \in E}{\nabla} Q_e]\!]\rho & \mathrel{\widehat=} & \{(s,\aleph) \mid E \cap \sigma(s,\aleph) = \{\} \wedge (s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho\} \\ & & \cup \\ & & \{(s,\aleph) \mid \exists\, t\,;\, e \in E \bullet \; \begin{array}[t]{l} s \upharpoonright t \downharpoonright E = \langle(t,e)\rangle \;\wedge \\ E \cap \sigma(\aleph \upharpoonright t) = \{\} \;\wedge \\ begin(s \uparrow t) \geqslant t+\delta \;\wedge \\ (s \upharpoonright t \setminus E, \aleph \upharpoonright t) \in \mathcal{F}_T[\![P]\!]\rho \;\wedge \\ (s,\aleph) - (t+\delta) \in \mathcal{F}_T[\![Q_e]\!]\rho \;\} \end{array} \end{array}$$

In either case, the part of $(s,\aleph)$ pertaining to P is obtained by taking that part of the behaviour that lies before time t, and removing any mention of the interrnpt events. There is no need to delete e or E from the refnsal set, as we know that these events are not refused before time t.

2.7 Interaction

Choice and Delay

Consider the process P defined by

$$\begin{array}{rcl} P & \hat{=} & a \to STOP \\ & & \Box \\ & & WAIT\ 1\ ;\ b \to SKIP \end{array}$$

At any time before time 1, this process is prepared to engage only in event a; the subsequent behaviour is that of the deadlocked process $STOP$. However, if one time unit has elapsed since control was passed to the process, the event b is also available. Now consider the process Q defined by

$$\begin{array}{rcl} Q & \hat{=} & (a \to STOP \\ & & \Box \\ & & WAIT\ 1)\ ;\ b \to SKIP \end{array}$$

This process also offers event a until time 1. Unlike process P, it then withdraws the offer. At time 1, if a has not occurred, the $WAIT$ construct offers the termiuation event. This event is hidden from the environment by the sequential composition operator and occurs immediately, resolving the deterministic choice and passiug control to the process

$$b \to SKIP$$

If event a is offered to the process at time 1, the outcome will be nondeterministic.

In process P, the $WAIT$ operator simply delays the offer of event b, it does not affect the availability of event a; the termination event that enables event b is hidden from the choice construct. In process Q, it acts as a timeout on the offer of a; if this event does not occur at or before time 1, the choice construct terminates and the offer is withdrawn.

Interleaving and Termination

The termination event can be used to interrupt the execution of a process. In the expression

$$(a \to b \to STOP\ |||\ WAIT\ 2)\ ;\ P$$

control is passed to process P after two seconds, regardless of the progress made by the first component of the parallel construct. Note that the subsequent behaviour is independent of the state of the interrupted process. The same is true of the interrupt constructs $P \underset{t}{\not\downarrow} Q$ and $P \underset{e}{\nabla} Q$.

Without an explicit record of the system state, we must use some form of polling if we are to interrupt a process in a reliable way; a process must cooperate on an interrupt event. For example, in the construct below, the interrupt event *break* is disabled after process P performs a *lock* event; the *break* remains disabled until one second after the next *unlock* event.

$$\begin{aligned} P \;\hat{=}\;& WAIT\ 1\ ; lock \xrightarrow{2} unlock \xrightarrow{1} P \\ & \Box \\ & break \xrightarrow{1} SKIP \end{aligned}$$

The deterministic choice ensures that the environment is offered *break* for a full second before *lock* becomes available (again).

Process P permits a *break* event only when the number of *lock* events is equal to the number of *unlock* events; this condition might be a prerequisite for a safe termination of the process. The combination of deterministic choice and delay provides for a simple representation of priority choice in Timed CSP.

Hiding and Synchronisation

Consider the process P defined by

$$P \;\hat{=}\; ((WAIT\ 1\ ; a \rightarrow STOP)\ {}_{\{a\}}\|_{\{a,b\}}\ (b \rightarrow SKIP \Box a \rightarrow STOP)) \setminus a$$

For the first second of its existence, P is prepared to engage in event b and terminate time δ later. Internal event a is not yet possible, as it requires the cooperation of both sides of the parallel combination. At time 1, if event b has not occurred, a becomes available on both sides of the parallel operator. As a hidden event, a occurs as soon as it becomes available, resolving the choice against b.

The possible behaviours of process P are precisely those of the timeout process

$$Q \;\hat{=}\; b \rightarrow SKIP \overset{1}{\rhd} STOP$$

If the environment offers b at time 1, the outcome is nondeterministic.

2.8 Example

We consider the definition of a sensitive vending machine *SVM* which behaves as *VMS* in section 1.5, except that it may fail to dispense a drink if kicked while the coin is dropping. As before, we nse the events *coin* and *coke* to represent the insertion of a coin and the removal of a drink, respectively. Without timing information, our process description is

$$\begin{array}{rcl} SVM & \mathrel{\widehat=} & coin \rightarrow (PAID \,\square\, reset \rightarrow SVM \\ & & \qquad\qquad\quad \sqcap \\ & & \qquad\qquad\quad PAID) \\ \\ PAID & \mathrel{\widehat=} & coke \rightarrow SVM \end{array}$$

The event *reset* represents the effect of a kick on the machine; althongh the machine may be kicked at any time, there is no effect nnless a coin is dropping. Without timing information, we have no way of modelling the progress of the coin inside the machine. The event *reset* is nondeterministically available until a drink is collected.

To add timing information to onr description, we assume that the mechanism becomes sensitive to kicks at time t_1 following the insertion of a coin. After an additional delay of time t_2, the coiu has passed through the mechanism, and the machine may be kicked with impnnity:

$$\begin{array}{rcl} TSVM & \mathrel{\widehat=} & coin \xrightarrow{t_1} (reset \xrightarrow{t_3} TSVM \\ & & \qquad\qquad\quad \overset{t_2}{\rhd} \\ & & \qquad\qquad\quad PAID) \\ \\ PAID & \mathrel{\widehat=} & coke \xrightarrow{t_4} TSVM \end{array}$$

Tbe process *TSVM* offers the event *coin* to the environment. If this event is observed at a time t, then the event *reset* is available between time $t + t_1$ and time $t + t_1 + t_2$. If this event occurs, the machine returns to its initial state after a further time t_3, without offering a drink.

If the event *reset* has not occurred by the time the coin has dropped, then the offer of *reset* is withdrawn by the timeout construct, and the machine offers the environment a drink. The addition of timing information has eliminated the nondeterminism present in the untimed description; the process *TSVM* is a timewise refinement of *SVM*, in the sense of section 5.7.

Chapter 3

Recursive Processes

In [Hoare 85], recursive definitions take the form

$$P \;\hat{=}\; \mu X \bullet F(X)$$

The expression $\mu X \bullet F(X)$ denotes a process that behaves as $F(X)$, with variable X representing a recursive invocation of the process. In [Reed & Roscoe 86], a delay of δ is associated with each recursive call. This has the advantage of making *all* syntactic recursions well-defined: any equation of the form

$$P \;=\; \mu X \bullet F(X)$$

will admit to a unique solution in the semantic model. If we accept that some syntactic recursions will be invalid, we can dispense with this constant delay.

In this chapter, we introduce an *immediate* form of the recursion operator, and give a sufficient condition for the validity of a recursive definition. This treatment of recursion is extended to permit *mutual recursion*: processes may be defined by mutually recnrsive sets of equations. These sets may be arbitrarily large.

3.1 Constructive Terms

The semantics of a Timed CSP term P is a function of the set of term variables appearing in P. For example, the term defined by

$$P \;\hat{=}\; a \xrightarrow{l} X$$

has a semantic set that is parametrised by $\rho[\![X]\!]$, the semantics of X in the current environment. If P appears as the body of a recursive process, then that process has a well-defined semantics if and only if P corresponds to a contraction mapping in the semantic model TM_F. For this to be true, it is sufficient that P is *constructive* for the variable bound by the recursion.

Definition 3.1 If P is a *TCSP* term, possibly including free occurrences of term variable X, then P is *t-constructive* for X if

$$\forall\, t_0 : TIME\, ; \rho : ENV \bullet$$
$$\mathcal{F}_T[\![P]\!]\rho \restriction t_0 + t = \mathcal{F}_T[\![P]\!]\rho[\rho[\![X]\!] \restriction t_0/X] \restriction t_0 + t$$

$\diamondsuit$

If term P is t-constructive for variable X, then the behaviour of P up until a time $t_0 + t$ is independent of the behaviour of X after time t_0. The reader should recall the over-riding notation for environments defined in section 2.3:

$$\begin{array}{rcll} \rho[Y/X][\![Z]\!] & \mathrel{\widehat{=}} & Y & \text{if } Z = X \\ & & \rho[\![Z]\!] & \text{otherwise} \end{array}$$

Definition 3.2 We say that a term P is *constructive* for X if there is a strictly positive time t such that P is t-constructive for X. $\diamondsuit$

Our definition of *constructive* differs from the one used in [Reed 88]. Reed considers that a term P is constructive for X iff

$$\forall\, t_0 : TIME\, ; S, T : TM_F\, ; \rho : ENV \bullet$$
$$S \restriction t_0 = T \restriction t_0 \Rightarrow (\mathcal{F}_T[\![P]\!]\rho[S/X]) \restriction t_0 + t = (\mathcal{F}_T[\![P]\!]\rho[T/X]) \restriction t_0 + t$$

Our definition places a stronger condition upon P and X.

Lemma 3.3 If term P is t-constructive for variable X, then

$$\forall\, t_0 : TIME\, ; S, T : TS_F\, ; \rho : ENV \bullet$$
$$S \restriction t_0 = T \restriction t_0 \Rightarrow (\mathcal{F}_T[\![P]\!]\rho[S/X]) \restriction t_0 + t = (\mathcal{F}_T[\![P]\!]\rho[T/X]) \restriction t_0 + t$$

$\heartsuit$

From the semantic equations for the *TCSP* operators we can derive a number of useful results[1] about constructive terms.

Lemma 3.4 For any X and t,

1. *STOP*, *SKIP*, $\perp$, and *WAIT* t_0 are all t-constructive for X
2. X is *0*-constructive for X, and t-constructive for $Y \neq X$
3. $\mu\, X \bullet P$ is t-constructive for X

$\heartsuit$

[1]An example derivation is included in appendix A.1

Lemma 3.5 If P is t-constructive for X,

1. $a \xrightarrow{t_0} P$ and $WAIT\ t_0\ ;\ P$ are $(t + t_0)$-constructive for X
2. $\mu\, Y \bullet P$, $P \setminus A$, $f(P)$, $f^{-1}(P)$ are all t-constructive for X
3. P is t'-constructive for X, for any $t' < t$

♡

Lemma 3.6 If P is t_1-constructive for X and Q is t_2-constructive for X,

1. $P \Box Q$, $P \sqcap Q$, $P\ ;\ Q$, $P \mid\mid\mid Q$, $P \parallel Q$, $P\ {}_A\!\|_B\ Q$
   are all $min\{t_1, t_2\}$-constructive for X
2. $P \overset{t}{\rhd} Q$ and $P \underset{t}{\overset{i}{}} Q$ are both $min\{t_1, t_2 + t\}$-constructive for X
3. $P; Q$ and $P \underset{e}{\nabla} Q$ are $\min\{t_1, t_2 + \delta\}$-constructive for X

♡

Observe that all Timed CSP terms are *0*-constructive for any process variable.

Restrained Terms

A sequential composition of terms is also constructive if the first term is constructive, and cannot terminate immediately. We say that a term P is *t-restrained* if it cannot terminate within time t:

Definition 3.7 If P is a *TCSP* term, then

$$P \text{ is } t\text{-restrained} \quad \Leftrightarrow \quad (s \in traces(P) \wedge end(s) < t) \Rightarrow \checkmark \notin \sigma(s)$$

for any instantiation of free variables in P. ◇

A Timed CSP process is t-restrained if the event ✓, signalling successful termination, is not included in the set of events that may be observed before time t. A *TCSP* term P is t-restrained if this condition holds whatever the values of any free variables in P. In particular, we must be able to replace these with the termination process *SKIP*. From the semantic equations of the TCSP operators, we can obtain a number of simple results about restrained terms.

Lemma 3.8 For any time t,

1. *SKIP* is *0*-restrained
2. *WAIT t* is t-restrained
3. *STOP* and $\perp$ are ∞-restrained

♡

Lemma 3.9 If term P is t-restrained,

1. $a \xrightarrow{t_0} P$ and $WAIT\ t_0 ; P$ are $(t + t_0)$-restrained
2. $\mu\, Y \bullet P$, $P \setminus A$, $P ; X$, $f(P)$, and $f^{-1}(P)$ are all t-restrained
3. P is t'-restrained, for any $t' < t$

$\heartsuit$

Lemma 3.10 If P is t_1-restrained, and Q is t_2-restrained,

1. $P \,\square\, Q$, $P \sqcap Q$, $P \mathbin{{}_A\|_B} Q$, and $P \mathbin{|||} Q$ are all $min\{t_1, t_2\}$-restrained
2. $P ; Q$ is $(t_1 + t_2)$-restrained
3. $P \overset{t}{\rhd} Q$ and $P \underset{t}{\overset{i}{\lightning}} Q$ are both $min\{t_1, t_2 + t\}$-restrained
4. $P \underset{t}{\nabla} Q$ is $min\{t_1, t_2 + \delta\}$-restrained
5. $P \parallel Q$ is $max\{t_1, t_2\}$-restrained
6. $\checkmark \notin A \cup B \Rightarrow P \mathbin{{}_A\|_B} Q$ is ∞-restrained
7. $\checkmark \in A \cap B \Rightarrow P \mathbin{{}_A\|_B} Q$ is $max\{t_1, t_2\}$-restrained

$\heartsuit$

Using the notion of a restrained term, we can add a further result to our list of lemmata about constructive terms:

Lemma 3.11 If term P is t-restrained and t-constructive for X, then the term $P ; Q$ is t-constructive for X, for any t and Q. $\heartsuit$

3.2 Recursive Processes

We extend our syntax with two single fixed point recursion operators:

$$P \quad ::= \quad \mu\, X \bullet P \;\;\Big|\;\; \mu\, X \circ P$$

The first of these associates a delay of time δ with each recursive call, while the second transfers control to a recnrsive invocation of the process immediately upon reaching an instance of variable X. We will refer to these operators as *delayed* and *immediate* recursion, respectively.

We may regard the semantics of a term P with free variable X and environment ρ as a function defined upon TS_F. This function maps a set of failures S to the semantics of P evaluated in an environment $(\rho[S/X])$ obtained by associating variable X with the set S.

Definition 3.12 If P is a *TCSP* term, and X and Y are variables such that Y does not occur free in P, then

$$M(X,P)\rho \;\hat{=}\; \lambda\, Y \bullet \mathcal{F}_T[\![P]\!]\rho[Y/X]$$

$\diamond$

To give a semantics to the delayed recursion operator, we consider the composition of this mapping with the function W_δ.

Definition 3.13 If P is a *TCSP* term, and X and Y are variables such that Y does not occur free in P, then

$$M_\delta(X,P)\rho \;\hat{=}\; W_\delta \cdot \lambda\, Y \bullet \mathcal{F}_T[\![P]\!]\rho[Y/X]$$

where W_δ is the mapping defined by

$$W_\delta \;\hat{=}\; \lambda\, Y \bullet \mathcal{F}_T[\![WAIT\ \delta\,; X]\!]\rho[Y/X]$$

$\diamond$

The environment parameter provides a binding for any free variables remaining in term P, and the definition of W_δ reflects the delay associated with this form of recursion—observe that W_δ does not depend upon the choice of environment ρ. We may now give the semantics of the recursion operators.

$$\mathcal{F}_T[\![\mu\, X \circ P]\!]\rho \;\hat{=}\; \text{the unique fixed point of the mapping } M(X\,,P)\rho$$

$$\mathcal{F}_T[\![\mu\, X \bullet P]\!]\rho \;\hat{=}\; \text{the unique fixed point of the mapping } M_\delta(X\,,P)\rho$$

Reed has shown [Reed 88] that the mapping $M_\delta(X,P)\rho$ will always have a unique fixed point in TM_F, and hence that the semantics of delayed recursion is always well-defined. Tbis result does not hold for the immediate recursion operator. We will show that the semantics of immediate recursion is well-defined if term P is constructive for variable X.

Lemma 3.14 If term P is constructive for process variable X then the mapping $M(X,P)\rho$ is a contraction mapping on the space of sets of failures TS_F. $\heartsuit$

Proof A mapping F in TS_F is a contraction mapping if and only if

$$\exists\, r < 1 \;\bullet\; \forall\, S, T : TS_F \bullet d(F(S),F(T)) \leqslant r.d(S,T)$$

where d is the metric defined by

$$d(S,T) \;\hat{=}\; inf(\{2^{-k} \mid S \upharpoonright k = T \upharpoonright k\} \cup \{1\})$$

Now take any two processes S and T in TS_F. If $S = T$ then $F(S) = F(T)$ and both sides of the above inequality are zero. Else, let

$$d(S, T) \;=\; 2^{-k}$$

If we take F to be the mapping $M(X, P)\rho$, then

$$\forall R : TS_F \;\bullet\; F(R) = \mathcal{F}_T[\![P]\!]\rho[R/X]$$

From the definition of constructive and lemma 3.3, we know that there is a strictly positive time t such that

$$S \upharpoonright k = T \upharpoonright k \;\Rightarrow\; F(S) \upharpoonright k + t = F(T) \upharpoonright k + t$$

for any S and T in TS_F. From this, we obtain

$$d(F(S), F(T)) \;\leqslant\; 2^{-(k+t)} \;=\; 2^{-t}.d(S, T)$$

We note that $2^{-t} < 1$ as t is strictly positive, and conclude that F is a contraction mapping in TS_F. $\square$

We have established that the mapping corresponding to a constructive term is a contraction mapping on TS_F. To establish that such a mapping has a unique fixed point, we require the following result from [Sutherland 75].

The Banach Fixed Point Theorem If (M, d) is a complete metric space and $F : M \rightarrow M$ is a contraction mapping, then F has a unique fixed point $\mathit{fix}(F)$. Furthermore, for all S in M, $\mathit{fix}(F) = \lim_{n\rightarrow\infty} F^n(S)$. $\diamondsuit$

The semantic model TM_F is a subset of TS_F, and both are complete metric spaces under the metric d defined in section 2.2. A contraction mapping on TS_F is therefore a contraction mapping on the complete subspace TM_F, hence

Lemma 3.15 If $F : TS_F \rightarrow TS_F$ is a contraction mapping which maps TM_F into TM_F, then F has a unique fixed point $\mathit{fix}(F)$ in TM_F. $\heartsuit$

Any function derived from the semantics of a *TCSP* term will preserve the axioms of the semantic model, mapping TM_F into TM_F. We may combine lemmata 3.14 and 3.15 to obtain the required result:

Theorem 3.16 If term P is constructive for variable X, then the semantics

$$\mathcal{F}_T[\![\mu X \circ P]\!]\rho$$

is well-defined for all environments ρ. $\heartsuit$

The semantics of immediate recursion gives rise to the familiar equivalence

Theorem 3.17

$$\mu X \circ F(X) \;\equiv\; F(\mu X \circ F(X))$$

♡

This result justifies the use of recursive equations as process definitions. For example, a process that is willing to perform the event a at one second intervals may be defined by the equation

$$P \;=\; a \xrightarrow{1} P$$

This equational definition is equivalent to the following definition of P using the immediate recursion operator:

$$P \;\widehat{=}\; \mu X \circ a \xrightarrow{1} X$$

In fact, we can easily prove that

Corollary 3.18 If $\mu X \circ F(X)$ is well-defined, then

$$P \;=\; F(P) \quad \text{if and only if} \quad P \;=\; \mu X \circ F(X)$$

♡

The equational style is more concise, especially in the case of mutual recursion. Indeed, we cannot reasonably write an infinite mutual recursion using μ-notation.

It should be remembered that this result (theorem 3.17) does not hold for the delayed recursion operator; we have instead that

$$\mu X \bullet F(X) \;\equiv\; F(\mathit{WAIT}\,\delta\,;(\mu X \bullet F(X)))$$

which is inconsistent with the use of equations to define recursive processes. For example, there is no delayed recursive process which will satisfy the recursive equation $P = a \rightarrow P$.

3.3 Mutual Recursion

We will now consider processes defined by sets of mutually recursive equations. The definition of *constructive* in section 3.1 extends in a natural way to vectors of terms and variables, and we are able to exhibit a sufficient condition for a syntactic mutual recursion to have a well-defined semantics.

Syntax

A *TCSP* term P may be defined by a vector of mutually recursive equations

$$P \;\hat{=}\; (X_i = P_i)_j \qquad i \in I$$

with an initial index j to indicate the starting point of the recursion. We will employ a simple vector notation for terms:

$$\underline{P} \;=\; \langle P_1, P_2, \ldots\; P_n, \ldots \rangle$$

The sets used to index these vectors need not be finite. Using this notation, we can write our equation vectors in the form $(\mu\, \underline{X} \circ \underline{P})$.

As an example, consider the process algebra representation of a device that has two states: ON and OFF. This device may produce a *beep* as often as once a second when ON. The two states correspond to the two mutually recursive equations below.

$$\begin{aligned} ON &= (beep \xrightarrow{1} ON) \,\Box\, (off \xrightarrow{1} OFF) \\ OFF &= on \xrightarrow{3} ON \end{aligned}$$

This may be considered as a single recursive equation, on a vector of process variables $\langle ON, OFF\rangle$. The device is then modelled by the component of the vector corresponding to the initial state OFF. Alternatively, we may represent the device as a single recursive process:

$$OFF \;\hat{=}\; \mu\, X \circ on \xrightarrow{3} (\mu\, Y \circ (beep \xrightarrow{1} Y) \,\Box\, (off \xrightarrow{1} X))$$

This nested recursion defines the same process as the first component of the mutual recursion above, and falls within the standard syntax for Timed CSP given in [Reed 88]. In practice, it will be more convenient to represent mutually recursive processes using equation sets, particularly when the set of named states is infinite. For example, consider the case of an integer store STO. Initially, the store is willing to input an integer value:

$$STO \;\hat{=}\; in?x \xrightarrow{1} STO_x$$

Thereafter, it is prepared to accept another input, or output the current value stored, as often as once a second:

$$\begin{array}{rcl} STO_x & \widehat{=} & in?y \xrightarrow{1} STO_y \\ & & \Box \\ & & out!x \xrightarrow{1} STO_x \end{array}$$

This is an infinite set of mutually recursive equations, where STO_x models the state of the store containing x.

With the delayed form of recursion, mutually recursive definitions should not be written with an equality symbol, as the left- and right-hand sides do not represent equivalent processes. We use the reverse implication symbol, $\Leftarrow$, to indicate that there is a delay of δ involved in the unfolding of the recursion. The above example would be written as

$$\begin{array}{rcl} STO & \widehat{=} & in?x \xrightarrow{1} STO_x \\ STO_x & \Leftarrow & in?y \xrightarrow{1-\delta} STO_y \\ & & \Box \\ & & out!x \xrightarrow{1-\delta} STO_x \end{array}$$

This form of recursion is always valid in the semantic model. However, the immediate recursion operator makes recursive definitions easier to understand, and allows the user to choose the point and duration of any delay.

Semantics

Consider an equation set $\langle X_i = P_i \rangle_j$, where the indices i and j are drawn from set I. The semantic domain required to model a solution is TS_F^I; this is a product space with one copy of the model TS_F for each $i \in I$. For any I, this domain is a complete metric space, with the following distance metric on vectors.

$$\underline{d}(\underline{V}, \underline{W}) \ \widehat{=}\ sup\{d(V_i, W_i) \mid i \in I\}$$

To construct a semantic function for vectors of terms, we extend the use of environments to include mappings from vectors of variables to vectors of processes.

$$\rho[\![\underline{X}]\!] \ \widehat{=}\ \langle \rho[\![X_i]\!] \mid i \in I \rangle$$

where I is the indexing set of vector $\underline{X}$. We overload the mapping notation defined in section 3.2 with

Definition 3.19 If $\underline{P}$ a vector of *TCSP* terms, and $\underline{X}$ and $\underline{Y}$ are vectors of variables, all indexed by set I, and no component of $\underline{Y}$ occurs free in $\underline{P}$ then

$$M(\underline{X}, \underline{P})\rho \ \widehat{=} \ \lambda\, \underline{Y} \bullet \mathcal{F}_T[\![\underline{P}]\!]\rho[\underline{Y}/\underline{X}]$$

is the mapping on TS_F^I corresponding to $\underline{X}$ and $\underline{P}$. ◇

Definition 3.20 If $\underline{P}$ is a vector of *TCSP* terms, then

$$\mathcal{F}_T[\![\langle X_i = P_i\rangle_j]\!]\rho \ \widehat{=} \ S_j \text{ where } \underline{S} \text{ is a fixed point of } M(\underline{X}, \underline{P})\rho$$

◇

This semantics is well-defined when all fixed points of the mapping $M(\underline{X}, \underline{P})\rho$ agree on the j component. Clearly, it is enough that this mapping has a unique fixed point. For this to be true, it is sufficient that the vector of terms $\underline{P}$ is *constructive* for the vector of variables.

Constructive Vectors

A partial ordering $\prec$ on a set S is a well-ordering if and only if there are no infinite descending sequences $\langle s_i \mid i : \mathbb{N}\rangle$ such that $\forall\, i : \mathbb{N} \bullet s_{i+1} \prec s_i$. We define the initial segment of an element of i in the usual way.

Definition 3.21 If $\prec$ is a partial ordering on I, and i is an element of I, then the initial segment of i in $(I, \prec)$ is defined by

$$\text{seg}(i) \ \widehat{=} \ \{j : I \mid j \prec i\}$$

◇

For tbe mapping $M(\underline{X}, \underline{P})\rho$ to have a unique fixed point, it is sufficient that the vector of terms $\underline{P}$ is *constructive* for the vector of variables $\underline{X}$.

Definition 3.22 A vector of terms $\underline{P}$ is t-constructive for a vector of process variables $\underline{X}$ if there is a well-ordering $\prec$ of the indexing set I such that

$$\forall j, i : I \bullet j \notin \text{seg}(i) \Rightarrow P_i \text{ is } t\text{-constructive for } X_j$$

◇

Definition 3.23 A vector of terms $\underline{P}$ is *constructive* for a vector of process variables $\underline{X}$ if there is a strictly positive time t such that $\underline{P}$ is t-constructive for $\underline{X}$. ◇

If this condition is met then the only possible unguarded recursive calls in term P_i correspond to variables X_j where $j \prec i$. Thus any sequence of unguarded recursive calls is indexed by a descending seqnence from the set I, and must therefore be finite. Any particnlar behaviour of the process is generated by a finite number of recursive calls, and an infinite nnmber of recursive calls in a finite time is impossible.

In many applications, it is not necessary to identify a well-ordering of the index set I. If all recnrsive calls are guarded by a single positive time t, then any well-ordering of I will be enough to show that the vector of terms is constructive for the vector of variables. In this case, we say that the vector of terms is *uniformly constructive*. Formally,

Definition 3.24 A vector of terms $\underline{P}$ is uniformly t-constructive for a vector of variables $\underline{X}$ if P_i is t-constructive for all X_j. $\diamondsuit$

Definition 3.25 A vector of terms $\underline{P}$ is uniformly constructive for a vector of variables $\underline{X}$ if there exists a positive time t such that $\underline{P}$ is uniformly t-constructive for $\underline{X}$. $\diamondsuit$

Observe that any uniformly constructive vector of terms is constructive. In this case $M(\underline{X},\underline{P})\rho$ will be a contraction mapping in the semantic model TS_F^I.

We have defined *constructive* for vectors in a component-wise fashion. That a vector of terms is constructive for a vector of variables can be established by a case analysis on pairs (X_i, P_j) in our equation set, a relatively simple procedure. We will show that this is a sufficient condition for the semantics of a mutual recursion to be well-defined. First, we must demonstrate that our pointwise definitions are enough to establish the corresponding vector results.

Theorem 3.26 (Finite Dependency Theorem) If P is a $TCSP$ term, possibly containing free occurrences of process variables drawn from the set $\{X_i \mid i \in I\}$, and ρ is an environment, then

$$\begin{aligned}(s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho \ \Rightarrow\ & \exists\, N : \mathbb{F}\, I \bullet \forall\, \rho' : ENV \bullet \\ & \qquad (\forall\, i : N \bullet \rho[\![X_i]\!] = \rho'[\![X_i]\!]) \Rightarrow (s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho'\end{aligned}$$

$\heartsuit$

The presence of a given behaviour $(s,\aleph)$ in the semantic set of a term P depends only upon the values of a finite set of variables N, even if the term is an infinite mutnal recnrsion. We may change the environment of the term without removing the behaviour, providing that we preserve the values of the variables in N. A detailed proof of this theorem is presented in section A.2 of the appendix.

We may restate this theorem in a more applicable form, using the over-riding notation for environments:

Corollary 3.27 If P is a *TCSP* term, possibly containing free occurrences of process variables drawn from the set $\{X_i \mid i \in I\}$, and ρ and ρ' are environments, then

$$(s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \;\Rightarrow\; \exists N : \mathbb{F}\, I \bullet \forall \rho' \bullet (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho'[\rho[\![X_i]\!]/X_i \mid i \in N]$$

$\heartsuit$

We say that a term P is t-constructive for a vector of variables $\underline{X}$ if the semantics of P up until a time t_0 is independent of the behaviour of every component of $\underline{X}$ after time t_0. This is a simple extension of definition 3.1.

Definition 3.28 If P is a *TCSP* term, then P is ***t-constructive*** for a vector of variables $\underline{X}$, indexed by set I, iff

$$\begin{aligned} &\forall t_0 : \mathit{TIME}\,; \rho : \mathit{ENV} \bullet \\ &\qquad \mathcal{F}_T[\![P]\!]\rho \restriction t_0 + t = \mathcal{F}_T[\![P]\!]\rho[\rho[\![X_i]\!] \restriction t_0/X_i \mid i \in I] \restriction t_0 + t \end{aligned}$$

$\diamondsuit$

With this definition, a simple induction upon the length of finite vector $\underline{X}$ is enough to establish the following lemma:

Lemma 3.29 If P is t-constructive for each of $\{X_i \mid i \in N\}$, and N is a finite set, then P is t-constructive for the vector $\underline{X}$ indexed by N. $\heartsuit$

We may combine this result with the Finite Dependency Theorem to obtain the theorem below, which will allow us to obtain vector results from our pointwise definitions.

Theorem 3.30 If P is t-constructive for each of $\{X_i \mid i \in I\}$ then P is t-constructive for the vector $\underline{X}$ indexed by I. $\heartsuit$

Proof To show that P is t-constructive for vector $\underline{X}$, we must show that for any time t_0 and environment ρ,

$$\mathcal{F}_T[\![P]\!]\rho \restriction t_0 + t \;=\; \mathcal{F}_T[\![P]\!]\rho[\rho[\![X_i]\!] \restriction t_0/X_i \mid i \in I] \restriction t_0 + t$$

If we take $(s, \aleph)$ to be an element of the left-hand side, we may apply the corollary to the Finite Dependency Theorem, yielding

$$\exists N : \mathbb{F}\, I \bullet \forall \rho' \;\bullet\; (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho'[\rho[\![X_i]\!]/X_i \mid i \in N] \restriction t_0 + t$$

We take ρ' to be the environment

$$\rho[(\rho[\![X_i]\!] \restriction t_0)/X_i \mid i \in I \wedge i \notin N]$$

and appeal to lemma 3.29. We have assumed that P is t-constructive for each X_i, so it must be t-constructive for the finite vector $\langle X_i \mid i \in N\rangle$. Expanding definition 3.28, we discover that

$$\begin{aligned}\mathcal{F}_T[\![P]\!]\rho'[\rho[\![X_i]\!]/X_i \mid i \in N] \restriction t_0+t &= \mathcal{F}_T[\![P]\!]\rho'[\rho[\![X_i]\!] \restriction t_0/X_i \mid i \in N] \restriction t_0+t \\ &= \mathcal{F}_T[\![P]\!]\rho[\rho[\![X_i]\!] \restriction t_0/X_i \mid i \in I] \restriction t_0+t\end{aligned}$$

and hence that $(s, \aleph)$ is an element of the right-hand side. A symmetric argument will establish the converse, completing the proof of the theorem. □

We may use this theorem to show that any mutual recursiou in which the vector of terms is constructive for the vector of variables has a well-defined semantics.

Theorem 3.31 (Unique Fixed Point Theorem) If vector of terms $\underline{P}$ is constructivefor vector of variables $\underline{X}$, then the mapping $M(\underline{X}, \underline{P})\rho$ has a unique fixed point in TS_F^I. ♡

Although the proof is quite involved, it is both important and instructive.

Proof We begin by defining a secondary vector of terms $\underline{Q}$ by transfinite recursion. We show that the mapping $M(\underline{X}, \underline{Q})\rho$ has a unique fixed point, and that this is also a fixed point of the mapping $M(\underline{X}, \underline{P})\rho$. We complete the proof by demonstrating that this fixed point is unique.

The vector of *TCSP* terms $\underline{Q}$ is defined by

$$Q_i \;\widehat{=}\; P_i[Q_j/X_j \mid j \in \mathrm{seg}(i)]$$

The i component of $\underline{Q}$ is that of $\underline{P}$, with the following modification: we replace everyvariable with an index lower than i with the corresponding component of $\underline{Q}$.

Lemma A The vector $\underline{Q}$ is well-defined. This is an instance of the following theorem schema, established in [Enderton 77].

Transfinite Recursion Theorem If $\prec$ is a well-ordering on I, and for any function f there is a unique y such that $\varphi(f, y)$ is true, then there exists a unique function F such that

$$\forall i : I \bullet \varphi(F \downharpoonleft \mathrm{seg}(i), F(i))$$

and the domain of F is the whole of I. ◇

We use this theorem to construct a function F of type $I \to (I \to TCSP)$. That is, a function from indices to vectors of Timed CSP terms. We choose the formula φ carefully:

$$\varphi(f, y) \;\hat{=}\; y = \underline{P}[f(j)_j / X_j \mid j \in \mathrm{dom}(f)]$$

This formula holds of (f, y) exactly when y is the vector obtained from $\underline{P}$ by replacing every occurrence of variable X_j, for every j in the domain of function f. Each term X_j is replaced with the j component of the vector $f(j)$. It is clear that this defines a unique y for every function f. If F is the φ-constructed function, then we define

$$Q_i \;\hat{=}\; F(i)_i$$

yielding the required vector $\underline{Q}$

$$\begin{aligned} & F(i)_i &=& \; (\underline{P}[(F \downharpoonright \mathrm{seg}(i))(j)_j / X_j \mid j \in \mathrm{dom}(F \downharpoonright \mathrm{seg}(i))])_i \\ \Rightarrow\; & F(i)_i &=& \; (\underline{P}[Q_j / X_j \mid j \in \mathrm{seg}(i)])_i \\ \Rightarrow\; & Q_i &=& \; P_i[Q_j / X_j \mid j \in \mathrm{seg}(i)] \end{aligned}$$

Lemma B The mapping $M(\underline{X}, \underline{Q})\rho$ is a contraction mapping in TS_F^I, and hence has a unique fixed point. By analogy with theorem 3.16, it is enough to show that there exists a strictly positive time t such that

$$\begin{aligned} & \forall\, \underline{S}, \underline{T} : TS_F^I \;;\; t_0 : TIME \;\bullet \\ & \qquad \underline{S} \upharpoonright t_0 = \underline{T} \upharpoonright t_0 \Rightarrow (M(\underline{X}, \underline{Q})\rho\, \underline{S}) \upharpoonright t_0 + t = (M(\underline{X}, \underline{Q})\rho\, \underline{T}) \upharpoonright t_0 + t \end{aligned}$$

To prove this, we will assume that $\underline{S} \upharpoonright t_0 = \underline{T} \upharpoonright t_0$ and deduce the consequent above, which is equivalent to

$$\mathcal{F}_T[\![\underline{Q}]\!]\rho[\underline{S}/\underline{X}] \upharpoonright t_0 + t \;=\; \mathcal{F}_T[\![\underline{Q}]\!]\rho[\underline{T}/\underline{X}] \upharpoonright t_0 + t$$

We will employ the following result from [Enderton 77].

Transfinite Induction Principle If $\prec$ is a well-ordering on set I, and J is a subset of I with the property

$$\forall\, i : I \;\bullet\; \mathrm{seg}(i) \subseteq J \Rightarrow i \in J$$

then J coincides with I. ◇

We define J to be the set

$$J \;\hat{=}\; \{i : I \mid \mathcal{F}_T[\![Q_i]\!]\rho[\underline{S}/\underline{X}] \restriction t_0 + t = \mathcal{F}_T[\![Q_i]\!]\rho[\underline{T}/\underline{X}] \restriction t_0 + t\}$$

and assume that $\mathrm{seg}(i) \subseteq J$. We have to show that $i \in J$. To show this, we must show that

$$\mathcal{F}_T[\![Q_i]\!]\rho_S \restriction t_0 + t \;=\; \mathcal{F}_T[\![Q_i]\!]\rho_T \restriction t_0 + t$$

where

$$\begin{aligned} \rho_S &\;\hat{=}\; \rho[\underline{S}/\underline{X}] \\ \rho_T &\;\hat{=}\; \rho[\underline{T}/\underline{X}] \end{aligned}$$

From the definition of vector $\underline{Q}$, we obtain

$$\begin{aligned} \mathcal{F}_T[\![Q_i]\!]\rho_S &= \mathcal{F}_T[\![P_i[Q_j/X_j \mid j \in \mathrm{seg}(i)]\!]\rho_S \\ &= \mathcal{F}_T[\![P_i]\!]\rho_S[\mathcal{F}_T[\![Q_j]\!]\rho_S/X_j \mid j \in \mathrm{seg}(i)] \end{aligned}$$

A similar argument applies for ρ_T, and if we let

$$\begin{aligned} \rho'_S &\;\hat{=}\; \rho_S[\mathcal{F}_T[\![Q_j]\!]\rho_S/X_j \mid j \in \mathrm{seg}(i)] \\ \rho'_T &\;\hat{=}\; \rho_T[\mathcal{F}_T[\![Q_j]\!]\rho_T/X_j \mid j \in \mathrm{seg}(i)] \end{aligned}$$

we reduce our proof obligation to

$$\mathcal{F}_T[\![P_i]\!]\rho'_S \restriction t_0 + t \;=\; \mathcal{F}_T[\![P_i]\!]\rho'_T \restriction t_0 + t$$

We assume that $(s, \aleph)$ is an element of the left-hand side, and apply the first corollary to the Finite Dependency Theorem, corollary 3.27. Then there exists a finite set N such that

$$(s, \aleph) \;\in\; \mathcal{F}_T[\![P_i]\!]\rho'_T[\rho'_S[\![X_k]\!]/X_k \mid k \in N]$$

We partition the set N into two sets, and give names to two useful vectors

$$\begin{aligned} A &\;\hat{=}\; N \cap \mathrm{seg}(i) \\ B &\;\hat{=}\; N - \mathrm{seg}(i) \\ Y &\;\hat{=}\; \langle \mathcal{F}_T[\![Q_j]\!]\rho_S \mid j \in I\rangle \\ Z &\;\hat{=}\; \langle \mathcal{F}_T[\![Q_j]\!]\rho_T \mid j \in I\rangle \end{aligned}$$

and define, for each vector $\underline{V}$ in $\{\underline{X}, \underline{Y}, \underline{Z}\}$

$$\begin{aligned} V_A &\;\hat{=}\; \langle V_i \mid i \in A\rangle \\ V_B &\;\hat{=}\; \langle V_i \mid i \in B\rangle \end{aligned}$$

Our inductive hypothesis can then be re-written as

$$\forall k : \mathrm{seg}(i) \bullet \quad Y_k \upharpoonright t_0 + t = Z_k \upharpoonright t_0 + t$$

which implies that

$$Y_A \upharpoonright t_0 + t \; = \; Z_A \upharpoonright t_0 + t$$

All *TCSP* terms are *0*-constructive for any variable, so by Theorem 3.30 all *TCSP* terms are *0*-constructive for any vector. Applying this result to the terms Q_j, given that the vectors $\underline{S}$ and $\underline{T}$ agree up until time t_0, we obtain

$$Y_B \upharpoonright t_0 = Z_B \upharpoonright t_0$$

We recall our assumption about behaviour $(s, \aleph)$:

$$\begin{aligned}(s,\aleph) \;&\in\; \mathcal{F}_T[\![P_i]\!]\rho'_T[\rho'_S[\![X_k]\!]/X_k \mid k \in A][\rho'_S[\![X_k]\!]/X_k \mid k \in B] \\ &=\; \mathcal{F}_T[\![P_i]\!]\rho'_T[\mathcal{F}_T[\![Q_k]\!]\rho_S/X_k \mid k \in A][\mathcal{F}_T[\![Q_k]\!]\rho_S/X_k \mid k \in B] \\ &=\; \mathcal{F}_T[\![P_i]\!]\rho'_T[Y_A/X_A][Y_B/X_B]\end{aligned}$$

Again, any *TCSP* term is *0*-constructive for any vector, so P_i is *0*-constructive for X_A. Hence

$$\mathcal{F}_T[\![P_i]\!]\rho'_T[Y_A/X_A][Y_B/X_B] \upharpoonright t_0 + t \; = \; \mathcal{F}_T[\![P_i]\!]\rho'_T[Z_A/X_A][Y_B/X_B] \upharpoonright t_0 + t$$

Further, vector $\underline{P}$ is t-constructive, so term P_i is t-constructive for any X_j with $j \notin \mathrm{seg}(i)$. By Theorem 3.30, P_i is t-constructive for vector X_B. Hence

$$\begin{aligned}\mathcal{F}_T[\![P_i]\!]\rho'_T[Z_A/X_A][Y_B/X_B] \upharpoonright t_0 + t \;&=\; \mathcal{F}_T[\![P_i]\!]\rho'_T[Z_A/X_A][Z_B/X_B] \upharpoonright t_0 + t \\ &=\; \mathcal{F}_T[\![P_i]\!]\rho'_T \upharpoonright t_0 + t\end{aligned}$$

Remembering that $end(s, \aleph) < t_0 + t$, we have established that

$$(s, \aleph) \;\in\; \mathcal{F}_T[\![P_i]\!]\rho'_T \upharpoonright t_0 + t$$

The argument is symmetrical in S and T, and hence

$$\begin{aligned}\mathcal{F}_T[\![P_i]\!]\rho'_S \upharpoonright t_0 + t \;&=\; \mathcal{F}_T[\![P_i]\!]\rho'_T \upharpoonright t_0 + t \\ (\Rightarrow \; \mathcal{F}_T[\![Q_i]\!]\rho_S \upharpoonright t_0 + t \;&=\; \mathcal{F}_T[\![Q_i]\!]\rho_T \upharpoonright t_0 + t\;)\end{aligned}$$

and we see that $i \in J$. By transfinite induction

$$\mathcal{F}_T[\![\underline{Q}]\!]\rho[\underline{S}/\underline{X}] \upharpoonright t_0 + t \; = \; \mathcal{F}_T[\![\underline{Q}]\!]\rho[\underline{T}/\underline{X}] \upharpoonright t_0 + t$$

We conclude that $M(\underline{X}, \underline{Q})\rho$ is a contraction mapping, with a unique fixed point.

Lemma C The unique fixed point of $M(\underline{X},\underline{Q})\rho$ is a fixed point of $M(\underline{X},\underline{P})\rho$. To see this, let $\underline{S}$ be the unique fixed point of $M(\underline{X},\underline{Q})\rho$, and observe that

$$\begin{aligned}
&(M(\underline{X},\underline{P})\rho\,(\underline{S}))_i\\
&\quad= (\mathcal{F}_T[\![\underline{P}]\!]\rho[\underline{S}/\underline{X}])_i\\
&\quad= (\mathcal{F}_T[\![\underline{P}]\!]\rho[(M(\underline{X},\underline{Q})\rho\,(\underline{S}))_j/X_j \mid j \in \mathrm{seg}(i)][S_j/X_j \mid j \notin \mathrm{seg}(i)])_i\\
&\quad= \mathcal{F}_T[\![P_i]\!]\rho[(M(\underline{X},\underline{Q})\rho\,(\underline{S}))_j/X_j \mid j \in \mathrm{seg}(i)][S_j/X_j \mid j \notin \mathrm{seg}(i)]\\
&\quad= \mathcal{F}_T[\![P_i]\!]\rho[\mathcal{F}_T[\![Q_j]\!]\rho[\underline{S}/\underline{X}]/X_j \mid j \in \mathrm{seg}(i)][S_j/X_j \mid j \notin \mathrm{seg}(i)]\\
&\quad= \mathcal{F}_T[\![P_i[Q_j/X_j \mid j \in \mathrm{seg}(i)]]\!]\rho[\underline{S}/\underline{X}]\\
&\quad= \mathcal{F}_T[\![Q_i]\!]\rho[\underline{S}/\underline{X}]\\
&\quad= (M(\underline{X},\underline{Q})\rho\,(\underline{S}))_i\\
&\quad= S_i
\end{aligned}$$

Hence, as this holds for every $i \in I$, we have that

$$M(\underline{X},\underline{P})\rho\,(\underline{S}) \;=\; \underline{S}$$

establishing that $\underline{S}$ is a fixed point of $M(\underline{X},\underline{P})\rho$.

Lemma D The above fixed point ($\underline{S}$) is the only fixed point of $M(\underline{X},\underline{P})\rho$. We know that there is a positive time t such that $\underline{P}$ is t-constructive for $\underline{X}$. Let $\underline{T}$ be an arbitrary fixed point of $M(\underline{X},\underline{P})\rho$, and define a counterexample set C

$$\begin{aligned}
C \;\hat{=}\; \{k : I \mid \exists\, t_0 \bullet\; & j \in \mathrm{seg}(k) \Rightarrow T_j \upharpoonright (t_0 + t) = S_j \upharpoonright (t_0 + t)\\
& \wedge\\
& T_k \upharpoonright (t_0 + t) \neq S_k \upharpoonright (t_0 + t)\\
& \wedge\\
& j \notin \mathrm{seg}(k) \Rightarrow T_j \upharpoonright t_0 = S_j \upharpoonright t_0
\end{aligned}$$

Then C is the set of indices k such that the two vectors $\underline{S}$ and $\underline{T}$ first become different at component k between times t_0 and $t_0 + t$, agree on all components indexed from seg(k) up until time $t_0 + t$, and agree on all other components up until time t_0. We claim that

$$C = \{\} \;\Rightarrow\; \underline{S} = \underline{T}$$

To show this, we establish the contrapositive of the result, by assuming that $\underline{S} \neq \underline{T}$. Define a sequence of indices i_n from I and a sequence of times t_n such that

$$\begin{aligned}
i_0 &\in \{i : I \mid S_i \neq T_i \wedge j \in \mathrm{seg}(i) \Rightarrow S_j = T_j\}\\
t_0 &= t + sup\{t' \mid S_{i_0} \upharpoonright t' = T_{i_0} \upharpoonright t'\}\\
i_{n+1} &\in \{i : I \mid S_i \upharpoonright t_n \neq T_i \upharpoonright t_n \wedge j \in \mathrm{seg}(i) \Rightarrow S_j \upharpoonright t_n = T_j \upharpoonright t_n\}\\
t_{n+1} &= t_n - t
\end{aligned}$$

Observe that i_0 exists, and that t_0 is therefore finite. We are assuming that $\underline{S}$ differs at some point from $\underline{T}$, and hence (as the index set I is well-ordered) there will be a least index i_0 where the two vectors differ. Either i_n exists for all $n : \mathbb{N}$, in which case

$$\forall\, n : \mathbb{N} \bullet t_0 - nt > 0$$

which contradicts the fact that t_0 is finite, or there is a n such that i_n exists but i_{n+1} does not. In this case, $i_n \in C$ and so C is not empty, as required. This establishes our claim.

We have now to prove that the set C is empty. To do this, we assume for a contradiction that $k \in C$, then we know that

$$\begin{aligned} \exists\, t_0 \quad \bullet \quad & T_k \upharpoonright t_0 + t \neq S_k \upharpoonright t_0 + t \\ & \wedge j \in \text{seg}(k) \Rightarrow T_j \upharpoonright t_0 + t = S_j \upharpoonright t_0 + t \\ & \wedge j \notin \text{seg}(k) \Rightarrow T_j \upharpoonright t_0 = S_j \upharpoonright t_0 \end{aligned}$$

The vector $\underline{P}$ is t-constructive for $\underline{X}$, hence

$$j \notin \text{seg}(k) \Rightarrow P_k \text{ is } t\text{-constructive for } X_j$$

Applying lemma 3.3 we have that, for $j \notin \text{seg}(k)$,

$$S_j \upharpoonright t_0 = T_j \upharpoonright t_0 \quad \Rightarrow \quad \mathcal{F}_T [\![P_k]\!] \rho[S_j/X_j] \upharpoonright t_0 + t = \mathcal{F}_T [\![P_k]\!] \rho[T_j/X_j] \upharpoonright t_0 + t$$

and recalling that P_k must be 0-constructive for all X_j, we obtain

$$\forall j : I \quad \bullet \quad \mathcal{F}_T [\![P_k]\!] \rho[S_j/X_j] \upharpoonright t_0 + t = \mathcal{F}_T [\![P_k]\!] \rho[T_j/X_j] \upharpoonright t_0 + t$$

We may now apply Theorem 3.30. This gives us that

$$\mathcal{F}_T [\![P_k]\!] \rho[\underline{S}/\underline{X}] \upharpoonright t_0 + t \;\; = \;\; \mathcal{F}_T [\![P_k]\!] \rho[\underline{T}/\underline{X}] \upharpoonright t_0 + t$$

We have also that $\underline{S}$ is a fixed point of $M(\underline{X}, \underline{P})\rho$. In this case

$$\begin{aligned} & M(\underline{X}, \underline{P})\rho\, \underline{S} &=& \;\underline{S} \\ \Rightarrow \quad & \lambda\, \underline{Y} \bullet \mathcal{F}_T [\![\underline{P}]\!] \rho[\underline{Y}/\underline{X}]\, \underline{S} &=& \;\underline{S} \\ \Rightarrow \quad & \mathcal{F}_T [\![\underline{P}]\!] \rho[\underline{S}/\underline{X}] &=& \;\underline{S} \\ \Rightarrow \quad & \mathcal{F}_T [\![P_k]\!] \rho[\underline{S}/\underline{X}] &=& \;S_k \end{aligned}$$

and a similar result holds for the other fixed point, $\underline{T}$. Hence

$$S_k \upharpoonright t_0 = T_k \upharpoonright t_0 \quad \Rightarrow \quad S_k \upharpoonright t_0 + t = T_k \upharpoonright t_0 + t$$

which contradicts our choice of $k \in C$. Hence the set C is empty. By our earlier claim, this means that the two vectors $\underline{S}$ and $\underline{T}$ are identical. Hence $\underline{S}$ is the only fixed point of $M(\underline{X}, \underline{P})\rho$.

To summarise: the secondary vector $\underline{Q}$ is well-defined, and corresponds to a contraction mapping in the semantic model; the unique fixed point of this mapping is a fixed point of $M(\underline{X}, \underline{P})\rho$, the mapping corresponding to $\underline{P}$ and $\underline{X}$; further, it is the *only* fixed point of this mapping. We may conclude that, although the mapping corresponding to $\underline{P}$ need not be contraction mapping, it has a unique fixed point in TS_F^I. □

From this result, we may deduce the welcome corollary:

Corollary 3.32 If vector of terms $\underline{P}$ is constructive for vector of variables $\underline{X}$, then the recursion $\mu\, \underline{X} \circ \underline{P}$ is well-defined. ♡

This justifies our definition of *constructive*, and lays the foundation for the theory of recursion induction presented in chapter 5.

3.4 Equation Sets

Consider the following mutual recursion:

$$\begin{aligned} P &= a \xrightarrow{I} Q \\ Q &= b \xrightarrow{I} P \end{aligned}$$

It should be obvious that

$$P = a \xrightarrow{I} b \xrightarrow{I} P$$

We can derive rules that allow us to make such transformations while preserving the semantics of the term defined by the equation set. For example, we may wish to replace all free occurrences of a recursive variable:

Rule 3.33 (Substitution) If the equation $X_k = P_k$ has a unique solution in TS_F, and appears in $\langle X_i = P_i \rangle$, then we may substitute $\mu\, X_k \circ P_k$ for all free occurrences of X_k in all equations of the equation set. Formally,

$$\langle X_i = P_i \rangle_j \equiv \langle X_i = P_i[\mu\, X_k \circ P_k / X_k] \rangle_j$$

△

Proof If $X_k = P_k$ has a unique solution in TM_F, and $\underline{S}$ is a fixed point of $M(\underline{X}, \underline{P})$, then

$$\mathcal{F}_T \llbracket P_k \rrbracket \rho[\underline{S}/\underline{X}][\mathcal{F}_T \llbracket \mu\, X_k \circ P_k \rrbracket \rho[\underline{S}/\underline{X}]/X_k] = S_k$$

To see this, observe that

$$\begin{aligned} S_k &= (M(\underline{X},\underline{P})\rho\,\underline{S})_k \\ &= \mathcal{F}_T[\![P_k]\!]\rho[\underline{S}/\underline{X}] \\ &= \mathcal{F}_T[\![P_k]\!]\rho[\underline{S}/\underline{X}][S_k/X_k] \end{aligned}$$

Hence S_k is the fixed point of the function $M(X_k, P_k)\rho[\underline{S}/\underline{X}]$, and is therefore equal to

$$\mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{S}/\underline{X}] \;=\; \mathcal{F}_T[\![P_k]\!]\rho[\underline{S}/\underline{X}][\mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{S}/\underline{X}]$$

Assuming that $\underline{S}$ is a fixed point of $M(\underline{X},\underline{P})\rho$, we define $\underline{Q}$ to be the vector obtained by substituting $\mu\, X_k \circ P_k$ for all free occurrences of X_k in the vector $\underline{P}$.

$$\begin{aligned} (M(\underline{X},\underline{Q})\rho\,\underline{S})_j &= \mathcal{F}_T[\![P_j[\mu\, X_k \circ P_k/X_k]]\!]\rho[\underline{S}/\underline{X}] \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{S}/\underline{X}][\mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{S}/\underline{X}]/X_k] \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{S}/\underline{X}][S_k/X_k] \\ &= S_j \end{aligned}$$

Conversely, if $\underline{T}$ is a fixed point of the function $M(\underline{X},\underline{Q})\rho$, then

$$\mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{T}/\underline{X}] \;=\; T_k$$

To see this, observe that

$$\begin{aligned} T_k &= (M(\underline{X},\underline{Q})\rho\,\underline{T})_k \\ &= \mathcal{F}_T[\![Q_k]\!]\rho[\underline{T}/\underline{X}] \\ &= \mathcal{F}_T[\![P_k[\mu\, X_k \circ P_k/X_k]]\!]\rho[\underline{T}/\underline{X}] \\ &= \mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{T}/\underline{X}] \end{aligned}$$

Assuming that $\underline{T}$ is a fixed point of $M(\underline{X},\underline{Q})\rho$, we have that

$$\begin{aligned} (M(\underline{X},\underline{P})\rho\,\underline{T})_j &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{T}/\underline{X}] \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{T}/\underline{X}][T_k/X_k] \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{T}/\underline{X}][\mathcal{F}_T[\![\mu\, X_k \circ P_k]\!]\rho[\underline{T}/\underline{X}]/X_k] \\ &= \mathcal{F}_T[\![P_j[\mu\, X_k \circ P_k/X_k]]\!]\rho[\underline{T}/\underline{X}] \\ &= \mathcal{F}_T[\![Q_j]\!]\rho[\underline{T}/\underline{X}] \\ &= T_j \end{aligned}$$

We have demonstrated that a vector U is a fixed point of $M(\underline{X},\underline{P})\rho$ if and only if it is a fixed point of $M(\underline{X},\underline{Q})\rho$. The soundness of the rewrite rule follows immediately. □

The presence of 0-constructive terms in an equation set may mean that there is more than one solution to the equations. In this case, the semantics of the recursion is not well-defined. However, if the offending terms do not affect the semantics of the selected component, we can rewrite the equation set to eliminate them. For example, in the equation set

$$\begin{aligned} P &= a \xrightarrow{t} P \\ Q &= P \sqcap Q \end{aligned}$$

recursive variable Q does not appear free in the term part of the first equation. We may remove the second equation from the equation set without affecting the P component of the solution. In this way, we may delete unnecessary or undesirable equations from our equation set. We capture this result as a proof rule:

Rule 3.34 (Elimination)

$$\frac{\forall j : J \bullet \forall i : (I - J) \bullet X_i \text{ is not free in } P_j}{\langle X_i = P_i \mid i \in I \rangle_k \equiv \langle X_i = P_i \mid i \in J \rangle_k} \quad [\, k \in J \wedge J \subseteq I \,]$$

△

In a set of equations indexed by I, we may eliminate those equations

$$X_i = P_i$$

for which X_i does not occur free in any of the terms $P_j \mid j \in J$, where J indexes the set of remaining equations. This is enough to ensure that the semantics of the remaining components is preserved: that

$$\langle X_i = P_i \mid i \in I \rangle_k = \langle X_i = P_i \mid i \in J \rangle_k$$

whenever $k \in J$.

Proof Suppose that $\underline{S}$ is a fixed point of the function $M(\underline{X}, \underline{P})\rho$, where $\underline{P}$ and $\underline{X}$ are indexed by I. Let $\underline{X}'$, $\underline{P}'$ and $\underline{S}'$ be the corresponding vectors indexed by set J.

$$\begin{aligned} S'_j &= S_j \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{S}/\underline{X}] \\ &= \mathcal{F}_T[\![P_j]\!]\rho[\underline{S}'/\underline{X}'] \\ &= (M(\underline{X}', \underline{P}')\rho\, S')_j \end{aligned}$$

Hence, any solution to the equation set $\langle X_i = P_i \rangle$ gives rise to a solution of the set $\langle X'_i = P'_i \rangle$. This is enough to establish the soundness of the rule. □

Returning to the example at the beginning of this section, we may now establish that the two definitions of P given below are equivalent:

$$\begin{aligned} P &\mathrel{\hat=} a \xrightarrow{1} Q \\ Q &\mathrel{\hat=} b \xrightarrow{1} P \end{aligned} \qquad\qquad P \mathrel{\hat=} a \xrightarrow{1} b \xrightarrow{1} P$$

We begin by writing the left-hand definition in vector form:

$$P \mathrel{\hat=} \left\langle \begin{array}{lcl} X_1 &=& a \xrightarrow{1} X_2 \\ X_2 &=& b \xrightarrow{1} X_1 \end{array} \right\rangle_1$$

From lemma 3.5, we know that $b \xrightarrow{1} X_1$ is constructive for X_2, hence the second equation has a unique solution in TS_F. Applying the rule 3.33, we obtain

$$P = \left\langle \begin{array}{lcl} X_1 &=& a \xrightarrow{1} (\mu X_2 \circ b \xrightarrow{1} X_1) \\ X_2 &=& b \xrightarrow{1} X_1 \end{array} \right\rangle_1$$

and X_2 is not free in any of the right-hand terms, rule 3.34 gives us that

$$\begin{aligned} P &= \left\langle\, X_1 = a \xrightarrow{1} (\mu X_2 \circ b \xrightarrow{1} X_1) \,\right\rangle_1 \\ &= \left\langle\, X_1 = a \xrightarrow{1} b \xrightarrow{1} X_1 \,\right\rangle_1 \\ &= (\mu\langle X_1\rangle \circ \langle a \xrightarrow{1} b \xrightarrow{1} X_1\rangle)_1 \\ &= \mu X_1 \circ a \xrightarrow{1} b \xrightarrow{1} X_1 \end{aligned}$$

We may then apply corollary 3.18, yielding

$$P = a \xrightarrow{1} b \xrightarrow{1} P$$

as required. From this example, it is clear that the following derived rule will be useful:

Rule 3.35 Given the equation set $\langle X_i = P_i\rangle_j$, where X_k does not occur free in P_k, we may substitute P_k for all free occurrences of X_k in the remaining equations of the equation set, and remove $X_k = P_k$ from the equation set. Formally,

$$\langle X_i = P_i\rangle_j \;=\; \langle X_i = P_i[P_k/X_k \mid i \neq k]\rangle_j$$

providing that $j \neq k$. △

Proof This rule follows easily from rules 3.33 and 3.34. □

3.5 Examples

Consider the timed sensitive vending machine of section 2.8:

$$\begin{aligned} TSVM &\mathrel{\hat=} coin \xrightarrow{t_1} (reset \xrightarrow{t_3} TSVM \\ &\qquad\qquad\qquad \overset{t_2}{\rhd} \\ &\qquad\qquad\qquad PAID) \end{aligned}$$

$$PAID \mathrel{\hat=} coke \xrightarrow{t_4} TVSM$$

This process is defined by a set of mutually recursive equations, in which the vector of terms is uniformly constructive for the vector of variables. To see this, observe that the terms

$$\begin{aligned} &coin \xrightarrow{t_1} (reset \xrightarrow{t_3} TSVM \\ &\qquad \overset{t_2}{\rhd} \\ &\qquad PAID) \end{aligned} \qquad \text{and} \qquad coke \xrightarrow{t_4} TVSM$$

are both $min\{t_1, t_4\}$-constructive for any variable, by lemma 3.5. We may conclude that this mutual recursion has a well-defined semantics.

We may apply rule 3.35 to eliminate the second equation

$$\begin{aligned} TSVM &\mathrel{\hat=} coin \xrightarrow{t_1} (reset \xrightarrow{t_3} TSVM \\ &\qquad\qquad\qquad \overset{t_2}{\rhd} \\ &\qquad\qquad\qquad coke \xrightarrow{t_4} TSVM) \end{aligned}$$

and rewrite the process defiuition as a single recursion.

As an example of a mutual recursion in which the term vector is constructive but not uniformly constructive, consider the process *POINTER* defined by

$$\begin{aligned} POINTER &\mathrel{\hat=} POINTER_0 \\ POINTER_0 &\mathrel{\hat=} incr \xrightarrow{1} POINTER_1 \\ POINTER_{n+1} &\mathrel{\hat=} incr \xrightarrow{1} POINTER_{n+2} \\ &\qquad \sqcap \\ &\qquad POINTER_n \end{aligned}$$

where index n is drawn from the set of natural numbers $\mathbb{N}$. The form of the equations prevents the application of rules 3.33 and 3.34; an infiuite number of substitutions would be required.

However, the term corresponding to $POINTER_{n+1}$ is *1*-constructive for any instance of $POINTER_{n+m}$, whenever $m \geqslant 1$. With the usual ordering on the natural numbers, we may establish that the vector of terms is constructive for the vector of variables. If X_n denotes the n^{th} element of the variable vector, and P_n denotes the n^{th} element of the term vector, then

$$\forall i, j : \mathbb{N} \bullet j \geqslant i \Rightarrow P_i \text{ is constructive for } X_j$$

This is precisely the condition for $\underline{P}$ to be constructive for $\underline{X}$. We conclude that this mutual recursion has a well-defined semantics.

Chapter 4

Specification

We consider a *specification* of a system to be a formal description of its intended behaviour. In this chapter, we show how the language of timed failures may be used to produce specifications of real-time systems.

4.1 Behavioural Specifications

In Timed CSP, the semantics of a process is the set of all possible behaviours of that process; we may write specifications as predicates upon these semantic sets. In the Timed Failures model, each behaviour is recorded as a timed failure: a trace of events performed, and a set of events refused. In [Reed 88], Reed defines a specification A as a mapping from the model TM_F to $\{true, false\}$: the space of truth values. This specification holds of process P if and only if

$$A(\mathcal{F}_T[\![P]\!]\rho) \quad = \quad true$$

We choose instead to define predicates upon a typical element of the semantic set of a process: these are *behavioural specifications.*

In the Timed Failures model, a behavioural specification is a predicate $S(s, \aleph)$, with free variables s and $\aleph$ representing the two components of a possible behaviour. We say that a term P *satisfies* a behavioural specification $S(s, \aleph)$ in environment ρ if predicate S holds of every behaviour of P. Formally,

Definition 4.1

$$P \, \mathbf{sat}_\rho \, S(s, \aleph) \quad \widehat{=} \quad \forall s, \aleph \bullet (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \Rightarrow S(s, \aleph)$$

$\Diamond$

If P is a process, then we may omit the environment parameter to obtain the familiar **sat** notation, employed in [Hoare 85].

Definition 4.2

$$P \;\mathbf{sat}\; S(s,\aleph) \;\widehat{=}\; \forall \rho, s, \aleph \bullet (s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho \Rightarrow S(s,\aleph)$$

$\diamond$

Reed's approach to specification is more powerful: for every statement of the form $P\,\mathbf{sat}_\rho\, S(s,\aleph)$ there is a mapping A_S from TM_F to $\{\mathit{true}, \mathit{false}\}$ given by

$$A_S(Y) \;\widehat{=}\; \forall s, \aleph \bullet (s,\aleph) \in Y \bullet S(s,\aleph)$$

such that the following equivalence holds

$$P\,\mathrm{sat}_\rho\, S(s,\aleph) \;\equiv\; A_S(\mathcal{F}_T[\![P]\!]\rho)$$

and some of Reed's statements cannot be expressed with the sat notation. For example, consider the predicate A given by:

$$A(Y) \;\widehat{=}\; \exists s, \aleph \bullet (s,\aleph) \in Y \wedge a \in \sigma(s)$$

This requires that event a is a possible observation of any process represented by set Y in TM_F. There is no behavioural specification S such that

$$P\,\mathrm{sat}_\rho\, S(s,\aleph) \;\equiv\; A(\mathcal{F}_T[\![P]\!]\rho)$$

Reed's approach permits a more detailed analysis of the process semantics; ours is more suitable for the capture of process requirements. A behavioural specification is satisfied only when every behaviour of a term is acceptable. A statement of the form P sat $S(s,\aleph)$ is a *guarantee* of satisfactory behaviour.

Example

As an example of a behavioural specification, consider the following requirement upon a cash dispenser $CASH$: that it should not allow a user to make more than one withdrawal in any twenty-four hour period. We choose the event $u.cash$ to represent a withdrawal by user u, where u is drawn from a set of all possible users, $USER$. We capture this requirement as follows:

$$\begin{aligned} CASH \;\;\mathbf{sat}\;\; & \forall u : USER\,;\, I : TINT \bullet \\ & \qquad length(I) > 24 \Rightarrow \#(s \uparrow I \downharpoonright u.cash) \leqslant 1 \end{aligned}$$

Given any user u, if we consider the events observed during any interval of time longer than twenty-four hours, then there should be no more than one occurrence of the event $u.cash$; the length of the trace s during interval I restricted to this event should be no greater than 1.

The number of occurrences of a given event is a useful qnantity. To simplify future specifications, we define a counting operator

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

to yield the number of occurrences of events from set A in trace s. As usual, if A is a singleton set, we will omit the set braces.

Satisfiable Specifications

We say that a behavioural specification is *satisfiable* if there exists a Timed CSP process that satisfies it.

Definition 4.3 If $S(s, \aleph)$ is a behavioural specification, then S is satisfiable in the model TM_F if and only if

$$\exists\, Y : TS_F \;\bullet\; Y \in TM_F \;\wedge\; \forall\, s, \aleph \bullet (s, \aleph) \in Y \Rightarrow S(s, \aleph)$$

$\diamondsuit$

As we shall see in chapter 9, the existing syntax of *TCSP* is not enough to implement all of the processes in TM_F. Our definition of *satisfiable* allows for further additions to the syntax, or a strengthening of the axiom set. We may demonstrate that a specification is satisfiable by exhibiting a suitable piece of syntax:

Lemma 4.4 If $S(s, \aleph)$ is a behavioural specification such that

$$\exists\, P : TCSP \;\bullet\; P \,\mathsf{sat}\, S(s, \aleph)$$

then $S(s, \aleph)$ is satisfiable. $\heartsuit$

This result will be useful in chapter 5, when we consider the theory of recursion induction. To show that a recursive process meets a satisfiable behavioural specification S, we have only to show that S is preserved by each recursive call.

In applying Timed CSP to the specification and development of a real-time system, we would prefer to identify specifications that are not satisfiable before suggesting an implementation. The axioms of the semantic model give rise to necessary conditions for a specification to be satisfiable. For example,

Lemma 4.5 If $S(s, \aleph)$ is satisfiable, then $S(\langle\rangle, \{\})$. $\heartsuit$

Proof From definition 4.3:

$$\exists\, Y : TS_F \;\bullet\; Y \in TM_F \wedge \forall (s, \aleph) \bullet (s, \aleph) \in Y \Rightarrow S(s, \aleph)$$

From the first axiom of the semantic model given in 2.2, we have that:

$$\exists\, Y : TS_F \;\bullet\; (\langle\rangle, \{\}) \in Y \wedge \forall (s, \aleph) \bullet (s, \aleph) \in Y \Rightarrow S(s, \aleph)$$

The result follows immediately. $\square$

Any satisfiable behavioural specification must be true of the empty behaviour, which is a possible behaviour of every process. As a result, such specifications may not insist that a certain timed event appears in a trace or refusal, without a qualifying assumption. A more surprising result is:

Lemma 4.6 If $S(s, \aleph)$ is a behavioural specification such that

$$\exists\, e : T\Sigma \;\bullet\; S(s, \aleph) \Rightarrow e \notin \aleph$$

then $S(s, \aleph)$ is not satisfiable. ♡

Proof Suppose for a contradiction that $S(s, \aleph)$ is satisfiable, and that there exists a timed event e for which

$$S(s, \aleph) \;\Rightarrow\; e \notin \aleph$$

Let Y be a process satisfying $S(s, \aleph)$, and choose t and a such that $e = (t, a)$.

$$\begin{aligned}(s, \aleph) \in Y \;&\Rightarrow\; S(s, \aleph)\\ &\Rightarrow\; (t, a) \notin \aleph\end{aligned}$$

From the fourth axiom of the semantic model, given in section 2.2, we obtain

$$\begin{aligned}(s, \aleph) \in Y \;\Rightarrow\; &\exists\, \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s, \aleph') \in Y \wedge\\ &((t, a) \notin \aleph' \Rightarrow (s \upharpoonright t^\frown\langle(t, a)\rangle, \aleph' \upharpoonright t) \in Y)\end{aligned}$$

If an event a is excluded from all refusal sets at time t, then it must be possible for a to occur at that time. We know that (t, a) is excluded from all refusal sets of process Y, hence we have that

$$\begin{aligned}(s, \aleph) \in Y \;\Rightarrow\; &\exists\, \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s, \aleph') \in Y \wedge\\ &(s \upharpoonright t^\frown\langle(t, a)\rangle, \aleph' \upharpoonright t) \in Y\end{aligned}$$

From the first axiom of TM_F we know that the empty behaviour $(\langle\rangle, \{\})$ is present in Y. With this choice for $(s, \aleph)$, the above implication yields that

$$\exists\, \aleph : RSET \;\bullet\; (\langle\rangle, \aleph) \in Y \wedge (\langle(t, a)\rangle, \aleph \upharpoonright t) \in Y$$

A simple induction will establish that, for any natural number n, the trace in which n copies of event a occur at time t is a possible trace of Y.

$$\exists\, \aleph' : RSET \;\bullet\; (\langle(t, a)\rangle^n, \aleph' \upharpoonright t) \in Y$$

However, the bounded speed axiom of the semantic model places a natural number bound $n(t)$ on the number of events that may appear in any trace of Y before time t.

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

This forces a contradiction, and establishes the required result. □

It is always possible that a process will perform each observable event as soon as it becomes available. If process P makes n copies of event a available at time t, and it is offered $n + 1$ copies of a, then (t, a) will appear in the refusal set. Hence a satisfiable behavioural specification may not insist that a timed event is absent from the refusal set.

The first lemma shows that we cannot insist that an observable event occurs without making a qualifying assumption. This will be an assumption about the environment of the process; we will discuss such assumptions in section 4.3. Together, these lemmata dictate the form of safety and liveness specifications in the timed failures model.

4.2 Safety and Liveness

We will follow Lamport's informal classification of safety and liveness properties, [Lamport 77]: a *safety property* is a requirement that 'nothing bad happens', while a *liveness property* insists that 'some good thing will occur'. In either case, we must exclude undesirable behaviours from the semantic set of the process in question. In our computational model, a safety property corresponds to the requirement that a given event may not occur except under certain conditions: e.g.

* event a does not occur within time t of event b;
* if event a occurs, it must do so within time t of event b;
* event a may occur only at specified times.

Some safety specifications require timed refusal information—we may insist that a given event is not performed unless another has been offered—but most can be captured as predicates on traces.

The lemmata of the previous section lead to the following restriction upon liveness specifications in Timed CSP: we may insist only that certain timed events occur *or* are made available. For example, the following constraints may be expressed as satisfiable liveness specifications:

* event a is possible at time I;
* if the last event observed is b at time I, then event a is available at all times after $I + t$;
* if a has not occurred, then it is available.

Liveness properties are expressed as predicates on failures.

Safety Specifications

Any safety specification on process Y may be written in the form

$$\forall s, \aleph \bullet (s, \aleph) \in Y \Rightarrow s \notin U$$

where U is some set of undesirable traces. If this specification is to be satisfiable, then the empty trace must be an acceptable behaviour. As a result, the deadlock process $STOP$ will satisfy any satisfiable safety specification.

We can use the operators defined in section 2.1 to construct simple safety specifications. For example, we may wish to specify that the events *tick* and *tock* occur alternately in any trace of $CLOCK$:

$$CLOCK \quad \text{sat} \quad (s \downarrow tick = s \downarrow tock) \vee (s \downarrow tick = s \downarrow tock + 1)$$

Recalling that $s \downarrow a$ denotes the uumher of occurrences of event a in trace s, we see that the process $CLOCK$ must perform a *tick* before every *tock*.

An *event precondition* for event a is a predicate that describes the process state necessary for a to occur. In Timed CSP, any state information must be deduced from observable behaviours; we write event preconditions as predicates upon timed failures. As an example, consider the behavioural specification S defined by

$$S(s, \aleph) \;\widehat{=}\; \langle (t, a) \rangle \text{ in } s \Rightarrow b \in \sigma(s \upharpoonright t - 1) \wedge a \notin \sigma(\aleph \upharpoonright t)$$

This places the followiug precondition upon event a: if this event is observed at time t, then event b must be seen more than one time unit before t, and event a must be available up until time t. Event preconditions correspond closely to the notion of *firing conditions* in sequential state-based languages such as the Z notation [Woodcock 90].

Any event precondition upon event a can he written as a constraint upon the behaviour of the process up until the time at which a is observed:

$$\langle (t, a) \rangle \text{ in } s \quad \Rightarrow \quad C((s, \aleph) \uparrow [0, t))$$

The prefix closure property of process behaviours allows us to simplify such specifications. From the second axiom of the semantic model, we know that if a appears in a trace s of process P, then there is another behaviour of P in which a is the last event observed.

Lemma 4.7 If P represents a process, then

$$P \quad \text{sat} \quad \langle (t, a) \rangle \text{ in } s \Rightarrow C((s, \aleph) \uparrow [0, t))$$

if and only if

$$P \quad \text{sat} \quad foot(s) = (t, a) \Rightarrow C((s, \aleph) \uparrow [0, t))$$

♡

Proof The proof of *only if* is trivial. Conversely, assume that

$$P \ \ \mathbf{sat} \ \ foot(s) = (t, a) \Rightarrow C((s, \aleph) \uparrow [0, t))$$

and that

$$(s, \aleph) \in \mathcal{F}_T[\![P]\!] \ \wedge \ \langle (t, a) \rangle \ \mathbf{in} \ s$$

Choose trace w such that

$$s \uparrow [0,t) = w \uparrow [0, t) \ \wedge \ s \uparrow [t, \infty) \cong w \uparrow [t, \infty) \ \wedge \ foot(w \upharpoonright t) = (t, a)$$

The third axiom of the semantic model (section 2.2) states that every process is closed under trace equivalence, so $(w, \aleph)$ is also present in the semantic set of P. From the second axiom of the semantic model we obtain:

$$((w \upharpoonright t)^\frown(w \upharpoonleft t), \aleph) \in \mathcal{F}_T[\![P]\!] \ \ \Rightarrow \ \ (w \upharpoonright t, \aleph \upharpoonright begin(w \upharpoonleft t)) \in \mathcal{F}_T[\![P]\!]$$

Applying the first of our assumptions to the failure $(w \upharpoonright t, \aleph \upharpoonright begin(w \upharpoonleft t))$, we may conclude that

$$C((w \upharpoonright t, \aleph \upharpoonright (begin(w \upharpoonleft t))) \uparrow [0, t))$$

By our choice of w, and the properties of before, after and during:

$$\begin{aligned}(w \upharpoonright t, \aleph \upharpoonright (begin(w \upharpoonleft t))) \uparrow [0, t) &= (w, \aleph) \uparrow [0, t) \\ &= (s, \aleph) \uparrow [0, t)\end{aligned}$$

hence we have established that

$$C((s, \aleph) \uparrow [0, t))$$

We may conclude that the two specifications are equivalent. □

A behavioural specification must be satisfied by *all* behaviours of a process, so it is sufficient to consider the case in which a is the last event observed. The exclusion of trace information at time t is important; our intuition concerning cause and effect excludes information about events at time t from a precondition for the timed event (t, a).

Liveness Specifications

In our model of computation, a process and its environment cooperate on all observable actions. The visible events of a Timed CSP process represent an interface with the environment. Without some knowledge of the environment, we may not insist that a process performs an event at a particular time. We express liveness conditions as requirements on the availability of events, ensuring that the process will perform an event if the environment should agree.

In section 4.1, we saw that we cannot require that an event a is available at a particular time t without considering trace information. If an event occurs as soon as it becomes available, its availability may not be recorded. As a result, liveness conditions may take the form

$$a \notin \sigma(\aleph \uparrow I) \ \vee \ a \in \sigma(s \uparrow J)$$

The event a is made available throughout some interval I unless it occurs during some interval J.

Lemma 4.8 If the behavioural specification $S(s, \aleph)$ defined by

$$S(s,\aleph) \ \hat{=} \ a \notin \sigma(\aleph \uparrow I) \ \vee \ a \in \sigma(s \uparrow J)$$

is satisfiable, then $I \subseteq J$. ♡

Proof Suppose for a contradiction that S is satisfied by a process Y, and that there exists a time $t \in I - J$. We observe that

$$S(s,\aleph) \ \Rightarrow \ a \notin \sigma(\aleph \uparrow t) \ \vee \ a \in \sigma(s \uparrow J)$$

From our assumption that Y satisfies $S(s,\aleph)$, we may conclude that

$$(s,\aleph) \in Y \wedge a \notin \sigma(s \uparrow J) \ \Rightarrow \ a \notin \sigma(\aleph \uparrow t)$$

The fourth axiom of the semantic model states that

$$\begin{aligned}(s,\aleph) \in Y \ \Rightarrow \ & \exists \aleph' : RSET \bullet \aleph \subseteq \aleph' \wedge (s,\aleph') \in Y \wedge \\ & ((t,a) \notin \aleph' \Rightarrow (s \upharpoonright t^\frown\langle(t,a)\rangle, \aleph' \upharpoonright t) \in Y)\end{aligned}$$

Combining these properties, we obtain

$$(s,\aleph) \in Y \wedge a \notin \sigma(s \uparrow J) \ \Rightarrow \ \exists \aleph' : RSET \bullet (s \upharpoonright t^\frown\langle(t,a)\rangle, \aleph' \upharpoonright t) \in Y$$

From the first axiom of TM_F we know that the empty behaviour $(\langle\rangle, \{\})$ is present in Y. With this choice for $(s,\aleph)$, the above implication yields that

$$\exists \aleph : RSET \ \bullet \ (\langle\rangle, \aleph) \in Y \wedge (\langle(t,a)\rangle, \aleph \upharpoonright t) \in Y$$

As in the proof of lemma 4.6, a simple induction will establish that for any natural number n:

$$\exists \aleph' : RSET \bullet (\langle(t,a)\rangle^n, \aleph' \restriction t) \in Y$$

The bounded speed axiom of the semantic model places a natural number bound $n(t)$ on the number of events that may appear in any trace of Y before time t.

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

Again, this forces a contradiction, and establishes the required result. □

As an example of a liveness specification, consider the case of an electronic lock *LOCK*. If a key is inserted, then the lock must permit the door to be opened after five seconds. If *open* represents the act of opening the door, and *key* represents a key insertion, then this requirement may be written as follows:

$$\begin{aligned} LOCK \quad \text{sat} \quad \langle(t, key)\rangle \text{ in } s \Rightarrow & \; open \notin \sigma(\aleph \upharpoonright t+5) \\ & \vee \\ & \; open \in \sigma(s \upharpoonright t) \end{aligned}$$

For simplicity, we have assumed that the door is opened only once.

An *event postcondition* for a is a predicate that places a constraint upon the possible behaviours of a process following the observation of a. Any event postcondition may be written in the form:

$$\langle(t,a)\rangle \text{ in } s \Rightarrow C((s,\aleph) \uparrow (t,\infty))$$

When placing an event precondition on an event a, it is sufficient to consider the case in which a is the last event observed. This result does not hold for event postconditions, even if we restrict our attention to the last occurrence of a:

$$foot(s \downharpoonright a) = (t,a) \Rightarrow C((s,\aleph) \uparrow (t,\infty))$$

Although a useful form of specification, this is not equivalent to the event postcondition given above; the requirement that

$$a \rightarrow a \rightarrow STOP \quad \text{sat} \quad foot(s \downharpoonright a) = (t,a) \Rightarrow a \notin \sigma(s \upharpoonright t)$$

is easy to establish, while the following requirement is impossible:

$$a \rightarrow a \rightarrow STOP \quad \text{sat} \quad \langle(t,a)\rangle \text{ in } s \Rightarrow a \notin \sigma(s \upharpoonright t)$$

4.3 Environmental Conditions

We may use the notation of timed failures to analyse the behaviour of a process under a certain set of environmental conditions. One of the assumptions of our computational model is *maximum liveness*: if a process and its environment are both prepared to engage in a particular timed event, then that event will occur. This postulate allows us to include assumptions about the offers made by the environment as preconditions in a behavioural specification.

These preconditions may be used to reason about *non-robust* interfaces, where correct behaviour is dependent upon the cooperation of the environment. When specifying the behaviour of a system component, we may assume that certain patterns of external communication will never be encountered.

Assumptions

To include an assumption about the environment in a specification of a process, we write the specification in the form

$$P \ \textbf{sat} \ \ E(s,\aleph) \Rightarrow F(s,\aleph)$$

where E is a predicate that corresponds to our assumption, and F characterises our requirements. This implication is vacuously true for any behaviour that does not meet the environmental condition; in this case, no requirement is placed upon the process. However, we must ensure that predicate F is true of any behaviour of P that meets condition E.

It is instructive to consider the extreme assumptions *true* and *false*. In the first case, we are placing no constraint upon the environment; the following equivalence will hold:

$$P \ \textbf{sat} \ (\mathit{true} \Rightarrow F(s,\aleph)) \ \equiv \ P \ \textbf{sat} \ F(s,\aleph)$$

To show that a process P meets requirement F in any environment, we must show that F holds of all the behaviours of P. If our environmental assumption is *false* then we are assuming a miraculous environment, in which any process meets every requirement:

$$P \ \textbf{sat} \ (\mathit{false} \Rightarrow F(s,\aleph)) \ \equiv \ \mathit{true}$$

In practice, our environmental assumptions will be more reasonable.

Trace Conditions

A timed trace is a record of observable events performed by the process; each of these events requires the cooperation of the environment. If the environment never offers a timed event (t, a), then this event will never be observed. To examine the resulting behaviour of the process, we restrict our attention to those failures which exclude this event from the trace. This may be extended to disqualify whole sequences of possible events.

For example, we may wish to specify that a personal computer PC will behave according to specification $SPEC(s, \aleph)$, providing that it is switched on before a disk is inserted. If we use *on* to represent the activation of the machine, and *insert* to represent a disk insertion, then we may capture this requirement as follows:

$$PC \quad \text{sat} \quad (begin(s \downharpoonright on) < begin(s \downharpoonright insert)) \Rightarrow SPEC(s, \aleph)$$

We are assuming that the machine is activated only once. The addition of an *off* event to our description would permit a more realistic treatment.

An assumption about possible traces corresponds to a safety specification upon the environment of a process. If we reqnire that

$$P \quad \text{sat} \quad s \notin U \Rightarrow SPEC(s, \aleph)$$

where U denotes a set of disqnalified traces, then we are assuming that the environment will not offer these sequences of timed events. If the environment of P is another process Q such that

$$Q \quad \text{sat} \quad s \notin U$$

then the behaviours of P should meet specification $SPEC$.

Failure Conditions

A timed refusal is a partial record of offers made by the environment of a process. If an event e is present in the refusal set, then we may infer that the environment offers more copies of e than the process is able to perform. By considering only those failures which include e in the refusal set, we may examine the result of placing a process in an environment which is willing to perform as many copies of e as necessary.

For example, we may require that a process P meets a specification F, providing that the environment is willing to accept at least one output every five time units:

$$P \quad \text{sat} \quad E(s, \aleph) \Rightarrow F(s, \aleph)$$

where E is an environmental condition defined by

$$\begin{aligned} E(s,\aleph) \;\; &\widehat{=} \;\; \forall I : TINT \bullet \\ &\qquad length(I) \geqslant 5 \Rightarrow output \in \sigma(s \uparrow I) \vee output \in \sigma(\aleph \uparrow I) \end{aligned}$$

If I is an interval of time longer than five time units, then there must be some time during that interval at which the environment offers to participate in *output*. This corresponds to the inclusion of the event $(t, output)$ in the trace or refusal, depending on whether or not the offer was accepted.

A failure condition corresponds to a liveness specification upon the environment of a process. For example, if we wish process Q to model that part of the environment that accepts output from P, we should ensure that

$$\begin{aligned} Q \;\; \textbf{sat} \;\; &\forall I : TINT \bullet \\ &\qquad length(I) \geqslant 5 \Rightarrow output \in \sigma(s \uparrow I) \vee (I \times \{output\}) \nsubseteq \aleph \end{aligned}$$

Our choice of Q means that the event *output* is concealed from the rest of the environment. Assuming that P and Q have no other events in common, we may combine them as follows:

$$(P \underset{output}{\parallel} Q) \setminus output$$

As we shall see, the concealment of an event corresponds to the assumption that all external offers are refused. The parallel combination of P and Q can refuse *output* when either process refuses. For any interval I longer than five time units, either P refuses *output* at some time during I or Q refuses output throughout I, in which case *output* must occur. In either case, the E condition is satisfied and specification F must hold.

The above example illustrates the dual relationship between liveness conditions and readiness assumptions:

$$\begin{array}{lcl} a \in \sigma(s \uparrow I) & & a \in \sigma(s \uparrow I) \\ \vee & \sim & \vee \\ J \times \{a\} \nsubseteq \aleph & & a \in \sigma(\aleph \uparrow J) \end{array}$$

If a process Q satisfies the liveness condition (on the left), then it will serve as a suitable environment for any process requiring the readiness assumption.

4.4 Example

We consider a specification of the timed sensitive vending machine defined in section 2.8. This machine was intended to dispense a drink for every coin inserted; we use the events *coin* and *coke* to represent the insertion of a coin and the removal of a drink, respectively. The company that operates the machine requires that every drink is paid for in advance, so we must place the following safety specification upon *TSVM*:

$$SAFE(s) \mathrel{\widehat{=}} s \downarrow coke \leqslant s \downarrow coin$$

The number of drinks dispensed is no greater than the number of coins accepted.

For profitability, the company requires that the machine is ready to accept another coin within time t_4 of a drink being dispensed. We place the following liveness specification upon the implementation:

$$\begin{aligned} NEXT(s, \aleph) \mathrel{\widehat{=}} \langle (t, coke) \rangle \text{ in } s \Rightarrow\ & coin \notin \sigma(\aleph \upharpoonright t + t_4) \\ & \vee \\ & coin \in \sigma(s \upharpoonright t) \end{aligned}$$

If a drink is removed at time t, then the event *coin* must become available no later than time $t + t_4$. This offer is represented by the absence of the event from the refusal set, or the presence of the event in the trace.

If the machine is kicked within time t_1 of a coin being inserted, a *reset* event will occur, and the coin will be lost. Rather than design a more robust mechanism, the manufacturers construct the machine to the following specification:

$$OKAY(s, \aleph) \mathrel{\widehat{=}} E(s, \aleph) \Rightarrow F(s, \aleph)$$

where $F(s, \aleph)$ requires that a drink is made available time t_5 after a coin is inserted:

$$\begin{aligned} F(s, \aleph) \mathrel{\widehat{=}} \langle (t, coin) \rangle \text{ in } s \Rightarrow\ & coke \notin \sigma(\aleph \upharpoonright t + t_5) \\ & \vee \\ & coke \in \sigma(s \upharpoonright t) \end{aligned}$$

and $E(s, \aleph)$ is an environmental condition corresponding to the assumption that the machine is treated gently for at least t_1 after each coin is inserted:

$$E(s, \aleph) \mathrel{\widehat{=}} \langle (t, coin) \rangle \text{ in } s \Rightarrow reset \notin \sigma(s \uparrow [t, t + 1])$$

If this environmental condition is met, then the machine guarantees to offer a drink at the appropriate time.

Chapter 5

Proof

Chapter 2 presented a algorithmic language for the description of real-time systems, and chapter 4 showed that behavioural specifications may be used to describe the safety and liveness properties of a such a system. In this chapter, we address the problem of proving that a suggested *TCSP* implementation satisfies a given behavioural specification.

5.1 A Proof System

In [Brookes 83] Brookes presented a proof system for untimed CSP, based upon a set of semantics-preserving algebraic laws. With the addition of timing information, many of these laws must be repealed. For example,

$$\begin{array}{rcl} (a \to P) \,|||\, (b \to Q) & \not\equiv & a \to (P \,|||\, (b \to Q)) \\ & & \Box \\ & & b \to ((a \to P) \,|||\, Q) \end{array}$$

The left-hand process may engage in the two events a and b simultaneously; the right-hand side describes a process which is initially sequential: after performing the first event, a strictly positive time δ must elapse before it can perform another. We cannot change the degree of parallelism in a real-time system without considering the behaviours of the processes involved; there is no rule for the elimination of interleaving parallel operator.

Similar problems arise when we consider the properties of the hiding operator. When we conceal a set of events from the environment of a process, we do more than simply remove them from the trace: we determine the times at which they are scheduled to occur. Although many of the equivalences presented in [Brookes 83] are preserved, they serve only to illustrate desirable properties of our semantic model. They do not comprise a *complete* set of laws; there are other equivalences that we are unable to demonstrate without reconrse to the semantic equations. This precludes the algebraic method of proof pioneered by Milner in [Milner 80], in which similar laws are used to establish that a suggested implementation is equivalent to a process already known to have the reqnired properties.

This is no cause for alarm. We are able to produce a complete proof system for *proofs of satisfaction* in the model TM_F. If it is true that all the behavionrs of an implementation P meet a certain behavioural specification $S(s,\aleph)$, then it will be possible to show that

$$P \;\; \mathbf{sat} \;\; S(s,\aleph)$$

using the inference rules presented in this chapter. Each of these rules will take the following form:

$$\frac{\begin{array}{c} \textit{antecedent} \\ \vdots \\ \textit{antecedent} \end{array}}{\textit{consequent}} \;\; [\, \textit{side condition} \,]$$

If we establish the truth of each *antecedent*, then we can be assured of the truth of the *consequent*, providing that the *side condition* holds.

We will present an inference rule for each clause in the syntax of *TCSP*, expressing the behavioural properties of a process in terms of component specifications. For compound processes, the antecedent part of the rule will consist of behavioural specifications for the syntactic subcomponents. For atomic processes, the rules will be without antecedents. In either case, the consequent will be the strongest specification that may be inferred about the process.

We may use the definition of $\mathbf{sat}_p$ in the previous chapter to establish the following logical rules:

$$\frac{}{P\,\mathbf{sat}_p\; \mathit{true}} \qquad \frac{\begin{array}{l} P\,\mathbf{sat}_p\, S(s,\aleph) \\ P\,\mathbf{sat}_p\, T(s,\aleph) \end{array}}{P\,\mathbf{sat}_p\, S(s,\aleph) \wedge T(s,\aleph)} \qquad \frac{\begin{array}{l} P\,\mathbf{sat}_p\, S(s,\aleph) \\ S(s,\aleph) \Rightarrow T(s,\aleph) \end{array}}{P\,\mathbf{sat}_p\; T(s,\aleph)}$$

The null specification is true of any process, each goal may be addressed separately, and we may weaken any specification already established.

5.2 Sequential Processes

Atoms

The processes $\bot$ and *STOP* are both unwilling to participate in any external activity. The inference rules for these operators are:

$$\frac{}{\bot\ \mathbf{sat}_\rho\ s = \langle\rangle} \qquad\qquad \frac{}{STOP\ \mathbf{sat}_\rho\ s = \langle\rangle}$$

Any trace of either process must be equal to the empty trace, but we can infer nothing about a typical refusal set: $\aleph$ may be any element of *RSET*.

The process *SKIP* is initially prepared to perform the termination event $\checkmark$, the only action that this process may perform:

$$\frac{}{\begin{array}{ll} SKIP\ \mathbf{sat}_\rho & (s = \langle\rangle \wedge \checkmark \notin \sigma(\aleph)) \\ & \vee \\ & (s = \langle(t, \checkmark)\rangle \wedge \checkmark \notin \sigma(\aleph \upharpoonright t) \wedge t \geqslant 0) \end{array}}$$

Either no events have been observed and the event $\checkmark$ is available, or $\checkmark$ has been observed (at some time t) and was continuously available beforehand. A similar rule pertains to the delay process *WAIT t*, in which the termination event becomes available at time t:

$$\frac{}{\begin{array}{ll} WAIT\ t\ \mathbf{sat}_\rho & s = \langle\rangle \wedge \checkmark \notin \sigma(\aleph \upharpoonleft t) \\ & \vee \\ & s = \langle(t', \checkmark)\rangle \wedge \checkmark \notin \sigma(\aleph \uparrow [t, t')) \wedge t' \geqslant t \end{array}}$$

If no events have been observed then $\checkmark$ must be available continuously from time t onwards. Otherwise, $\checkmark$ is observed at a time $t' \geqslant t$ and made available at all times between t and t'.

We do not require a proof rule for term variables. Timed CSP processes will contain no free occurrences of any variable; whenever we come to establish a result about a term P with a free variable X, we will be within the scope of the assumption

$$X \quad \mathbf{sat}_\rho \quad S(s, \aleph)$$

for some behavioural specification S.

Prefix

The undecorated prefix operator is associated with a constant delay of δ. From the semantic equation given in chapter 2, we may derive the inference rule below:

$$\frac{P \ \mathbf{sat}_\rho \ S(s,\aleph)}{\begin{array}{ll} a \to P \ \mathbf{sat}_\rho & s = \langle\rangle \wedge a \notin \sigma(\aleph) \\ & \vee \\ & s = \langle(t,a)\rangle^\frown s' \wedge a \notin \sigma(\aleph \upharpoonright t) \ \wedge begin(s') \geqslant (t+\delta) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\, S((s',\aleph) - (t+\delta)) \end{array}}$$

Under the assumption that P meets behavioural specification $S(s,\aleph)$ in the current environment ρ, we may infer the following statements about a typical failure of the term $a \to P$:

* if s is empty, then the event a may not be refnsed, and is therefore absent from the refusal set $\aleph$
* if s is non-empty, then the first event must be a. If a occurs at time t, we know that a is not refused before this time.
* if a occurs at time t, then the subseqnent behaviour is that of P, following a delay of δ. This subsequent behavionr must satisfy the predicate S.

The inference rule for delayed prefix is a simple generalisation:

$$\frac{P \ \mathbf{sat}_\rho \ S(s,\aleph)}{\begin{array}{ll} a \xrightarrow{t} P \ \mathbf{sat}_\rho & s = \langle\rangle \wedge a \notin \sigma(\aleph) \\ & \vee \\ & s = \langle(t',a)\rangle^\frown s' \wedge a \notin \sigma(\aleph \upharpoonright t') \ \wedge begin(s') \geqslant (t'+t) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\, S((s',\aleph) - (t'+t)) \end{array}}$$

In this case, if event a is observed at time t', the subsequent behaviour is that of P starting at time $t'+t$.

Sequential Composition

The inference rule for the sequential composition $P\,;Q$ is complicated by the fact that both terms are able to perform actions at the time of transfer of control. If control has not been transferred, then any trace of the composite term is a trace of P during which $\checkmark$ is not observed, and would be refnsed if offered. Otberwise, we may infer only that the trace is a permutation of traces s_P and s_Q, performed by P and Q respectively:

$$\begin{array}{l}
P \;\mathbf{sat}_\rho\; S(s,\aleph) \\
Q \;\mathbf{sat}_\rho\; T(s,\aleph) \\
\hline
P\,;Q \;\mathbf{sat}_\rho\; \checkmark \notin \sigma(s) \wedge \forall I \in \mathit{TINT} \bullet S(s, \aleph \cup (I \times \{\checkmark\})) \\
\qquad\qquad \vee \\
\qquad\qquad \exists\, s_P, s_Q \bullet s \cong s_P{}^\frown s_Q \wedge \checkmark \notin \sigma(s_P) \wedge \mathit{begin}(s_Q) \geqslant t \\
\qquad\qquad\qquad\qquad \wedge\; S(s_P{}^\frown\langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\}) \\
\qquad\qquad\qquad\qquad \wedge\; T((s_Q,\aleph) - t)
\end{array}$$

The trace s_P may be extended with a $\checkmark$ event at some time t (this event is hidden by the sequential composition operator). In the presence of the sequential composition operator, the event $\checkmark$ occurs as soon as it becomes available, so we know that it is refused at any time before t. Hence the failure

$$(s_P{}^\frown\langle(t,\checkmark)\rangle, \aleph \upharpoonright t \cup [0,t) \times \{\checkmark\})$$

must be a behaviour of P, which meets specification S. The second part of the trace, together with the refusals after t, forms a behaviour of Q.

To simplify the process of reasoning about sequential composition, we exhibit derived inference rules for the cases in which either P or Q is a delay process. The expression $\mathit{WAIT}\ t\,;P$ denotes a term that behaves as P, after an initial delay of time t:

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

In the expression $P\,;\mathit{WAIT}\ t\,;Q$, a delay of time t is associated with the transfer of control from P to Q. This delay allows us to separate the behaviours of the component processes:

$$\begin{array}{l}
P \;\mathbf{sat}_\rho\; S(s,\aleph) \\
Q \;\mathbf{sat}_\rho\; T(s,\aleph) \\
\hline
P\,;\mathit{WAIT}\ t\,;Q \;\mathbf{sat}_\rho\; \checkmark \notin \sigma(s) \wedge \forall I \in \mathit{TINT} \bullet S(s, \aleph \cup I \times \{\checkmark\}) \\
\qquad\qquad\qquad\qquad \vee \\
\qquad\qquad\qquad\qquad S(s \upharpoonright t'{}^\frown\langle(t',\checkmark)\rangle, \aleph \upharpoonright t' \cup [0,t') \times \{\checkmark\}) \\
\qquad\qquad\qquad\qquad\qquad \wedge\; s \uparrow (t', t'+t) = \langle\rangle \\
\qquad\qquad\qquad\qquad\qquad \wedge\; T((s,\aleph) - (t+t'))
\end{array}$$

The delayed sequential composition operator $\mathbf{;}$ is a special case of this construct.

Nondeterministic Choice

Any behaviour of the nondeterministic choice $P \sqcap Q$ must arise from either P or Q. This gives rise to the obvious inference rule:

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s,\aleph) \\ Q \,\mathbf{sat}_\rho\, T(s,\aleph) \end{array}}{P \sqcap Q \,\mathbf{sat}_\rho\, S(s,\aleph) \vee T(s,\aleph)}$$

The indexed form of this operator is not well-defined unless the set of alternatives is uniformly bounded. This requirement appears as a side-conditiou in the inference rule below:

$$\frac{\forall i : I \bullet P_i \,\mathbf{sat}_\rho\, S(s,\aleph)}{\bigsqcap_{i \in I} P_i \,\mathbf{sat}_\rho\, S(s,\aleph)} \quad [\ \{P_i \mid i \in I\} \text{ is uniformly bounded }]$$

This condition is trivially true for a choice of delay processes. The proof rule for nondeterministic delay is simply

$$\frac{P \,\mathbf{sat}_\rho\, S(s,\aleph)}{WAIT\ T\ ; P \,\mathbf{sat}_\rho\, \exists\, t : T \bullet begin(s) \geqslant t \wedge S((s,\aleph) - t)}$$

We may infer that this process behaves as P, starting at some time t taken from the set T.

Deterministic Choice

As in the case of nondeterministic choice, we may infer that the combination $P \Box Q$ behaves as either P or Q. We may also infer that any event refused before the first observable event occurs must be refused by both processes:

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s,\aleph) \\ Q \,\mathbf{sat}_\rho\, T(s,\aleph) \end{array}}{\begin{array}{ll} P \Box Q \,\mathbf{sat} & S(s,\aleph) \vee T(s,\aleph) \\ & \wedge \\ & S(\langle\rangle, \aleph \upharpoonright begin(s)) \wedge T(\langle\rangle, \aleph \upharpoonright begin(s)) \end{array}}$$

Any behaviour of the form $(\langle\rangle, \aleph)$ must be common to both alternatives.

An indexed choice requires the side-condition that the set of alternatives is uniformly bounded:

$$\frac{\forall a : A \bullet P_a \ \mathbf{sat}_\rho\ S_a(s, \aleph)}{a : A \xrightarrow{t_a} P_a} \quad [\ \{P_a\} \text{ uniformly bounded }]$$

$$\begin{array}{l} \quad \mathbf{sat}_\rho \\ \qquad s = \langle\rangle \wedge A \cap \sigma(\aleph) = \{\} \\ \qquad \vee \\ \qquad a \in A \ \wedge s = \langle(t, a)\rangle^\frown s' \\ \qquad\qquad \wedge\ A \cap \sigma(\aleph \upharpoonright t) = \{\} \\ \qquad\qquad \wedge\ begin(s') \geqslant t + t_a \\ \qquad\qquad \wedge\ S_a((s', \aleph) - (t + t_a)) \end{array}$$

If no events have been observed, then all of the events in set A should be available. As a result, the intersection of A with the event set of the refusal $\aleph$ must be empty. Otherwise, if a is the first event observed, we know that $a \in A$, and the subsequent behaviour is that of P_a. A delay of t_a is associated with the transfer of control to the process P_a.

Relabelling

The inverse image of of P may engage in an event a whenever P may engage in the event $f(a)$.

$$\frac{P \ \mathbf{sat}_\rho\ S(s, \aleph)}{f^{-1}(P) \ \mathbf{sat}_\rho\ S(f(s), f(\aleph))}$$

The direct image of P may engage in an event $f(a)$ whenever P can engage in the event a:

$$\frac{P \ \mathbf{sat}_\rho\ S(s, \aleph)}{f(P) \ \mathbf{sat}_\rho\ \exists s' \bullet s = f(s') \wedge S(s', f^{-1}(\aleph))}$$

In the second inference rule above, the expression $f^{-1}(\aleph)$ denotes the set

$$\{(t, a) \mid (t, f(a)) \in \aleph\}$$

This is the inverse image of refusal set $\aleph$ under function f.

Abstraction

To reason about a term of the form $P \setminus A$, we identify the behaviours of P in which events from A occur as soon as possible. In section 2.4, we saw that these are failures of P in which events from A may be continuously refused:

$$\frac{P \,\mathsf{sat}_\rho\, S(s, \aleph)}{P \setminus A \,\mathsf{sat}_\rho\, \exists\, s' \bullet s = s' \setminus A \wedge S(s', \aleph \cup ([0, end(s', \aleph)) \times A))}$$

If $(s, \aleph)$ is a behaviour of $P \setminus A$, then there is a trace s' of P which matches s if we ignore the events from A. This trace, together with the refusal set $\aleph$, must be a behaviour of P. Further, we may add events from A to the refusal set. We infer that the failure

$$(s', \aleph \cup [0, end(s', \aleph)) \times A)$$

is a behaviour of P, and hence satisfies specification S.

Although this rule is easy to derive, it is difficult to apply. In chapter 6, we will show how to separate the concerns of concealment and scheduling. We will derive a simple inference rule for hiding, and show how it may be used to structure timed failures specifications.

5.3 Parallel Processes

Alphabet Parallel

If s is a trace of the alphabet parallel combination $P \;{}_A\|_B\; Q$, then we know that the restriction of s to set A must be the trace of events performed by process P. Similarly, the restriction of s to set B is the trace of events performed by Q. We may also infer that s contains only events drawn from the union of these two sets. To summarise, the predicate

$$\exists\, s_P, s_Q \quad \bullet \quad s \upharpoonright A = s_P \wedge s \upharpoonright B = s_Q \wedge s \upharpoonright (A \cup B) = s$$

must hold for s, where s_P and s_Q are traces of P and Q.

Suppose that $(s, \aleph)$ is a behaviour of this parallel combination, and that it corresponds to behaviours $(s_P, \aleph_P)$ and $(s_Q, \aleph_Q)$ of components P and Q. From the semantic equation for this operator, we know that we can choose these component behaviours such that

$$\sigma(\aleph_P) \subseteq A \quad \wedge \quad \sigma(\aleph_Q) \subseteq B$$

Any event from set A will require the cooperation of component P, and any event from set B will require the cooperation of component Q, so we may infer that

$$\aleph_P \subseteq \aleph \upharpoonright A \quad \wedge \quad \aleph_Q \subseteq \aleph \upharpoonright B$$

Finally, an event from $A \cup B$ may be refused by the parallel combination only if it occurs in at least one of these refusal sets.

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s, \aleph) \\ Q \,\mathbf{sat}_\rho\, T(s, \aleph) \end{array}}{P \;{}_A\|_B\; Q \,\mathbf{sat}_\rho\, \exists\, s_P, \aleph_P, s_Q, \aleph_Q \bullet \begin{array}[t]{l} S(s_P, \aleph_P) \wedge T(s_Q, \aleph_Q) \\ \wedge \\ s_P = s \upharpoonright A \wedge s_Q = s \upharpoonright B \\ \wedge \\ s \upharpoonright (A \cup B) = s \\ \wedge \\ \aleph_P \subseteq \aleph \upharpoonright A \wedge \aleph_Q \subseteq \aleph \upharpoonright B \\ \wedge \\ \aleph \upharpoonright (A \cup B) = \aleph_P \cup \aleph_Q \end{array}}$$

Simple Parallel

In the parallel combination $P \parallel Q$ processes P and Q must cooperate on every action that is performed. The relative simplicity of its semantics is reflected in the following inference rule:

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s, \aleph) \\ Q \,\mathbf{sat}_\rho\, T(s, \aleph) \end{array}}{P \parallel Q \;\mathbf{sat}_\rho\, \exists\, \aleph_P, \aleph_Q \bullet \aleph = \aleph_P \cup \aleph_Q \wedge S(s, \aleph_P) \wedge T(s, \aleph_Q)}$$

Interleaving

The interleaved parallel combination $P \,|||\, Q$ may engage in an event a when either P or Q is prepared to engage in a:

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s, \aleph) \\ Q \,\mathbf{sat}_\rho\, T(s, \aleph) \end{array}}{P \,|||\, Q \;\mathbf{sat}_\rho\, \exists\, s_P, s_Q \bullet s \in s_P \,|||\, s_Q \wedge S(s_P, \aleph) \wedge T(s_Q, \aleph)}$$

Recall that $s_P \,|||\, s_Q$ denotes the set of possible interleavings of s_P and s_Q.

Communicating Parallel

In the communicating parallel combination

$$P \underset{C}{\|} Q$$

processes P and Q are reqnired to cooperate on events from set C. If s is a trace of this process, then there must exist traces s_P and s_Q such that

$$(s \upharpoonright C = s_P \upharpoonright C = s_Q \upharpoonright C) \quad \wedge \quad s \setminus C \in (s_P \setminus C \mathbin{|||} s_Q \setminus C)$$

Trace s restricted to events outside set C must be an interleaving of s_P and s_Q, similarly restricted, and all three traces must agree on events from set C. We abbreviate this requirement as $s \in s_P \underset{C}{\|} s_Q$.

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s, \aleph) \\ Q \,\mathbf{sat}_\rho\, T(s, \aleph) \end{array}}{\begin{array}{ll} P \underset{C}{\|} Q \,\mathbf{sat}_\rho\, \exists\, s_P, s_Q, \aleph_P, \aleph_Q \bullet & s \in s_P \underset{C}{\|} s_Q \\ & \wedge \\ & \aleph \upharpoonright C = (\aleph_P \cup \aleph_Q) \upharpoonright C \\ & \wedge \\ & \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C \\ & \wedge \\ & S(s_P, \aleph_P) \\ & \wedge \\ & T(s_Q, \aleph_Q) \end{array}}$$

A timed event (t, a) may be refused if a is in C and either process refuses to cooperate, or a is not in C and both processes refuse to cooperate. If C is the intersection of the process alphabets, we may simplify the consequent:

$$\frac{\begin{array}{l} P \,\mathbf{sat}_\rho\, S(s, \aleph) \\ Q \,\mathbf{sat}_\rho\, T(s, \aleph) \end{array}}{\begin{array}{ll} P \underset{C}{\|} Q \,\mathbf{sat}_\rho\, \exists \aleph_P, \aleph_Q \bullet & \aleph \upharpoonright C = (\aleph_P \cup \aleph_Q) \upharpoonright C \\ & \wedge \\ & \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C \\ & \wedge \\ & S(s \upharpoonright \sigma(P), \aleph_P) \\ & \wedge \\ & T(s \upharpoonright \sigma(Q), \aleph_Q) \end{array}} \quad [\, \sigma(P) \cap \sigma(Q) = C \,]$$

This form of the rnle will be sufficient for most applications.

5.4 Timeouts and Interrupts

Timeout

In the timeout construct $P \overset{t}{\triangleright} Q$, control is transferred to Q unless P performs an external action before time t.

$$\frac{\begin{array}{l} P \ \mathbf{sat}_\rho \ S(s, \aleph) \\ Q \ \mathbf{sat}_\rho \ T(s, \aleph) \end{array}}{\begin{array}{ll} P \overset{t}{\triangleright} Q \ \mathbf{sat}_\rho & begin(s) \leqslant t \wedge S(s, \aleph) \\ & \vee \\ & begin(s) \geqslant (t + \delta) \wedge S(\langle\rangle, \aleph \upharpoonright t) \wedge T((s, \aleph) - (t + \delta)) \end{array}}$$

The δ delay allows us to determine which of the two components has given rise to the current behaviour of $P \overset{t}{\triangleright} Q$; it is a behaviour of P if it starts at or before time t, and a behaviour of Q otherwise.

Timed Interrupt

In the timed interrupt construct

$$P \underset{t}{\not\downarrow} Q$$

control is passed from P to Q at time t, regardless of the progress made by P. Once again, a small delay of δ is associated with the transfer of control.

$$\frac{\begin{array}{l} P \ \mathbf{sat}_\rho \ S(s, \aleph) \\ Q \ \mathbf{sat}_\rho \ T(s, \aleph) \end{array}}{P \underset{t}{\not\downarrow} Q \ \mathbf{sat}_\rho \ begin(s \upharpoonleft t) \geqslant t + \delta \wedge S(s \upharpoonright t, \aleph \upharpoonright t) \wedge T((s, \aleph) - (t + \delta))}$$

No external activity is possible during transfer of control from P to Q, so

$$begin(s \upharpoonleft t) \geqslant t + \delta$$

Any activity before time t must be a possible behaviour of P; any activity after time $t + \delta$ must be a possible behaviour of Q.

Event Interrupt

If $(s, \aleph)$ is a behaviour of the construct

$$P \underset{e \in E}{\nabla} Q_e$$

in which no interrupt events have been observed, then the whole of set E must be available. Otherwise, there is an interrupt (t, e) such that e is the only interrupt event at or before time t; in this case, the subsequent activity must be a possible behaviour of Q_e

$$\frac{\begin{array}{l} P\ \mathbf{sat}_\rho\ S(s,\aleph) \\ \forall e : E \bullet Q_e\ \mathbf{sat}_\rho\ T_e(s,\aleph) \end{array}}{\begin{array}{l} P \underset{e \in E}{\nabla} Q_e \\ \quad \mathbf{sat}_\rho \\ \qquad \begin{array}{l} E \cap \sigma(s,\aleph) = \{\} \wedge S(s,\aleph) \\ \vee \\ \exists\, t : \mathit{TIME}\, ; e : E \bullet s \upharpoonright t \downharpoonright E = \langle (t,e) \rangle \ \wedge \\ \qquad\qquad\qquad E \cap \sigma(\aleph \upharpoonright t) = \{\} \ \wedge \\ \qquad\qquad\qquad \mathit{begin}(s \upharpoonleft t) \geqslant t + \delta \ \wedge \\ \qquad\qquad\qquad S(s \upharpoonright t \setminus e, \aleph \upharpoonright t) \ \wedge \\ \qquad\qquad\qquad T_e((s,\aleph) - (t+\delta)) \end{array} \end{array}} \quad [\, E \cap \sigma(P) = \{\}\,]$$

Apart from the occurrence of the interrupt event any observation up until time t must be a behaviour of P.

If only one interrupt event is offered, then we may eliminate one of the existential quantifiers in the consequent of the rule:

$$\frac{\begin{array}{l} P\ \mathbf{sat}_\rho\ S(s,\aleph) \\ Q\ \mathbf{sat}_\rho\ T(s,\aleph) \end{array}}{P \underset{e}{\nabla} Q\ \mathbf{sat}_\rho \ \begin{array}{l} e \notin \sigma(s,\aleph) \wedge S(s,\aleph) \\ \vee \\ \exists\, t : \mathit{TIME} \bullet s \upharpoonright t \downharpoonright e = \langle (t,e) \rangle \ \wedge \\ \qquad\qquad e \notin \sigma(\aleph \upharpoonright t) \ \wedge \\ \qquad\qquad \mathit{begin}(s \upharpoonleft t) \geqslant t + \delta \ \wedge \\ \qquad\qquad S(s \upharpoonright t \setminus e, \aleph \upharpoonright t) \ \wedge \\ \qquad\qquad T((s,\aleph) - (t+\delta)) \end{array}} \quad [\, e \notin \sigma(P)\,]$$

In either case, we include the assumption that P may not interrupt itself as a side condition to the rule.

5.5 Recursive Processes

In chapter 3, we showed how the theory of metric spaces may be used to give a semantics to recursively defined processes. To reason about the properties of these processes, we give a simple topology to the space TM_F and establish a theory of recursion induction, in the style of [Roscoe 82].

Recursion Induction

We will require the following definitions, taken from [Sutherland 75]:

Definition 5.1 A topological space $T = (A, \mathcal{T})$ consists of a non-empty set A together with a fixed collection $\mathcal{T}$ of subsets of A satisfying

1. $A, \{\} \in \mathcal{T}$
2. the intersection of any two sets in $\mathcal{T}$ is again in $\mathcal{T}$
3. the union of any collection of sets in $\mathcal{T}$ is again in $\mathcal{T}$

◇

We refer to the elements of $\mathcal{T}$ as the *open sets* of T. The *closed sets* of T are given by $\{A - U \mid U \in \mathcal{T}\}$. A mapping between topological spaces is *continuous* if inverse images of open sets are themselves open:

Definition 5.2 If $T_1 = (A_1, \mathcal{T}_1)$ and $T_2 = (A_2, \mathcal{T}_2)$ are topological spaces, then a mapping $f : A_1 \rightarrow A_2$ is continuous with respect to topologies $\mathcal{T}_1$ and $\mathcal{T}_2$ if

$$U \in \mathcal{T}_2 \quad \Rightarrow \quad f^{-1}(U) \in \mathcal{T}_1$$

◇

We may give a topology to the metric space (A, d) by defining $\mathcal{T}$ to be the set of *d-open* subsets of A. If we define

Definition 5.3 If $M = (A, d)$ is a metric space and ϵ is a strictly positive real number, then the *open ϵ-ball neighbourhood of a in M* is the set

$$B_\epsilon(d\,;a) \;\widehat{=}\; \{x : A \mid d(x, a) < \epsilon\}$$

◇

then we may characterise the d-open sets as unions of open balls. This is a consequence of the following definition:

Definition 5.4 A subset U of a metric space $M = (A, d)$ is *d-open in M* if given any $a \in U$ there exists $\epsilon_a > 0$ such that $B_{\epsilon_a}(d\,;a) \subset U$. ◇

The following theorems are taken from [Roscoe 82] and [Reed 88]:

Theorem 5.5 Let $M = (A, d)$ be a complete metric space, and let TV be the topological space $(\{true, false\}, \mathcal{T})$ where

$$\mathcal{T} \;\hat{=}\; \{\{\}, \{false\}, \{true, false\}\}$$

If $F : M \to T$ is continuous with respect to the d-open topology and $\mathcal{T}$, and the set $\{a \in A \mid F(a) = true\}$ is nonempty, then

$$(\forall x : A \bullet F(x) = true \Rightarrow F(C(x)) = true) \;\; \Rightarrow \;\; F(fix(C)) = true$$

for any contraction mapping $C : M \to M$. ♡

Theorem 5.6 If F is a mapping from the complete metric space (TS_F, d) to TV such that for any Y in TS_F

$$F(Y) = false \;\; \Rightarrow \;\; \exists\, t : TIME \bullet \forall\, Y' : TS_F \bullet Y \upharpoonright t = Y \upharpoonright t' \Rightarrow F(Y') = false$$

then F is continuous. ♡

Recall that the metric d upon TS_F was defined using the *before* operator on sets of failures. $Y \upharpoonright t = Y' \upharpoonright t$ if failure sets Y and Y' agree up until time t; these Y' form an open ball around Y in the metric space. If F is such that whenever $F(Y) = false$ there is an open ball around Y whose image is $\{false\}$, then F is continuous.

We identify predicates on timed failure sets with mappings from TS_F to the space of truth values TV.

Definition 5.7 A predicate R on elements of TS_F is a mapping from the space of timed failure sets TS_F to the space of truth values TV,

$$(\{true, false\}, \{\{\}, \{false\}, \{true, false\}\})$$

We say that R is a *continuous* predicate if it is a continuous mapping in the sense of definition 5.2. ◇

Definition 5.8 A predicate R is *plausible* if R is continuous and

$$\exists\, Y : TS_F \;\;\bullet\;\; R(Y) = true$$

◇

Immediate Recursion

To establish that a *plausible* predicate correctly describes a well-defined recursive process $\mu X \circ P$, it is sufficient to show that R is preserved by the mapping corresponding to X and P. If P is *constructive* for X then the following inference rule is valid:

Rule 5.9

$$\frac{\forall Y : TS_F \bullet R(Y) \Rightarrow R(\mathcal{F}_T[\![P]\!]\rho[Y/X])}{R(\mathcal{F}_T[\![\mu X \circ P]\!]\rho)} \quad [\, R \text{ plausible} \,]$$

$\triangle$

Proof If P is constructive for X, then the mapping $\lambda Y \bullet \mathcal{F}_T[\![P]\!]\rho[Y/X]$ is a contraction mapping on TS_F, by lemma 3.14. If R is plausible, then it is a continuous mapping from TS_F to TV such that the set $\{Y : TS_F \mid R(Y) = \mathit{true}\}$ is nonempty, and we have assumed that

$$\forall Y : TS_F \bullet R(Y) \Rightarrow R(\mathcal{F}_T[\![P]\!]\rho[Y/X])$$

We may apply theorem 5.5 and deduce that the rule is sound. $\square$

In our proof system, we wish to establish that a predicate holds not of a process, but of a typical behaviour of that process: we wish to show that a process satisfies a behavioural specification. In this case, our proof obligation can be simplified. We need only show that the specification is preserved by each recursive call:

Rule 5.10

$$\frac{X \, \mathbf{sat}_\rho \, S(s, \aleph) \Rightarrow P \, \mathbf{sat}_\rho \, S(s, \aleph)}{\mu X \circ P \, \mathbf{sat}_\rho \, S(s, \aleph)}$$

$\triangle$

Proof We recall the definition of the $\mathbf{sat}_\rho$ operator

$$P \, \mathbf{sat}_\rho \, S(s, \aleph) \;\widehat{=}\; \forall s, \aleph \bullet (s, \aleph) \in \mathcal{F}_T[\![P]\!] \bullet S(s, \aleph)$$

We claim that any predicate of the form

$$R(Y) \;\widehat{=}\; \forall s, \aleph \bullet (s, \aleph) \in Y \Rightarrow S(s, \aleph)$$

is plausible in TS_F. Suppose that $Y \in TS_F$ and $R(Y) = \mathit{false}$, then

$$\exists s, \aleph \bullet (s, \aleph) \in Y \wedge \neg S(s, \aleph)$$

There must be a behaviour $(s, \aleph)$ in Y which does not meet S. If we choose a time $t > end(s, \aleph)$ then

$$Y' \upharpoonright t = Y \upharpoonright t \;\;\Rightarrow\;\; (s, \aleph) \in Y' \wedge \neg\, S(s, \aleph)$$

and $R(Y') = \mathit{false}$ for all Y' in an open ball of radius 2^{-t} around Y. By theorem 5.6, R is a continuous predicate. We know that $\{\} \in TS_F$, and it is easy to see that

$$R(\{\}) \;\; = \;\; \mathit{true}$$

Hence predicate R is plausible. This reduces the proof obligation to an instance of rule 5.9. Hence this rule is also sound. □

This gives a sufficient condition for the recursive process $\mu\, X \circ P$ to satisfy the specification $S(s, \aleph)$ on timed failures.

Delayed Recursion

The delayed recursion operator associates a delay of δ with each recursive call; the mapping on TS_F corresponding to a recursion $\mu\, X \bullet P$ is given by

$$M_\delta(X, P)\rho \;\; \widehat{=} \;\; W_\delta \cdot \lambda\, Y \bullet \mathcal{F}_T[\![P]\!]\rho[Y/X]$$

where the following equivalence holds for W_δ:

$$W_\delta \;\; \equiv \;\; \lambda\, Y \bullet \{(s, \aleph) \mid begin(s) \geqslant \delta \wedge ((s, \aleph) - \delta) \in Y\}$$

This mapping is a contraction mapping on TS_F for any choice of X, P; there is no need to establish that term P is constructive for the recursive variable. With an argument similar to that presented for rule 5.9, we may establish an inference rule for this operator:

Rule 5.11

$$\frac{\forall\, Y : TS_F \bullet R(Y) \Rightarrow R(\mathcal{F}_T[\![P]\!]\rho[W_\delta(Y)/X])}{R(\mathcal{F}_T[\![\,\mu\, X \bullet P]\!]\rho)} \quad [\, R \text{ plausible} \,]$$

△

From the proof of rule 5.10, we see that any such specification corresponds to a plausible predicate on elements of TS_F. If we choose R such that

$$R(Y) \;\; \widehat{=} \;\; \forall\, s, \aleph \bullet (s, \aleph) \in Y \Rightarrow S(s, \aleph)$$

then we may derive an inference rule for behavioural specifications:

Rule 5.12

$$\frac{\forall X : TS_F \bullet X \,\mathbf{sat}_\rho\, (S((s,\aleph) - \delta) \wedge begin(s) \geqslant \delta) \Rightarrow P \,\mathbf{sat}_\rho\, S(s,\aleph)}{\mu X \bullet P \,\mathbf{sat}_\rho\, S(s,\aleph)}$$

$\triangle$

Proof Similar to the proof of rule 5.10. $\square$

Mutual Recursion

We restrict our attention to those recursive equation sets in which the vector of terms is constructive for the vector of variables. We wish to establish results about processes of the form $\langle X_i = P_i \rangle_j$: the j component of the process vector defined by equation set $\{X_i = P_i\}$. To do this, we will need to establish similar results about each component of the vector $\underline{P}$.

To establish that a vector of predicates $\underline{R}$ correctly describes the fixed point of $M(\underline{X}, \underline{P})$, it is sufficient to show that $\underline{R}$ is preserved by $M(\underline{X}, \underline{P})$, and that each R_i is a plausible predicate.

Rule 5.13

$$\frac{(\forall i \bullet R_i(Y_i)) \Rightarrow \forall j \bullet R_j(\mathcal{F}_T[\![P_j]\!]\rho[\underline{Y}/\underline{X}])}{\underline{R}(\mathcal{F}_T[\![\mu \underline{X} \circ \underline{P}]\!]\rho)} \quad [\, R_i \text{ plausible, for all } i \,]$$

$\triangle$

Proof Assume that each R_i is plausible, and that

$$(\forall i : I \bullet R_i(Y_i)) \;\Rightarrow\; \forall j : I \bullet R_j(\mathcal{F}_T[\![P_j]\!]\rho[\underline{Y}/\underline{X}])$$

We claim that

$$(\forall i : I \bullet R_i(Y_i)) \;\Rightarrow\; \forall j : I \bullet R_j(\mathcal{F}_T[\![Q_j]\!]\rho[\underline{Y}/\underline{X}])$$

where vector $\underline{Q}$ is as defined in the proof of the Unique Fixed Point Theorem, theorem 3.31:

$$Q_i \;\widehat{=}\; P_i[Q_j/X_j \mid j \in \mathrm{seg}(i)]$$

To establish this result, we proceed by transfinite induction, with inductive set J defined by

$$J \;\widehat{=}\; \{k : I \bullet (\forall i : I \bullet R_i(Y_i)) \Rightarrow R_k(\mathcal{F}_T[\![Q_k]\!]\rho[\underline{Y}/\underline{X}])\}$$

We assume that $\mathrm{seg}(k) \subseteq J$, and observe that

$$\begin{aligned}\mathcal{F}_T[\![Q_k]\!]\rho[\underline{Y}/\underline{X}] &= \mathcal{F}_T[\![P_k[Q_l/X_l \mid l \in \mathrm{seg}(k)]]\!]\rho[\underline{Y}/\underline{X}] \\ &= \mathcal{F}_T[\![P_k]\!]\rho[\underline{Y}/\underline{X}][\mathcal{F}_T[\![Q_l]\!]\rho[\underline{Y}/\underline{X}]/X_l \mid l \in \mathrm{seg}(k)]\end{aligned}$$

Define a secondary vector $\underline{Z}$ by

$$\begin{aligned}Z_l \;\hat{=}\; & Y_l & (l \notin \mathrm{seg}(k)) \\ & \mathcal{F}_T[\![Q_l]\!]\rho[\underline{Y}/\underline{X}] & (l \in \mathrm{seg}(k))\end{aligned}$$

By our inductive hypothesis,

$$\forall i : I \bullet R_i(Y_i) \;\Rightarrow\; \forall i : I \bullet R_i(Z_i)$$

Whence our original assumption about $\underline{P}$ yields

$$\begin{aligned}(\forall i : I \bullet R_i(Y_i)) &\Rightarrow R_k(\mathcal{F}_T[\![P_k]\!]\rho[\underline{Z}/\underline{X}]) \\ &\Rightarrow R_k(\mathcal{F}_T[\![Q_k]\!]\rho[\underline{Y}/\underline{X}])\end{aligned}$$

hence $k \in J$, and the claim follows by transfinite induction. From the definition of the metric $\underline{d}$ upon the product space TS_F^I, we may establish that

$$(\forall i : I \bullet R_i \text{ plausible}) \;\Rightarrow\; \underline{R} \text{ plausible}$$

We have established that $M(\underline{X}, \underline{Q})\rho$ preserves $\underline{R}$, hence $\underline{R}$ must hold of the fixed point of this mapping, by theorem 5.5. But from the proof of the Unique Fixed Point Theorem we learn that

$$\mathit{fix}(M(\underline{X}, \underline{Q})\rho) \;=\; \mathit{fix}(M(\underline{X}, \underline{P})\rho)$$

We may conclude that the rule is sound. □

We may derive a rule for behavioural specifications by making a suitable choice for predicate $\underline{R}$:

Rule 5.14

$$\frac{(\forall i : I \bullet X_i \;\mathbf{sat}_\rho\; S_i(s, \aleph)) \Rightarrow \forall i : I \bullet P_i \;\mathbf{sat}_\rho\; S_i(s, \aleph)}{\langle X_i = P_i \rangle_j \;\mathbf{sat}_\rho\; S_j(s, \aleph)}$$

△

The proof that this rule is sound, as an instance of the previous rule, is entirely similar to the derivation of rule 5.10 from rule 5.9.

5.6 Soundness and Completeness

Our proof system has two desirable properties:

Theorem 5.15 The set of inference rules presented in this chapter is sound with respect to the semantic equations given in chapter 2. The truth of each rule may be established from the semantic equation for the corresponding operator, without recourse to the axioms of TM_F. $\heartsuit$

Theorem 5.16 The set of inference rules presented in this chapter is complete with respect to the semantic equations given in chapter 2. Any property that is true of every behaviour of a process P may be established using these rules. $\heartsuit$

Soundness

The presentation of the proof system has been chosen to emphasise the correspondence between inference rules and semantic equations. To see that each rule is sound, we have only to examine the defining equation for the operator in question. As an example, consider the rule for simple parallel combination:

$$\frac{\begin{array}{l} P \, \mathbf{sat}_\rho \, S(s,\aleph) \\ Q \, \mathbf{sat}_\rho \, T(s,\aleph) \end{array}}{P \parallel Q \; \mathbf{sat}_\rho \, \exists \aleph_P, \aleph_Q \bullet \aleph = \aleph_P \cup \aleph_Q \wedge S(s,\aleph_P) \wedge T(s,\aleph_Q)}$$

This operator was given the following semantics:

$$\mathcal{F}_T[\![P \parallel Q]\!]\rho \;\; \widehat{=} \;\; \{(s,\aleph_P \cup \aleph_Q) \mid (s,\aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \wedge (s,\aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho\}$$

A simple logical deductiou will suffice the establish the validity of the inference rule. Assume the two antecedents of the rule and suppose that $(s,\aleph)$ is a behaviour of $P \parallel Q$ in environment ρ. By the semantic equation,

$$\exists \aleph_P, \aleph_Q \;\; \bullet \;\; (s,\aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \wedge (s,\aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho \wedge \aleph = \aleph_P \cup \aleph_Q$$

From the antecedents, we obtain

$$\exists \aleph_P, \aleph_Q \;\; \bullet \;\; S(s,\aleph_P) \wedge T(s,\aleph_Q) \wedge \aleph = \aleph_P \cup \aleph_Q$$

We conclude that the rule is sound.

Similar arguments may be presented for the other inference rules, with the exception of the recursion induction rules; soundness proofs for these rules were presented in section 5.5.

Completeness

We claim that, if every behaviour of a process P meets predicate $S(s, \aleph)$, then the inference rules given in this chapter are sufficient to prove that P **sat** $S(s, \aleph)$.

Lemma 5.17 If $P \in TCSP$ meets the requirement that each recursion is constructive, then we may use the inference rules to establish that

$$P \;\; \mathbf{sat}_\rho \;\; (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho$$

for any environment ρ. ♡

Proof We proceed by structural induction upon the syntax $TCSP$. The result is easy to establish for basic processes. As an example, consider the case of the deadlock process. The semantic equation for this operator yields that

$$(s, \aleph) \in \mathcal{F}_T[\![STOP]\!]\rho \;\; \Leftrightarrow \;\; s = \langle\rangle$$

The inference rules

$$\frac{}{STOP\; \mathbf{sat}_\rho\; s = \langle\rangle} \qquad\qquad \frac{\begin{array}{l} P\; \mathbf{sat}_\rho\; S(s, \aleph) \\ S(s, \aleph) \Rightarrow T(s, \aleph) \end{array}}{P\; \mathbf{sat}_\rho\; T(s, \aleph)}$$

are enough to establish that $STOP \;\; \mathbf{sat}_\rho \;\; (s, \aleph) \in \mathcal{F}_T[\![STOP]\!]\rho$.

For compound processes, we assume that the result holds for each component, and apply the appropriate inference rule. Consider the case of the simple parallel operator, which is associated with the following inference rule:

$$\frac{\begin{array}{l} P\; \mathbf{sat}_\rho\; S(s, \aleph) \\ Q\; \mathbf{sat}_\rho\; T(s, \aleph) \end{array}}{P \parallel Q\; \mathbf{sat}_\rho\; \exists \aleph_P, \aleph_Q \bullet \aleph = \aleph_P \cup \aleph_Q \wedge S(s, \aleph_P) \wedge T(s, \aleph_Q)}$$

By our inductive hypothesis, the inference rules are enough to establish that

$$\begin{array}{lll} P & \mathbf{sat}_\rho & (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \\ Q & \mathbf{sat}_\rho & (s, \aleph) \in \mathcal{F}_T[\![Q]\!]\rho \end{array}$$

With these instantiations, we obtain the following consequent

$$\begin{array}{lll} P \parallel Q & \mathbf{sat}_\rho & \exists \aleph_P, \aleph_Q \bullet \aleph = \aleph_P \cup \aleph_Q \;\wedge (s, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho \\ & & \qquad\qquad\qquad\qquad\quad \wedge (s, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho \end{array}$$

From the semantic equation for this operator,

$$(s,\aleph) \in \mathcal{F}_T\llbracket P \parallel Q \rrbracket\rho \;\Leftrightarrow\; \exists\, \aleph_P, \aleph_Q \bullet \aleph = \aleph_P \cup \aleph_Q \wedge (s,\aleph_P) \in \mathcal{F}_T\llbracket P \rrbracket\rho \;\wedge$$
$$(s,\aleph_Q) \in \mathcal{F}_T\llbracket Q \rrbracket\rho$$

We conclude that

$$P \parallel Q \;\; \mathbf{sat}_\rho \;\; (s,\aleph) \in \mathcal{F}_T\llbracket P \parallel Q \rrbracket\rho$$

may be established using the inference rules of the TM_F proof system.

To see that the result is true for recursive processes, recall that the semantics of a recursive process is the unique fixed point of the corresponding mapping in the model TM_F. For example, the semantics of the instant recursion $\mu X \circ P$ is defined to be the unique fixed point of the mapping $M(X,P)\rho$, where

$$M(X,P)\rho \;\;\widehat{=}\; \lambda\, Y \bullet \mathcal{F}_T\llbracket P \rrbracket\rho[Y/X].$$

The following inference rule may be applied if P is constructive for X:

$$\frac{X \;\mathbf{sat}_\rho\; S(s,\aleph) \Rightarrow P \;\mathbf{sat}_\rho\; S(s,\aleph)}{\mu X \circ P \;\mathbf{sat}_\rho\; S(s,\aleph)}$$

We instantiate S with the specification $(s,\aleph) \in \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho$ and claim that the antecedent holds. Observe that

$$X \;\mathbf{sat}_\rho\, (s,\aleph) \in \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho \;\;\Rightarrow\;\; \rho\llbracket X \rrbracket \subseteq \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho$$

The semantics of each operator is defined *pointwise* upon sets of timed failures. As a resnlt, the mapping on TM_F corresponding to any *TCSP* term must be monotonic with respect to the subset ordering. Hence

$$M(X,P)\rho\,(\rho\llbracket X \rrbracket) \;\subseteq\; M(X,P)\rho\,(\mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho)$$

Expanding the definition of $M(X,P)\rho$, we obtain

$$\begin{array}{rrcl} & \mathcal{F}_T\llbracket P \rrbracket\rho[\rho\llbracket X \rrbracket/X] & \subseteq & M(X,P)\rho\, \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho \\ \Rightarrow & \mathcal{F}_T\llbracket P \rrbracket\rho & \subseteq & M(X,P)\rho\, \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho \\ \Rightarrow & \mathcal{F}_T\llbracket P \rrbracket\rho & \subseteq & \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho \\ \Rightarrow & P & \mathbf{sat}_\rho & (s,\aleph) \in \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho \end{array}$$

Hence the antecedent of the rule holds for this specification; we may infer that

$$\mu X \circ P \;\; \mathbf{sat}_\rho \;\; (s,\aleph) \in \mathcal{F}_T\llbracket \mu X \circ P \rrbracket\rho$$

The result follows by structural induction. □

We have shown that the inference rules of our proof system are enough to establish

$$P \quad \mathrm{sat}_\rho \quad (s,\aleph) \in \mathcal{F}_T[\![P]\!]\rho$$

for any P in *TCSP* providing that the body of any recursive term is constructive for the corresponding term variable. If a behavioural specification S holds of the timed failures semantics of P, then

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

The logical rule for weakening specifications (used in the proof of the previous lemma) enables us to complete the proof of

$$P \quad \mathrm{sat}_\rho \quad S(s,\aleph)$$

using only the inference rules of our proof system. We conclude that the proof system for TM_F presented in this chapter is complete for constructive recursive processes, with respect to the semantic function $\mathcal{F}_T$.

5.7 Timewise Refinement

A formal specification of a complex system will consist of several behavioural specifications, each of which may be established separately. If we wish to prove that

$$P \quad \mathrm{sat} \quad S(s,\aleph) \wedge T(s,\aleph)$$

then it will suffice to show

$$P \,\mathrm{sat}\, S(s,\aleph) \quad \wedge \quad P \,\mathrm{sat}\, T(s,\aleph)$$

Some of these specifications may not require the full expressive power of Timed Failures model. If this is the case, then we may use the hierarchy of models beneath TM_F to simplify our proof obligations.

If a predicate upon timed traces can be established without refusal information, then we may construct a proof in the Timed Traces model TM_T. The nature of the projection mappings ensures that this proof will be valid in TM_F. Similarly, if a property may be established without timing information, we may choose to construct a proof in the untimed Failures model M_F, or the untimed Traces model M_T. Of these models, M_T is the most useful for simplifying timed failures specifications; M_F is often inappropriate, and timed trace requirements may be established using a simplified version of the TM_F proof system.

$$\begin{tikzcd}
 & TM_F \arrow[dl] \arrow[dr] & \\
TM_T \arrow[dr] & & M_F \arrow[dl] \\
 & M_T &
\end{tikzcd}$$

Figure 5.1: the models beneath TM_F

The Timed Traces model is complicated by the need to record the times at which events become available; this information is required for the semantics of hiding and sequential composition. For any event a, the notation $\hat{a}$ denotes the communication of a at the first moment of availability. For example,

$$\mathcal{T}_T[\![P \parallel Q]\!] \quad \widehat{=} \quad \{s \mid \exists\, s_P, s_Q \bullet s = s_P \diamond s_Q \wedge s_P \in \mathcal{T}_T[\![P]\!] \wedge s_Q \in \mathcal{T}_T[\![Q]\!]\}$$

where $s_P \diamond s_Q$ is a timed trace with the same timed events as s_P and s_Q, such that the n^{th} element of $s_P \diamond s_Q$ is hatted if and only if the n^{th} element of either s_P or s_Q is hatted. For the Timed Traces model, refinemeut to timed failures is easy, but proofs remain complicated.

In the untimed failure (tr, X), the refusal set X is a set of events that may be refused following the observation of trace tr. In a timed context, X corresponds to the set of events that may be refused after all iuternal activity has ceased. Without a stability value, we have no record of internal activity; an untimed liveness reqnirement may insist only that an event is offered *eventually*; this may prove inadequate in the specification of a real-time system. Nevertheless, a projection mapping from TM_F to M_F might be used to establish important properties of a real-time system. For example, it may be possible to establish deadlock freedom using the algebraic properties of untimed CSP, instead of the timed satisfaction relation **sat**.

If a requirement can be established by reasoning within the untimed Traces model, then we may employ a simple *syntactic abstraction* from the timed syntax *TCSP* to the nntimed syntax *CSP*. In [Schneider 89], the author develops a theory of *timewise refinement* based upou the Timed Failures-Stability model TM_{FS}. In this section, we develop a similar theory for TM_F and exhibit a refinement proof rule for untimed safety specifications. We begiu by presenting a syntax for untimed CSP, together with a semautic function for Reed's untimed Traces model.

Traces Model

We give an extended syntax CSP for a language of untimed CSP terms, to match the syntax for $TCSP$ given in chapter 2:

$$\begin{array}{lll}
P ::= & STOP \mid SKIP \mid X \mid & \text{atoms} \\
& a \rightarrow P \mid P ; P \mid P \lightning P & \text{sequential composition} \\
& P \Box P \mid P \sqcap P \mid a : A \rightarrow P_a \mid & \text{alternation} \\
& P \parallel P \mid P \,{}_A\|_B\, P \mid P \interleave P \mid P \underset{A}{\parallel} P \mid & \text{parallel composition} \\
& P \setminus A \mid f(P) \mid f^{-1}(P) \mid & \text{abstraction and renaming} \\
& \mu X \bullet P \mid (X_i = P_i)_j & \text{recursion}
\end{array}$$

This is an extension of the syntax presented in [Reed 88]: apart from term variables and the new parallel operator, we have added an ***untimed interrupt*** operator $\lightning$ which may interrupt a process at any time during its execution.

We define a semantic function from CSP to the Traces model M_T, using environments to bind term variables:

$$\begin{array}{rcl}
\mathcal{T}[\![STOP]\!]\rho & \hat{=} & \{\langle\rangle\} \\
\mathcal{T}[\![SKIP]\!]\rho & \hat{=} & \{\langle\rangle, \langle\checkmark\rangle\} \\
\mathcal{T}[\![a \rightarrow P]\!]\rho & \hat{=} & \{\langle\rangle\} \cup \{\langle a\rangle^\frown tr \mid tr \in \mathcal{T}[\![P]\!]\rho\} \\
\mathcal{T}[\![P ; Q]\!]\rho & \hat{=} & \{tr \mid tr \in \mathcal{T}[\![P]\!]\rho \wedge \checkmark \notin \sigma(tr)\ \} \\
& & \cup \\
& & \{tr_P{}^\frown tr_Q \mid tr_P{}^\frown\langle\checkmark\rangle \in \mathcal{T}[\![P]\!]\rho \wedge tr_Q \in \mathcal{T}[\![Q]\!]\rho\} \\
\mathcal{T}[\![P \lightning Q]\!]\rho & \hat{=} & \{tr_P{}^\frown tr_Q \mid tr_P \in \mathcal{T}[\![P]\!]\rho \wedge tr_Q \in \mathcal{T}[\![Q]\!]\rho\} \\
\mathcal{T}[\![P \Box Q]\!]\rho & \hat{=} & \mathcal{T}[\![P]\!]\rho \cup \mathcal{T}[\![Q]\!]\rho \\
\mathcal{T}[\![P \sqcap Q]\!]\rho & \hat{=} & \mathcal{T}[\![P]\!]\rho \cup \mathcal{T}[\![Q]\!]\rho \\
\mathcal{T}[\![a : A \rightarrow P_a]\!]\rho & \hat{=} & \{\langle a\rangle^\frown tr \mid a \in A \wedge tr \in \mathcal{T}[\![P_a]\!]\rho\}
\end{array}$$

The untimed interrupt construct $P \lightning Q$ may transfer control from P to Q after any sequence of events; an arbitrary trace of this process may be any trace of P, followed by any trace of Q.

$$\begin{array}{rcl}
\mathcal{T}[\![P \parallel Q]\!]\rho & \mathrel{\widehat=} & \mathcal{T}[\![P]\!]\rho \cap \mathcal{T}[\![Q]\!]\rho \\[1ex]
\mathcal{T}[\![P \;{}_A\|_B\; Q]\!]\rho & \mathrel{\widehat=} & \{tr \mid tr \downharpoonright (A \cup B) = tr \;\wedge\; tr \downharpoonright A \in \mathcal{T}[\![P]\!]\rho \\
 & & \qquad\qquad\qquad\qquad \wedge\; tr \downharpoonright B \in \mathcal{T}[\![Q]\!]\rho\} \\[1ex]
\mathcal{T}[\![P \;|||\; Q]\!]\rho & \mathrel{\widehat=} & \{tr \mid \exists\, tr_P, tr_Q \bullet tr \textit{ interleaves } (tr_P, tr_Q) \;\wedge \\
 & & \qquad\qquad\qquad tr_P \in \mathcal{T}[\![P]\!]\rho \wedge tr_Q \in \mathcal{T}[\![Q]\!]\rho\} \\[1ex]
\mathcal{T}[\![P \underset{A}{\|} Q]\!]\rho & \mathrel{\widehat=} & \{tr \mid \exists\, tr_P, tr_Q \bullet tr \setminus A \textit{ interleaves } (tr_P \setminus A, tr_Q \setminus A) \;\wedge \\
 & & \qquad\qquad\qquad tr \downharpoonright A = tr_P \downharpoonright A = tr_Q \downharpoonright A \;\wedge \\
 & & \qquad\qquad\qquad tr_P \in \mathcal{T}[\![P]\!]\rho \wedge tr_Q \in \mathcal{T}[\![Q]\!]\rho\} \\[1ex]
\mathcal{T}[\![P \setminus A]\!]\rho & \mathrel{\widehat=} & \{tr \setminus A \mid tr \in \mathcal{T}[\![P]\!]\rho\} \\[1ex]
\mathcal{T}[\![f(P)]\!]\rho & \mathrel{\widehat=} & \{f(tr) \mid tr \in \mathcal{T}[\![P]\!]\rho\} \\[1ex]
\mathcal{T}[\![f^{-1}(P)]\!]\rho & \mathrel{\widehat=} & \{tr \mid f(tr) \in \mathcal{T}[\![P]\!]\rho\} \\[1ex]
\mathcal{T}[\![\mu X \bullet P]\!]\rho & \mathrel{\widehat=} & \text{the unique fixed point of the mapping on } M_T \\
 & & \text{corresponding to } X,\ P \text{ and } \rho \\[1ex]
\mathcal{T}[\![\langle X_i = P_i\rangle_j]\!]\rho & \mathrel{\widehat=} & \text{the } j^{\text{th}} \text{ component of the unique fixed point} \\
 & & \text{of the mapping on } M_T^I \text{ corresponding to } \underline{X}, \\
 & & \underline{P}, \text{ and } \rho
\end{array}$$

The subsidiary relation *interleaves* is defined in [Hoare 85]: tr *interleaves* (tr_P, tr_Q) precisely when tr is an interleaving of tr_P and tr_Q.

If $S(tr)$ represents a behavioural specification on untimed traces, then we may define a satisfaction relation:

$$P \textbf{ sat}_\rho\ S(tr) \;\mathrel{\widehat=}\; \forall\, tr \in \Sigma^* \;\bullet\; tr \in \mathcal{T}[\![P]\!]\rho \Rightarrow S(tr)$$

The choice of free variable identifies the model employed; we write tr to denote an arbitrary untimed trace. If the interpretation of **sat** is not obvious from the context, we will decorate it with the name of a semantic model. Using this relation, we may obtain a compositional proof system, similar to the TM_F proof system presented earlier in this chapter: e.g.

$$\frac{P \textbf{ sat}_\rho\ S(tr)}{a \rightarrow P \textbf{ sat}_\rho\ tr = \langle\rangle \vee tr = \langle a\rangle^\frown tr' \wedge S(tr')}$$

The inference rules for the other operators are straightforward, except in the case of recursion. For a recursive process to have a well-defined semantics, the body of the recursive definition should be *guarded* for the recursive variable; a term P is guarded for a variable X if each occurrence of X in P is prefixed by some observable event. If term P is guarded for X, then the following rule may be applied:

$$\frac{X\ \mathbf{sat}_p\ S(tr) \Rightarrow P\ \mathbf{sat}_p\ S(tr)}{\mu X \bullet P\ \mathbf{sat}_p\ S(tr)} \quad [\ STOP\ \mathbf{sat}\ S(tr)\ ]$$

An untimed trace specification is satisfiable iff it is satisfied by $STOP$; this is a consequence of the following axiom for M_T:

$$\forall Y \in M_T \bullet \langle\rangle \in Y$$

The requirement that S is satisfiable is expressed by the side condition of the rule.

This proof system is considerably simpler than the proof system for TM_F. If an untimed safety specification may be established within the untimed traces model, then we may remove the timing information from the syntax of the process and apply the inference rules for M_T.

Syntactic Abstraction

We may define a syntactic abstraction $\Theta : TCSP \to CSP$ by structural induction upon the timed syntax:

$$\begin{aligned}
\Theta(\perp) &\mathrel{\widehat=} STOP \\
\Theta(STOP) &\mathrel{\widehat=} STOP \\
\Theta(SKIP) &\mathrel{\widehat=} SKIP \\
\Theta(WAIT\ t) &\mathrel{\widehat=} SKIP \\
\Theta(X) &\mathrel{\widehat=} X
\end{aligned}$$

We abstract away timing information, identifying any form of sequential composition with the immediate transfer of control:

$$\begin{aligned}
\Theta(a \to P) &\mathrel{\widehat=} a \to \Theta(P) \\
\Theta(a \xrightarrow{t} P) &\mathrel{\widehat=} a \to \Theta(P) \\
\Theta(P \mathbin{;} Q) &\mathrel{\widehat=} \Theta(P) \mathbin{;} \Theta(Q) \\
\Theta(P \mathbin{\boldsymbol{;}} Q) &\mathrel{\widehat=} \Theta(P) \mathbin{;} \Theta(Q)
\end{aligned}$$

The mapping distributes through all of the standard operators:

$$
\begin{array}{rcl}
\Theta(P \Box Q) & \mathrel{\widehat{=}} & \Theta(P) \Box \Theta(Q) \\
\Theta(P \sqcap Q) & \mathrel{\widehat{=}} & \Theta(P) \sqcap \Theta(Q) \\
\Theta(a : A \rightarrow P_a) & \mathrel{\widehat{=}} & a : A \rightarrow \Theta(P_a) \\
\Theta(P \parallel Q) & \mathrel{\widehat{=}} & \Theta(P) \parallel \Theta(Q) \\
\Theta(P \mathbin{{}_A\|_B} Q) & \mathrel{\widehat{=}} & \Theta(P) \mathbin{{}_A\|_B} \Theta(Q) \\
\Theta(P \mathbin{|||} Q) & \mathrel{\widehat{=}} & \Theta(P) \mathbin{|||} \Theta(Q) \\
\Theta(P \underset{A}{\parallel} Q) & \mathrel{\widehat{=}} & \Theta(P) \underset{A}{\parallel} \Theta(Q) \\
\Theta(P \setminus A) & \mathrel{\widehat{=}} & \Theta(P) \setminus A \\
\Theta(f(P)) & \mathrel{\widehat{=}} & f(\Theta(P)) \\
\Theta(f^{-1}(P)) & \mathrel{\widehat{=}} & f^{-1}(\Theta(P)) \\
\Theta(\mu X \bullet P) & \mathrel{\widehat{=}} & \mu X \bullet \Theta(P) \\
\Theta(\mu X \circ P) & \mathrel{\widehat{=}} & \mu X \bullet \Theta(P) \\
\Theta(\langle X_i = P_i \rangle_j) & \mathrel{\widehat{=}} & \langle X_i = \Theta(P)_i \rangle_j
\end{array}
$$

The timeout construct $P \overset{t}{\triangleright} Q$ may offer the user a choice between P and Q, or may behave as Q, depending on whether the timeout has occurred. Without timing information there is a nondeterministic choice between these two alternatives:

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

The timed interrupt operators are mapped to untimed interrupts:

$$
\begin{array}{rcl}
\Theta(P \underset{t}{\lightning} Q) & \mathrel{\widehat{=}} & \Theta(P) \lightning \Theta(Q) \\
\Theta(P \underset{e}{\nabla} Q) & \mathrel{\widehat{=}} & \Theta(P) \lightning e \rightarrow \Theta(Q)
\end{array}
$$

The indexed nondeterministic choice and nondeterministic delay operators are mapped to the obvious targets:

$$
\begin{array}{rcl}
\Theta(\bigsqcap_{i \in I} P_i) & \mathrel{\widehat{=}} & \bigsqcap_{i \in I} \Theta(P_i) \\
\Theta(\mathit{WAIT}\ T) & \mathrel{\widehat{=}} & \mathit{SKIP}
\end{array}
$$

Timewise Refinement

If a specification $S(s)$ on timed traces is independent of timing information, then we may transform S into a behavioural specification on untimed traces. We define a simple projection mapping on elements of $T\Sigma^*_{\leqslant}$:

Definition 5.18

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

$\diamondsuit$

The mapping *tstrip* removes the time values from the trace. If the truth of a behavioural specification S is independent of these values, we say that S is a ***time-independent*** specification.

Definition 5.19 A behavioural specification $S(s)$ is time-independent iff

$$\forall s_1, s_2 : T\Sigma^*_{\leqslant} \bullet tstrip(s_1) = tstrip(s_2) \Rightarrow S(s_1) \equiv S(s_2)$$

$\diamondsuit$

If $S(s)$ is a time-independent specification on timed traces, then it will prove convenient to define a corresponding condition upon untimed traces:

Definition 5.20 If $S(s)$ is a timed trace specification, then

$$\Phi S(tr) \mathrel{\widehat=} \forall s \in T\Sigma^*_{\leqslant} \bullet tstrip(s) = tr \Rightarrow S(s)$$

$\diamondsuit$

An untimed trace tr meets the specification ΦS iff every assignment of time values to the events in tr produces a timed trace that meets S. Recall that the sequence of time values must be non-decreasing if the result is to be a valid timed trace.

Trace operators which do not refer to the times at which events occur may be applied to both timed and untimed traces. For example, the restriction and counting operators are defined on Σ^* by

$$\begin{aligned} \langle\rangle \upharpoonright A &\mathrel{\widehat=} \langle\rangle \\ (\langle a\rangle^\frown tr) \upharpoonright A &\mathrel{\widehat=} \begin{cases} \langle a\rangle^\frown(tr \upharpoonright A) & \text{if } a \in A \\ tr \upharpoonright A & \text{otherwise} \end{cases} \end{aligned}$$

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

If a timed trace specification $S(s)$ is constructed using such operators, then the untimed trace condition will often be a consequence of $S(tr)$. As an example, consider the timed trace specification

$$S(s) \;\widehat{=}\; (last(s) = a) \Rightarrow s \downarrow b = 0$$

A process satisfies this specification if whenever we record an a event, we find that no b events have been recorded. It is easy to establish that

$$(last(tr) = a) \Rightarrow tr \downarrow b = 0 \quad \Rightarrow \quad \Phi S(tr)$$

for any untimed trace tr.

If the image of S under Φ is satisfied by the untimed equivalent of a timed process P, then we might expect that P satisfies S. However, if we choose

$$P \;\widehat{=}\; (a \rightarrow STOP \,|||\, SKIP) \,;\, b \rightarrow STOP$$

then $\Theta(P)$ satisfies ΦS, where S is as defined above, but P does not satisfy S in TM_F. With instantaneous sequential composition, events b may be observed at the same time as a, and so may appear first in the trace. The trace $\langle(0, b), (0, a)\rangle$ is a possible observation of P, and does not meet S.

This problem may occur whenever we use the instantaneous form of sequential composition. We could remove this operator from the syntax of Timed CSP and use only the delayed form; this is the approach taken in [Schneider 89]. We choose instead to retain it, for greater flexibility in process descriptions, and identify the situations in which it may be safely applied.

Definition 5.21 A process P is Θ-safe iff

$$\forall s, \aleph \;\bullet\; (s, \aleph) \in \mathcal{F}_T[\![P]\!] \Rightarrow tstrip(s) \in \mathcal{T}[\![P]\!]$$

$\Diamond$

A direct application of this definition would be impractical; fortunately, this property may be established by a simple inspection of the process syntax.

Definition 5.22

- For any time t or set of times T, the terms $STOP$, $SKIP$, $\perp$, $WAIT\,t$, and X are all $\checkmark$-guarded.
- If P and Q are both $\checkmark$-guarded, then the terms $a \rightarrow P$, $a \xrightarrow{t} P$, $P \,\Box\, Q$, $P \sqcap Q$, $P \,\text{;}\, Q$, $P \| Q$, $P\,{}_A\|_B\, Q$, $P ||| Q$, $P \setminus A$, $f(P)$, $f^{-1}(P)$, $\mu X \bullet P$, $\mu X \circ P$, $P \overset{t}{\rhd} Q$, $P \underset{t}{\not\downarrow} Q$, and $P \underset{a \in A}{\nabla} Q_a$ are all $\checkmark$-guarded.

- P_a and P_i are $\checkmark$-guarded, for each event $a \in A$ or index i, then the terms $a : A \to P_a$, $\prod_{i \in I} P_i$, and $\langle X_i = P_i \rangle_j$ are all $\checkmark$-guarded.
- If P and Q are both $\checkmark$-guarded and t is strictly positive, then the terms $WAIT\ t\ ; P$, $P ; WAIT\ t$, and $P ; WAIT\ t\ ; Q$ are all $\checkmark$-guarded.

$\diamond$

A term P is *$\checkmark$-guarded* if every instantaneous sequential composition is accompanied by a delay operator. This syntactic property is a sufficient condition for a process to be Θ-safe:

Lemma 5.23 For any P in *TCSP*,

$$P \text{ is } \checkmark\text{-guarded} \quad \Rightarrow \quad P \text{ is } \Theta\text{-safe}$$

$\heartsuit$

This result follows directly from the semantic equations for the Timed Failures and untimed Traces models. We may now exhibit a refinement proof rule for untimed safety specifications:

Rule 5.24

$$\frac{\Theta(P) \ \mathbf{sat}\ \Phi S(tr)}{P\ \mathbf{sat}\ S(s)} \quad [\ P \text{ is } \Theta\text{-safe}\ ]$$

$\triangle$

If we can establish that the abstraction of a Θ-safe process P meets ΦS, then we may infer that P also satisfies S. The antecedent of the rule is a proof obligation in the untimed Traces model, the consequent is a proof obligation in TM_F.

Proof To see that this rule is sound, consider the following diagram:

$$\begin{tikzcd}
\text{timed process} \arrow[r, "\mathbf{sat}"] \arrow[d, "\Theta\text{-safe}"] & \text{timed specification} \\
\text{untimed process} \arrow[r, "\mathbf{sat}"] & \text{untimed specification} \arrow[u, "\text{definition of } \Phi"']
\end{tikzcd}$$

Figure 5.2: Timewise Refinement

If s is a timed trace of a Θ-safe process, then $tstrip(s)$ will be a trace of $\Theta(P)$. If $\Theta(P)$ satisfies the behavioural specification $\Phi S(tr)$, then we may infer that S holds of trace s. The consequent part of the rule follows immediately. $\square$

5.8 Example

Consider the following implementation of a timed sensitive vending machine:

$$
\begin{array}{rcl}
TSVM & \mathrel{\widehat{=}} & coin \xrightarrow{t_1} (reset \xrightarrow{t_3} TSVM \\
 & & \qquad\quad \overset{t_2}{\rhd} \\
 & & \qquad\quad coke \xrightarrow{t_4} TSVM)
\end{array}
$$

The company that operates this machine requires that every drink is paid for in advance, and that the machine is ready to accept another order within a certain time t_4. In section 4.4, we formalised these requirements as separate behavioural specifications:

$$
\begin{array}{rcl}
SAFE(s) & \mathrel{\widehat{=}} & s \downarrow coke \leqslant s \downarrow coin \\[1em]
NEXT(s, \aleph) & \mathrel{\widehat{=}} & \langle (t, coke) \rangle \text{ in } s \Rightarrow coin \notin \sigma(\aleph \upharpoonright t + t_4) \\
 & & \qquad\qquad\qquad\quad \vee \\
 & & \qquad\qquad\qquad\quad coin \in \sigma(s \upharpoonright t)
\end{array}
$$

The machine is supplied with the following manufacturer's guarantee: if a coin is inserted at time t, then a drink is ready before time $t + t_5$, providing that the user does not trigger a *reset* during the interval $[t + t_1, t + t_1 + t_2]$. This requirement corresponds to the following behavioural specification:

$$
OKAY(s, \aleph) \;\equiv\; \left.\begin{array}{l}
\langle (t, coin) \rangle \text{ in } s \\
\wedge \\
reset \notin \sigma(s \uparrow [t + t_1, t + t_1 + t_2])
\end{array}\right\} \Rightarrow
\begin{array}{l}
coke \notin \sigma(\aleph \upharpoonright t + t_5) \\
\vee \\
coke \in \sigma(s \upharpoonright t)
\end{array}
$$

We would like to establish that the suggested implementation *TSVM* meets the following behavioural specification:

$$
SPEC(s, \aleph) \;\mathrel{\widehat{=}}\; SAFE(s) \wedge NEXT(s, \aleph) \wedge OKAY(s, \aleph)
$$

Safety

The safety specification *SAFE* is independent of timing considerations. Although it is possible to show that *TSVM* satisfies this specification using the timed failures proof system, it will be easier to establish the result by timewise refinement. We observe that *TSVM* is $\checkmark$-guarded and hence Θ-safe, and that

$$
tr \downarrow coke \leqslant tr \downarrow coin \;\Rightarrow\; \Phi SAFE(tr)
$$

Applying the abstraction mapping Θ reveals that

$$\Theta(TSVM) \;\equiv\; SVM$$

where SVM is the untimed vending machine of section 2.8:

$$\begin{array}{lcl} SVM & \widehat{=} & coin \rightarrow (PAID \,\square\, reset \rightarrow SVM \\ & & \qquad\quad \sqcap \\ & & \qquad\quad coke \rightarrow SVM) \end{array}$$

An application of rule 5.24 reduces our proof obligation to

$$SVM \quad \mathbf{sat} \quad tr \downarrow coke \leqslant tr \downarrow coin$$

We observe that this recursion is guarded, and apply the inference rule for recursion in M_T. We have now to prove that

$$\begin{array}{ll} coin \rightarrow (PAID \,\square\, reset \rightarrow X & \\ \qquad\quad \sqcap & \mathbf{sat}_\rho \quad tr \downarrow coke \leqslant tr \downarrow coin \\ \qquad\quad coke \rightarrow X) & \end{array}$$

under the assumption that

$$X \quad \mathbf{sat}_\rho \quad tr \downarrow coke \leqslant tr \downarrow coin$$

This result may be established using the inference rules for event prefix, deterministic choice, and nondeterministic choice in M_T.

Liveness

The rest of the proof must be conducted within the Timed Failures model. We begin by observing that the body of the recursion SVM is *constructive*, providing that $t_1 > 0$. We may then apply the inference rule for recursion, reducing our proof obligation to

$$\begin{array}{ll} coin \xrightarrow{t_1} (reset \xrightarrow{t_3} X & \\ \qquad\quad \overset{t_2}{\rhd} & \mathbf{sat}_\rho \quad NEXT(s,\aleph) \wedge OKAY(s,\aleph) \\ \qquad\quad coke \xrightarrow{t_4} X) & \end{array}$$

under the assumption that

$$X \quad \mathbf{sat}_\rho \quad NEXT(s,\aleph) \wedge OKAY(s,\aleph)$$

If we restrict our attention to the second of these specifications, we may apply the inference rule for event prefix to yield:

$$\begin{array}{lll} reset \xrightarrow{t_3} X & \mathbf{sat}_\rho & s = \langle\rangle \\ & & \vee \\ & & s = \langle(t, reset)\rangle^\frown s' \wedge OKAY((s', \aleph) - (t + t_3)) \end{array}$$

Similarly, we may easily establish that

$$\begin{array}{lll} coke \xrightarrow{t_4} X & \mathbf{sat}_\rho & s = \langle\rangle \wedge coke \notin \sigma(\aleph) \\ & & \vee \\ & & s = \langle(t, coke)\rangle^\frown s' \wedge coke \notin \sigma(\aleph \upharpoonright t) \\ & & \qquad\qquad\qquad \wedge\ OKAY((s', \aleph) - (t + t_4)) \end{array}$$

Applying the inference rule for the timeout operator, we may deduce that

$$\begin{array}{ll} reset \xrightarrow{t_3} X & \\ \overset{t_2}{\triangleright} & \mathbf{sat}_\rho \quad A(s, \aleph) \vee B(s, \aleph) \vee C(s, \aleph) \\ coke \xrightarrow{t_4} X & \end{array}$$

where

$$\begin{array}{lll} A(s, \aleph) & \hat{=} & begin(s) \leqslant t_2 \ \wedge \\ & & s = \langle(t, reset)\rangle^\frown s' \ \wedge \\ & & OKAY((s', \aleph) - (t_3 + t)) \\ \\ B(s, \aleph) & \hat{=} & s = \langle\rangle \ \wedge \ coke \notin \sigma(\aleph - (t_2 + \delta)) \\ \\ C(s, \aleph) & \hat{=} & s - (t_2 + \delta) = \langle(t, coke)\rangle^\frown s' \ \wedge \\ & & coke \notin \sigma(\aleph - (t_2 + \delta) \upharpoonright t) \ \wedge \\ & & OKAY((s', \aleph) - (t_4 + t)) \end{array}$$

An application of the rule for event prefix reduces our proof obligation to

$$\left.\begin{array}{ll} s = \langle\rangle & \\ \vee & \\ s = \langle(t', coin)\rangle^\frown s' \wedge & A((s', \aleph) - (t_1 + t')) \\ & \vee \\ & B((s', \aleph) - (t_1 + t')) \\ & \vee \\ & C((s', \aleph) - (t_1 + t')) \end{array}\right\} \quad \Rightarrow \quad OKAY(s, \aleph)$$

The specification $OKAY$ is vacuons for the empty trace; we have only to prove

$$\left.\begin{array}{l} \langle (t_0, coin)\rangle \in s \;\wedge \\ reset \notin \sigma(s \uparrow [t_0 + t_1 \,.\, t_0 + t_1 + t_2]) \;\wedge \\ s = \langle (t', coin)\rangle^\frown s' \;\wedge\; \begin{array}{l} A((s', \aleph) - (t_1 + t')) \\ \vee \\ B((s', \aleph) - (t_1 + t')) \\ \vee \\ C((s', \aleph) - (t_1 + t')) \end{array} \end{array}\right\} \Rightarrow \begin{array}{l} coke \notin \sigma(\aleph \upharpoonright t_0 + t_s) \\ \vee \\ coke \in \sigma(s \upharpoonright t_0) \end{array}$$

If $t' \neq t_0$, then the result is easily established by expanding A, B, and C. Suppose then that $t' = t_0$; onr assumption that

$$reset \notin \sigma(s \uparrow [t' + t_1, t' + t_1 + t_2])$$

contradicts $A(s, \aleph)$, so we have only to show that

$$\left.s = \langle (t', coin)\rangle^\frown s' \;\wedge\; \begin{array}{l} B((s', \aleph) - (t_1 + t')) \\ \vee \\ C((s', \aleph) - (t_1 + t')) \end{array}\right\} \Rightarrow \begin{array}{l} coke \notin \sigma(\aleph \upharpoonright t' + t_s) \\ \vee \\ coke \in \sigma(s \upharpoonright t') \end{array}$$

From the definitions of B and C we obtain:

$$\begin{array}{l} s' = \langle\rangle \;\wedge\; coke \notin \sigma(\aleph - (t_1 + t_2 + t' + \delta)) \\ \vee \\ s' - (t_1 + t_2 + t' + \delta) = \langle (t'', coke)\rangle^\frown s'' \end{array}$$

The result follows, providing that

$$t_s \;\geqslant\; t_1 + t_2 + \delta$$

If a coin is inserted at time t, then we cannot guarantee to provide a drink any earlierthan $t + t_1 + t_2 + \delta$. We must allow a delay of t_1 for the coin to be inserted, a delay of t_2 for the coin to drop, and a delay of at least δ for control to be passed to the dispensing process.

The proof of $NEXT(s, \aleph)$ is similar, although an additional constraint must be addedto ensure that this specification is preserved by each recursive call. We may conclude that the fnll specification is satisfied:

$$TSVM \quad \text{sat} \quad SPEC(s, \aleph)$$

providing that the above condition upon t_s is observed.

Chapter 6

Structuring Specifications

If we wish to produce a readable specification of a large system, then we must take care to present our description in a clear, structured fashion. At each level of abstraction, we identify the interfaces between system components and conceal any events which are not of interest. We express our specification as a series of *service* specifications, each describing the service provided by a particular component of the system. In this way, we may refine a description of the service provided by a system towards a satisfactory implementation.

6.1 Abstraction

The hiding operator provides the mechanism for abstraction in Timed CSP; the expression $P \setminus A$ denotes a process that behaves as P, except that

* events from A occur as soon as they become available
* only events from outside A are observed

In section 5.2 we gave an inference rule for this operator that was easy to derive, but difficult to apply. We can achieve a significant reduction in complexity if we separate the concerns of concealment and scheduling. To this end, we define a predicate act_A which holds of any A-*active* behaviour:

Definition 6.1 $\mathrm{act}_A(s, \aleph) \mathrel{\widehat{=}} [0, end(s, \aleph)) \times A \subseteq \aleph$ $\Diamond$

A behaviour $(s, \aleph)$ is A-active if all events from set A occur as soon as they become available. If we wish to establish that $P \setminus A$ satisfies a specification $S(s, \aleph)$, it is sufficient to show that

* $S(s, \aleph)$ holds for all of the A-active behaviours of P
* $S(s, \aleph)$ is unaffected by the concealment of events from A

The second condition is satisfied if the truth of the specification is unaffected by the removal of A's events from the trace and refusal.

Definition 6.2 A behavioural specification $S(s, \aleph)$ is A-independent iff

$$\forall s : T\Sigma^*_{\leqslant} ; \aleph : RSET \bullet S(s, \aleph \cup [0, end(s)) \times A) \Rightarrow S(s \setminus A, \aleph)$$

$\diamond$

If S describes a service provided across an interface that is disjoint from A, then S should be A-independent. The following abbreviation will prove convenient:

Definition 6.3 $S(s, \aleph) \mathbin{\backslash\!\backslash} A \mathrel{\widehat{=}} \mathrm{act}_A(s, \aleph) \Rightarrow S(s, \aleph)$ $\diamond$

This states that the specification $S(s, \aleph)$ holds whenever the current behaviour is A-active. We may now present a simple proof rule for the hiding operator:

Rule 6.4

$$\frac{P \ \mathbf{sat}_\rho \ S(s, \aleph) \mathbin{\backslash\!\backslash} A}{P \setminus A \ \mathbf{sat}_\rho \ S(s, \aleph)} \quad [\, S \text{ is } A\text{-independent} \,]$$

$\triangle$

If an A-independent specification S holds for all A-active behaviours of a term P, then we may infer that $P \setminus A$ satisfies $S(s, \aleph)$.

Example

Suppose that process P satisfies the following specification:

$$\begin{aligned} T(s, \aleph) \mathrel{\widehat{=}} \ & a \notin \sigma(\aleph \uparrow [1, 2)) \vee a \in \sigma(s \uparrow [0, 2)) \\ & \wedge \\ & \langle (t, a) \rangle \ \mathrm{in} \ s \Rightarrow b \notin \sigma(\aleph \uparrow [t + 1, t + 2)) \\ & \qquad\qquad\qquad \vee \\ & \qquad\qquad\qquad b \in \sigma(s \uparrow [t + 1, t + 2)) \end{aligned}$$

In this case, the event a is available from time 1 to time 2, unless it has already occurred. Further, if an a is observed at any time t, then b either occurs or is available during the interval $[t + 1, t + 2)$.

If we consider only the a-active behaviours of P, then a is present for the dnration of refusal set $\aleph$. If the behaviour $(s, \aleph)$ extends beyond time 1, we know that a mnst occnr before time 2. We can show that

$$\begin{array}{lll} T(s,\aleph) \mathbin{\backslash\!\!\backslash} a & \equiv & end(s,\aleph) \geqslant 1 \Rightarrow a \in \sigma(s \uparrow [0,2)) \\ & & \wedge \\ & & ((t,a)) \text{ in } s \Rightarrow b \notin \sigma(\aleph \uparrow [t+1, t+2)) \\ & & \qquad\qquad \vee \\ & & \qquad\qquad b \in \sigma(s \uparrow [t+1, t+2)) \end{array}$$

and it is easy to establish that $T(s, \aleph) \mathbin{\backslash\!\!\backslash} a \;\Rightarrow\; S(s, \aleph) \mathbin{\backslash\!\!\backslash} a$, where

$$\begin{array}{lll} S(s,\aleph) & \widehat{=} & \exists\, t : [0,2) \bullet b \notin \sigma(\aleph \uparrow [t+1, t+2)) \\ & & \qquad\qquad \vee \\ & & \qquad\qquad b \in \sigma(s \uparrow [t+1, t+2)) \end{array}$$

It is clear that $S(s, \aleph)$ is an a-independent specification, so we may apply the new inference rule for the hiding operator to obtain

$$\begin{array}{lll} P \setminus a & \text{sat} & \exists\, t : [0,2) \bullet b \notin \sigma(\aleph \uparrow [t+1, t+2)) \\ & & \qquad\qquad \vee \\ & & \qquad\qquad b \in \sigma(s \uparrow [t+1, t+2)) \end{array}$$

The event b is made available, or is observed, during the interval $[t + 1, t + 2)$, where $0 \leqslant t < 2$.

6.2 Scheduling

The form of liveness condition employed above is both awkward and inadequate if we wish to abstract from the events concerned. If we have that

$$P \quad \text{sat} \quad a \notin \sigma(\aleph \uparrow I) \vee a \in \sigma(s \uparrow J)$$

then we may infer only that $P \setminus a$ performs event a at some time during the interval J. If we intend to conceal an event a, then any liveness specification involving a should address the time at which a becomes available.

Instead of requiring that an event a is offered during a fixed interval *unless* it is observed, we may insist that a is available *until* it is observed.

Definition 6.5

$$a \text{ from } t\ (s, \aleph) \quad \widehat{=} \quad a \notin \sigma(\aleph \uparrow [t, begin(s \uparrow [t, \infty) \downharpoonright a)))$$

◇

The right-hand predicate states that event a is absent from refusal set $\aleph$ between time t and the time at which the next a is observed; it is easy to see that the process $WAIT\ t\ ;\ a \rightarrow STOP$ will satisfy this specification.

This form of liveness specification allows us to determine the precise time of occurrence of hidden events. If a process P satisfies the liveness condition a from t, then we may infer that the event a occurs at time t in all a-active behaviours of P. If we define

Definition 6.6 $\quad a \text{ at } t\ (s, \aleph) \ \widehat{=}\ \langle (t, a) \rangle \textbf{ in } s$ $\quad\diamond$

then we obtain

$$a \text{ from } t\ (s, \aleph) \quad\Rightarrow\quad end(s, \aleph) > t \Rightarrow a \text{ at } t\ (s, \aleph) \setminus\!\!\setminus a$$

If a becomes available at t, and it is hidden, then it will be observed at t, if the current observation extends far enough. The above implication is a consequence of the following result:

Lemma 6.7 For any $S(s, \aleph)$, if $end(s, \aleph) > t$ then

$$(S(s, \aleph) \vee a \text{ from } t\ (s, \aleph)) \setminus\!\!\setminus a \quad\Rightarrow\quad (S(s, \aleph) \vee a \text{ at } t\ (s, \aleph)) \setminus\!\!\setminus a$$

$$(S(s, \aleph) \wedge a \text{ from } t\ (s, \aleph)) \setminus\!\!\setminus a \quad\Rightarrow\quad (S(s, \aleph) \wedge a \text{ at } t\ (s, \aleph)) \setminus\!\!\setminus a$$

$\heartsuit$

Proof From the definition of $\setminus\!\!\setminus$, we obtain

$$([0, end(s, \aleph)) \times \{a\}) \subseteq \aleph$$

Our assumption that $end(s, \aleph) > t$ allows us to infer that $a \in \sigma(\aleph \uparrow t)$. From definition 6.5, we deduce that

$$[t, begin(s \uparrow [t, \infty) \downharpoonright a)) \ = \ \{\}$$

and hence that $\langle (t, a) \rangle$ in s. The proof may be completed using tautologies of the propositional calculus. $\square$

We may allow a process to withdraw the offer of an event if it has not been accepted within a given period of time, or if another event has been observed.

Definition 6.8

$$a \text{ from } t \text{ until } t'\ (s, \aleph) \ \widehat{=}\ a \notin \sigma(\aleph \uparrow [t, min\{t', begin(s \uparrow [t, \infty) \downharpoonright a)\}))$$

$\diamond$

If tbe offer of event a has not been accepted by time t', the process may retract without violating the liveness specificatiou. This corresponds to an application of the timeout operator.

$$WAIT\, t_1 \,;\, ((a \rightarrow STOP) \overset{t_2}{\triangleright} STOP) \quad \textbf{sat} \quad a \text{ from } t_1 \text{ until } t_1 + t_2$$

The event a is enabled at time t_1 and, if it has uot been performed, disabled at time $t_1 + t_2$.

Definition 6.9

$$a \text{ from } t \text{ until } b\ (s, \aleph) \quad \widehat{=} \quad a \notin \sigma(\aleph \uparrow [t, begin(s \uparrow [t, \infty) \downharpoonright \{a, b\})))$$

◇

If a process is to satisfy this specification, event a must become available at time t, and must remain available until either a or b is observed. We may combine this condition with the possibility of a timeout:

Definition 6.10

$$\begin{aligned} & a \text{ from } t \text{ until } t' \text{ or } b\ (s, \aleph) \\ & \qquad \widehat{=} \quad a \notin \sigma(\aleph \uparrow [t, min\{t', begin(s \uparrow [t, \infty) \downharpoonright \{a, b\})\})) \end{aligned}$$

◇

It is worth observing that:

$$a \text{ from } t \text{ until } t' \text{ or } b \quad \equiv \quad a \text{ from } t \text{ until } t' \vee a \text{ from } t \text{ until } b$$

More usually, we will wish to insist that a becomes available at some time during a fixed interval I. We can capture this requirement witb a simple quantification:

Definition 6.11

$$a \text{ from } I\ (s, \aleph) \quad \widehat{=} \quad \exists\, t : I \bullet a \text{ from } t$$

◇

As an example, consider the process P defined by:

$$P \quad \widehat{=} \quad (WAIT\,[1, 2)\,;(a \rightarrow STOP \,\Box\, b \rightarrow STOP)) \overset{4}{\triangleright} STOP$$

We may use the timed failures proof system to show that

$$P \quad \mathbf{sat} \quad a \text{ from } [1,2) \text{ until } 4 \text{ or } b \ (s, \aleph)$$

The event a is enabled at some time between 1 and 2, and disabled at time 4 or when b occurs, whichever is the sooner.

With another existential quantification, we may allow the offer of a to be withdrawn at any time during an *interval* J. Further, we may require that event a made available until some event from set B is observed:

Definition 6.12

$$a \text{ from } I \text{ until } J \text{ or } B \ \mathrel{\widehat{=}} \ \exists\, t : I ;\, t' : J \bullet \exists\, b : B \bullet a \text{ from } t \text{ until } t' \text{ or } b$$

◇

The following equivalence confirms that existential quantification over set B has captured the required constraint:

$$a \text{ from } t \text{ until } B \ (s, \aleph) \ \equiv \ a \notin \sigma(\aleph \uparrow [t, begin(s \uparrow [t, \infty) \downarrow \{a\} \cup B)))$$

The offer of a may be withdrawn following the occurrence of any event from set B. This justifies the following definition:

Definition 6.13

$$A \text{ from } I \text{ until } J \text{ or } B \ \mathrel{\widehat{=}} \ \forall\, a : A \bullet a \text{ from } I \text{ until } J \text{ or } (A \cup B)$$

◇

If a process is to satisfy this specification, the *whole* of set A must become available at some time t in I, and remain available until some time t' in J, unless an event from $A \cup B$ is observed. The sequence of quantifiers is the most appropriate for our needs. the specification "A from I $(s, \aleph)$" is satisfied by any process that becomes ready for every event from A at some time during I. An example might be

$$WAIT\,[0,1)\,;\, in.m : in.M \rightarrow P_m$$

This process becomes ready to accept any message m from set M on channel in at some time during interval $[0,1)$.

A similar generalisation may be applied to the 'at' construct; we may replace the single event and time with a set and interval:

Definition 6.14 $\quad A \text{ at } I \ (s, \aleph) \ \mathrel{\widehat{=}} \ \exists\, a : A ;\, t : I \bullet a \text{ at } t \ (s, \aleph)$ ◇

This condition is true if some event from set A is observed at some time during the interval I.

We may generalise the statement of lemma 6.7. If $(s, \aleph)$ is an A-active behaviour, and every event from A is made available at some time during interval I, then some event from A must be observed during I.

Lemma 6.15 If $end(s, \aleph) > end(I)$ then

$$(S(s,\aleph) \vee A \text{ from } I\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \vee A \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

$$(S(s,\aleph) \wedge A \text{ from } I\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \wedge A \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

♡

Some care must be taken in the presence of an 'until' clause: if the interval in which an event is enabled intersects with the interval in which the offer may be withdrawn, then there is no guarantee that the event will occur.

Lemma 6.16 If $end(s, \aleph) > end(I)$ and $begin(J) > end(I)$ then

$$(S(s,\aleph) \vee A \text{ from } I \text{ until } J\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \vee A \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

$$(S(s,\aleph) \wedge A \text{ from } I \text{ until } J\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \wedge A \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

♡

If the offer of events from set A may be withdrawn on the observation of an event from B, then we know that some event from $A \cup B$ will occur in an A-active behaviour.

Lemma 6.17 If $end(s, \aleph) > end(I)$ then

$$(S(s,\aleph) \vee A \text{ from } I \text{ until } B\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \vee A \cup B \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

$$(S(s,\aleph) \wedge A \text{ from } I \text{ until } B\ (s,\aleph)) \setminus\!\!\setminus A \;\Rightarrow\; (S(s,\aleph) \wedge A \cup B \text{ at } I\ (s,\aleph)) \setminus\!\!\setminus A$$

♡

In each of these lemmata, we cannot assert that any event is observed unless the experiment is of sufficient duration: $end(s, \aleph) > end(I)$.

The 'at' and 'from' expressions are *macro*[1] statements in our timed failures specification language. We may use such expressions to make our specifications more palatable, although it will often be necessary to expand them in the course of a proof.

[1]From *macrose*: syntactic sugar.

6.3 A Specification Language

We may consider the macro expressions of the previous section as functions defined upon a typical timed failure. If we *lift* the boolean operators to take functions as arguments, we may obtain a simple language for timed specifications. For example, the requirement that

$$P \quad \text{sat} \quad a \text{ from } t\ (s, \aleph) \wedge \neg\ (b \text{ at } t\ (s, \aleph))$$

may be shortened to

$$P \quad \text{sat} \quad a \text{ from } t \wedge \neg\ (b \text{ at } t)$$

Not only are such specifications easier to read, but also they are open to interpretation in other models. This language is a first order logic with time-valued variables, comparable to those of [Hooman 90] and [Jahanian & Mok 86].

There is no need to extend the satisfaction relation between processes and specifications. If F is defined upon the set of *all* timed failures, then

$$P \text{ sat } F(s, \aleph) \quad \Leftrightarrow \quad \forall s, \aleph \bullet (s, \aleph) \in \mathcal{F}_T[\![P]\!] \Rightarrow F(s, \aleph)$$

In the case when F is applied to the typical failure $(s, \aleph)$, as in the example above, we will omit the function argument. We may employ the inference rules of the timed failures proof system to establish results expressed in our new language. We may also derive laws for reasoning about higher-order objects such as 'at' and 'from' expressions. If we define

$$\begin{aligned} \text{time}\,(s, \aleph) &\ \hat{=}\ \ end(s, \aleph) \\ \text{active}\,A\ (s, \aleph) &\ \hat{=}\ \ [0, end(s, \aleph)) \subseteq \aleph \end{aligned}$$

then lemma 6.7 gives rise to a simple example

$$\text{active}\,A \ \wedge\ A \text{ from } t \ \wedge\ \text{time} > t \ \Rightarrow\ A \text{ at } t$$

If events from set A do not require the cooperation of the environment, and all events from A are made available at time t, then some event from A will be observed at time t.

In the course of chapter 7, we will require a number of functions to extract information from a timed trace. One such function has already been defined,

$$A \text{ at } I\ (s, \aleph) \ \equiv\ \exists\, a : A \bullet a \in \sigma(s \uparrow I)$$

which returns a boolean expression whose value is *true* precisely when some event from set A is present in trace s restricted to interval I. The remaining functions will be defined using projection mappings upon timed failnres.

If M is a projection mapping from timed failures to time values, and φ is a predicate upon time values, then

$$F(s,\aleph) \;\widehat{=}\; \varphi(M(s,\aleph))$$

defines a behavioural specification on timed failures. A similar construction may be used for projection mappings whose results are timed events, sequences of events, or even components of events.

We will often wish to consider the sequence of data values passed along a certain channel c during a particular interval I. The following projection mapping yields precisely this information:

$$\text{data } c \text{ during } I\ (s,\aleph) \;\widehat{=}\; data(s \downharpoonright c.\Sigma \uparrow I)$$

where *data* is defined by

$$\begin{aligned} data(\langle\rangle) &\;\widehat{=}\; \langle\rangle \\ data(\langle(t,c.a)\rangle^\frown s) &\;\widehat{=}\; \langle a\rangle^\frown data(s) \end{aligned}$$

The result is a sequence of values drawn from the datatype of values permitted on channel c.

Another useful projection mapping returns the last timed event from set A observed during interval I, or strictly before time t:

$$\begin{aligned} \text{last } A \text{ during } I\ (s,\aleph) &\;\widehat{=}\; foot(s \downharpoonright A \uparrow I) \\ \text{last } A \text{ before } t\ (s,\aleph) &\;\widehat{=}\; foot(s \downharpoonright A \uparrow [0,t)) \end{aligned}$$

Similarly, we may define a projection mapping 'count' that yields the number of occurrences of events from a given set during a specified interval.

$$\begin{aligned} \text{count } A \text{ during } I\ (s,\aleph) &\;\widehat{=}\; \#(s \downharpoonright A \uparrow I) \\ \text{count } A \text{ before } t\ (s,\aleph) &\;\widehat{=}\; \#(s \downharpoonright A \uparrow [0,t)) \end{aligned}$$

Again, we may choose to count only the events observed before some time t.

It will prove convenient to give names to the projection mappings from timed events to times and events.

$$\begin{aligned} \text{time of } (t,a) &\;\widehat{=}\; t \\ \text{name of } (t,a) &\;\widehat{=}\; a \end{aligned}$$

for any timed event (t,a). We will add to this list of projection mappings and functions whenever we encounter constraints that cannot be expressed using our existing vocabulary.

6.4 Example

Consider a simple communications network, consisting of a sender process S, a receiver process R, and two communnications media, *M1* and *M2*. The network accepts messages on channel *in*, and delivers messages on channel *out*. The sender transmits each message to the receiver using either *M1* or *M2*. The choice of medium may depend upon message length, current load, or internal errors; in any case, the user is not informed. At this level of abstraction, the choice of medium is nondeterministic.

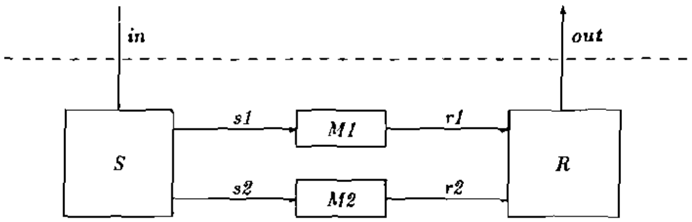

Figure 6.1: Transmission with a choice of media

Both transmission media are reliable—no messages will be lost—but each is associated with a different nondeterministic delay. The first will transmit messages from channel *s1* to channel *r1* with a delay of between *1* and *2* seconds, while the other will transmit messages from *s2* to *t2* with a delay of between *0.1* and *0.3* seconds. These channels are invisible to the network user.

We require that any implementation of this network should deliver a message within *3* seconds of its arrival on channel *in*. This requirement is captured by the following behavionral specification:

$$LIVE \;\hat{=}\; in.m \text{ at } t \Rightarrow out.m \text{ from } (t, t+3)$$

If a message is input at time t, it will be available on channel *out* at some time between t and $t + 3$. We will assume that all times are given in seconds.

We assnme that all messages are of message type M, and choose $c.M$ to denote the set of communications possible on channel c,

$$c.M \;\hat{=}\; \{c.m \mid m \in M\}$$

where c is any of *s1*, *s2*, *r1*, *r2*.

Each transmission medium is initially ready to accept a message, and is always prepared to accept a new message within *0.1* seconds. If medium *M1* accepts a message m on channel *s1* at time t, then it must begin to offer the communication *r1*.m at some time between $t + 1$ and $t + 2$, and become ready for a new message before time $t + 0.1$. We may capture these requirements with the following behavioural specification upon medium *M1*:

$$\begin{array}{lll} M1 & \text{sat} & s1.M \text{ from } 0 \\ & & \wedge \\ & & s1.m \text{ at } t \Rightarrow \begin{array}[t]{l} r1.m \text{ from } (t+1, t+2) \\ \wedge \\ s1.M \text{ from } (t, t+0.1) \end{array} \end{array}$$

The faster medium *M2* satisfies a similar specification, undertaking to deliver a message after a delay of between *0.1* and *0.3* seconds:

$$\begin{array}{lll} M2 & \text{sat} & s2.M \text{ from } 0 \\ & & \wedge \\ & & s2.m \text{ at } t \Rightarrow \begin{array}[t]{l} r2.m \text{ from } (t+0.1, t+0.3) \\ \wedge \\ s2.M \text{ from } (t, t+0.1) \end{array} \end{array}$$

For the purposes of this example, we have assumed that the media provide for adequate buffering of messages. This issne may be addressed separately, using timed safety specifications.

The sender process S passes each input to at least one of the transmission media within *0.2* seconds of its arrival on channel *in*:

$$\begin{array}{lll} S & \text{sat} & in.m \text{ at } t \Rightarrow \begin{array}[t]{l} s1.m \text{ from } (t, t+0.2) \\ \vee \\ s2.m \text{ from } (t, t+0.2) \end{array} \end{array}$$

If we are to guarantee the successful transmission of a specific message, we must ensure that the sender process does not flood the transmission media with spurious messages. This requirement corresponds to the the following untimed safety specification:

$$S \quad \text{sat} \quad \forall m : M \bullet \text{count}\{s1.m, s2.m\} \leqslant \text{count}\, in.m$$

For any message m, the number of transmissions of m is less than or equal to the number of times m is accepted on channel *in*.

The receiver R behaves in a complementary fashion. It is always prepared to accept a new message from either medium within 0.1 seconds. Messages are ready for output on channel *out* within 0.2 seconds of arrival.

$$\begin{array}{lll} R & \text{sat} & r1.M \cup r2.M \text{ from } 0 \\ & & \wedge \\ & & r1.m \text{ at } t \Rightarrow out.m \text{ from } (t, t+0.2) \\ & & \qquad\qquad\quad \wedge \\ & & \qquad\qquad\quad r1.M \cup r2.M \text{ from } (t, t+0.1) \\ & & \wedge \\ & & r2.m \text{ at } t \Rightarrow out.m \text{ from } (t, t+0.2) \\ & & \qquad\qquad\quad \wedge \\ & & \qquad\qquad\quad r1.M \cup r2.M \text{ from } (t, t+0.1) \end{array}$$

If a message is received on either $r1$ or $r2$ at time t, it is made available on channel *out* before time $t + 0.2$.

We assume that the sender and receiver have disjoint alphabets, and that the named channels are distinct. With these assumptions, we may implement the network as a simple parallel combination:

$$NET \;\widehat{=}\; (S \underset{\{s1,s2\}}{\|} (M1 \,|||\, M2) \underset{\{r1,r2\}}{\|} R) \setminus \{s1, s2, r1, r2\}$$

To demonstrate that this implementation meets our liveness requirement, we must show that

$$COMMS \;\; \text{sat} \;\; LIVE \setminus\!\!\setminus \{s1, s2, r1, r2\}$$

where $COMMS$ is the process

$$S \underset{\{s1,s2\}}{\|} (M1 \,|||\, M2) \underset{\{r1,r2\}}{\|} R$$

From the specifications of $M1$ and $M2$ and the inference rule for interleaving parallel combination, we may deduce that

$$\begin{array}{lll} M1 \,|||\, M2 & \text{sat} & s1.M \text{ from } 0 \;\wedge\; s2.M \text{ from } 0 \\ & & \wedge \\ & & s1.m \text{ at } t \Rightarrow r1.m \text{ from } (t+1, t+2) \\ & & \qquad\qquad\quad \wedge \\ & & \qquad\qquad\quad s1.M \text{ from } (t, t+0.1) \\ & & \wedge \\ & & s2.m \text{ at } t \Rightarrow r2.m \text{ from } (t+0.1, t+0.3) \\ & & \qquad\qquad\quad \wedge \\ & & \qquad\qquad\quad s2.M \text{ from } (t, t+0.1) \end{array}$$

Using the inference rule for communicating parallel, we may establish that:

$$\begin{array}{lll} S \underset{\{s1,s2\}}{\|} (M1 \mid\mid\mid M2) & \text{sat} & in.m \text{ at } t \Rightarrow s1.m \text{ from } (t, t+0.3) \\ & & \qquad\qquad\qquad \vee \\ & & \qquad\qquad\qquad s2.m \text{ from } (t, t+0.3) \\ & & \wedge \\ & & s1.m \text{ at } t \Rightarrow r1.m \text{ from } (t+1, t+2) \\ & & \wedge \\ & & s2.m \text{ at } t \Rightarrow r2.m \text{ from } (t+0.1, t+0.3) \end{array}$$

Observe that the maximum delay between input on channel *in* and readiness for transmission includes the *0.1* seconds that may be spent waiting for the medium to accept the message.

Another application of the inference rule yields that:

$$\begin{array}{lll} COMMS & \textbf{sat} & in.m \text{ at } t \Rightarrow s1.m \text{ from } (t, t+0.3) \\ & & \qquad\qquad\qquad \vee \\ & & \qquad\qquad\qquad s2.m \text{ from } (t, t+0.3) \\ & & \wedge \\ & & s1.m \text{ at } t \Rightarrow r1.m \text{ from } (t+1, t+2.1) \\ & & \wedge \\ & & s2.m \text{ at } t \Rightarrow r2.m \text{ from } (t+0.1, t+0.4) \\ & & \wedge \\ & & \{r1.m\} \cup \{r2.m\} \text{ at } t \Rightarrow out.m \text{ from } (t, t+0.2) \end{array}$$

Under the assumption that the set $\{s1, s2, r1, r2\}$ is hidden from the environment, we may apply lemma 6.7 and infer that:

$$\begin{array}{lll} COMMS & \textbf{sat} & (in.m \text{ at } t \Rightarrow out.m \text{ from } (t+0.1, t+0.9) \\ & & \qquad\qquad\qquad \vee \\ & & \qquad\qquad\qquad out.m \text{ from } (t+1, t+2.6)) \ \ \backslash\!\!\backslash \ \{s1, s2, r1, r2\} \end{array}$$

The disjunction corresponds to the hidden choice of media. If the message is sent via medium *M2*, there will be a delay of between *0.1* and *0.9* seconds; a different range of delays is introduced by the slower medium *M1*.

The above behavioural specification is $\{s1, s2, r1, r2\}$-independent; we may apply the inference rule for the hiding operator given in section 6.1 to obtain

$$\begin{array}{lll} NET & \text{sat} & in.m \text{ at } t \Rightarrow out.m \text{ from } (t+0.1, t+0.9) \\ & & \qquad\qquad\qquad \vee \\ & & \qquad\qquad\qquad out.m \text{ from } (t+1, t+2.6) \end{array}$$

which is enough to establish our liveness requirement:

$$NET \quad \textbf{sat} \quad in.m \text{ at } t \Rightarrow out.m \text{ from } (t, t+3)$$

In the above proof, we have omitted the details of the functional application and variable substitution necessary for the timed failures proof system. This cumbersome process can be avoided altogether if we derive inference rules for the direct manipulation of higher-order specification statements.

Chapter 7

An Ethernet-like Protocol

To illustrate the application of Timed CSP to the specification of real-time systems, we will show how the functions presented in chapter 6 may used to describe the behaviour of a communications protocol at two different levels of abstraction. Tbe protocol chosen for this purpose is based upon the Ethernet protocol defined in [Xerox 80], a standard protocol for local area networks.

The Ethernet protocol is a *broadcast* protocol: signals sent by one station may reach all of the stations upon the network. It is a *carrier-sense* protocol: stations listen for a carrier signal on the broadcast medium and act accordingly. Another important feature is *collision detection*. Each station must monitor the broadcast medium during transmission, and cease immediately if it becomes apparent that another station is also transmitting.

The Ethernet specification [Xerox 80] is divided into two parts, corresponding to the data link and physical layers of the ISO reference model described in [Tanenbaum 81]. This model consists of seven layers, each representing a different level of abstraction, from the hardware of the physical layer to the user software of the application layer. Each layer provides a service to the layer above, facilitating virtual communication between peer processes on different machines. In this chapter, we will concern ourselves with the bottom three layers of the model: the *communication subnet* of fignre 7.1.

The physical layer is the lowest layer in the model bierarchy, and transmits data as bits between the stations, or *nodes* of the network. We will provide a timed failures description of the service provided by this layer, but we will not attempt to describe its internal behaviour. Such a description would require a treatment of broadcast communication; our present model of computation is based upon synchronisation. In chapter 8, we will see how the Timed Failures model may be extended to include an element of broadcast concurrency; a description of the physical layer will be presented as an example.

The data link layer accepts *packets* of data from the layer above, inserting the data into *frames* for transmission to the physical layer. Each frame is transferred to the physical layer as a stream of bits. The data link is responsible for handling any errors which arise in frame transmission, providing its client layer with an error-free virtual communication medium. To provide this service, the data link must be capable of detecting errors, retransmitting damaged frames, and sending acknowledgments.

The *network layer* is the third layer of the ISO model. This layer converts *messages* into packets, and uses the data link to transmit them to their destination. We will refer to the network layer as the *client* layer, reserving the term *network* for the communication system as a whole.

$$\begin{tikzcd}
\textit{Client Layer} \arrow[d, leftrightarrow] \arrow[r, dashed, dash] & \textit{Client Layer} \arrow[d, leftrightarrow] \\
\textit{Data Link Layer} \arrow[d, leftrightarrow] \arrow[r, dashed, dash] & \textit{Data Link Layer} \arrow[d, leftrightarrow] \\
\textit{Physical Layer} \arrow[r, dash] & \textit{Physical Layer}
\end{tikzcd}$$

Figure 7.1: The Communication Subnet

The data link component of the Ethernet protocol does not correspond precisely to the ISO model. The data link component of Ethernet will attempt to transmit each frame no more than sixteen times; if all of these attempts are interrupted by collision detect signals from the physical layer, then the current frame is abandoned. Further, although incoming frames are checked for errors, no facility is provided for retransmission or acknowledgment. Errors (other than those caused by collisions) are simply reported to the client layer at the current node.

In this chapter, we will specify the data link component of a protocol that differs from the one described in [Xerox 80]. To simplify the presentation, we will assume that all errors are due to collisions on the broadcast medium. In this case, there is no need for error reporting at the destination node. We will also assume the existence of an implementation of the randomisation strategy employed in Ethernet, described at the end of section 7.6. With these assumptions, we may address the complex timing properties of an Ethernet-like protocol within the framework of a short case study.

7.1 A Hierarchy of Specifications

The layers of our protocol form a service hierarchy: each layer provides a service to the layer above, and makes demands upon the layer below. We will use timed failures specifications to capture the requirements at each level of hierarchy, and derive a correctness condition for any implementation of the protocol.

Specification

If a service is provided by layer L, it can be described in terms of the occurrence and availability of events from some set A_L. We will use H_L to denote the other actions performed by the layer, those hidden from the layer above. If L_i is a service hierarchy, then we require that

$$\begin{aligned} m \geqslant n + 2 &\;\Rightarrow\; \Sigma_{L_n} \cap \Sigma_{L_m} = \{\} \\ m = n + 1 &\;\Rightarrow\; \Sigma_{L_n} \cap \Sigma_{L_m} = A_{L_n} \end{aligned}$$

where Σ_L is the alphabet of layer L. Each layer should insulate the layer above from the service provided by the layer below. In the communication subnet, the data link layer DL should insulate the client layer NL from the service provided by the physical layer:

$$\begin{aligned} \Sigma_{NL} \cap \Sigma_{PL} &= \{\} \\ \Sigma_{NL} \cap \Sigma_{DL} &= A_{DL} \\ \Sigma_{DL} \cap \Sigma_{PL} &= A_{PL} \end{aligned}$$

The data link and physical layers communicate across an interface A_{PL}.

Each layer is associated with a *service* specification and a *total* specification. The first describes the service provided to the layer above, while the second describes the internal activity necessary to provide such a service. We use S_L and T_L to denote the service and total specifications of layer L, respectively. If L' is the layer below L, then the conjunction of T_L and $S_{L'}$ must be enough to ensure that the service S_L is provided:

$$(T_L \wedge S_{L'}) \setminus\!\!\setminus H_L \;\Rightarrow\; S_L$$

The use of the $\setminus\!\!\setminus$ operator in the above implication corresponds to the assumption that the events from H_L are to he concealed. We require also that S_L is H_L-independent: any service specification must be independent of hidden events.

Returning to the communication subnet, a data link implementation will satisfy a total specification T_{DL}. The service to the client layer, S_{DL}, should be a consequence of this specification, given that the physical layer provides a service S_{PL} to the data link:

$$\begin{aligned}(S_{PL} \wedge T_{DL}) \setminus\!\!\setminus H_{DL} &\Rightarrow S_{DL}\\ T_{PL} \setminus\!\!\setminus H_{PL} &\Rightarrow S_{PL}\end{aligned}$$

The service provided by the physical layer must be a consequence of its own internal activity; there is no layer beneath it.

Implementation

A protocol hierarchy may be implemented as a parallel combination of node processes, one for each node on the network. A node process will be the parallel combination of layer processes, one for each layer in the protocol:

$$\begin{aligned}PROTOCOL &\mathrel{\hat=} \|_{\Sigma_i}\ NODE_i \qquad i \in NODE\\ NODE_i &\mathrel{\hat=} \|_{\Sigma_{i,j}}\ LAYER_{i,j} \qquad j \in LAYER\end{aligned}$$

where Σ_i is the set of possible events at node i, and $\Sigma_{i,j}$ is the set of possible events on layer j at node i. The disjoint nature of the event sets allow us to rewrite the protocol as a combination of layers:

$$\begin{aligned}PROTOCOL &\mathrel{\hat=} \|_{\Sigma_j}\ LAYER_j \qquad j \in LAYER\\ LAYER_j &\mathrel{\hat=} |||\ LAYER_{i,j} \qquad i \in NODE \qquad (j \geqslant 1)\\ LAYER_0 &\mathrel{\hat=} \|_{\Sigma_{i,0}}\ LAYER_{i,0} \qquad i \in NODE\end{aligned}$$

The lowest layer of the protocol has access to a physical communication medium; all of the others have only virtual communication, corresponding to an interleaved parallel combination.

The communication subnet is implemented as follows:

$$\begin{aligned}ETHERNET &\mathrel{\hat=} DATALINK \underset{A_{PL}}{\|} PHYSICAL\\ DATALINK &\mathrel{\hat=} |||\ DL_i \qquad i \in NODE\\ PHYSICAL &\mathrel{\hat=} \|_{ALL_i}\ PL_i \qquad i \in NODE\end{aligned}$$

The data link layer is an interleaving combination of node processes; any synchronisation is by virtue of the service provided by the physical layer. The processes in the physical layer must agree upon certain events, corresponding to the presence of signals on the broadcast medium.

An implementation of a protocol hierarchy may be judged correct if

$$PROTOCOL \setminus H \quad \mathbf{sat} \quad S_L(s, \aleph)$$

where S_L is the service provided by the top layer, and H is the set of all internal events. If the protocol is implemented as a parallel combination of layers

$$PROTOCOL \;\hat{=}\; ((L_0 \underset{A_0}{\|} L_1) \underset{A_1}{\|} L_2) \underset{A_2}{\|} \ldots$$

then by the disjoint nature of our event sets and interfaces we have that

$$PROTOCOL \;\equiv\; (((L_0 \setminus H_0) \underset{A_0}{\|} L_1) \setminus H_1 \underset{A_1}{\|} L_2) \setminus H_2 \underset{A_2}{\|} \ldots$$

where H is the union of the hidden event sets H_j.

Applying the inference rules of chapter 5, it is sufficient to show that each layer j satisfies the corresponding total specification, and that an adequate service is provided at each stage:

$$\left.\begin{array}{l} S_j(s \downharpoonright \Sigma_j, \aleph_j) \\ T_{j+1}(s \downharpoonright \Sigma_{j+1}, \aleph_{j+1}) \\ \aleph \downharpoonright A_j = (\aleph_j \cup \aleph_{j+1}) \downharpoonright A_j \\ \aleph \setminus A_j = (\aleph_j \cap \aleph_{j+1}) \setminus A_j \end{array}\right\} \Rightarrow \; S_{j+1}(s, \aleph) \mathbin{\backslash\!\!\backslash} H_{j+1}$$

The total specification of layer $j + 1$, together with the service provided layer j, must be enough to provide the service S_{j+1}, given that the events from set H_{j+1} are to be hidden from the environment. The instantiation of trace and refusal sets in the left-hand side of the predicate comes from the inference rnle for the communicating parallel operator.

To establish that our example protocol is correct, we must show that

$$\begin{array}{rll} DL & \mathbf{sat} & T_{DL}(s, \aleph) \\ PL & \mathbf{sat} & T_{PL}(s, \aleph) \end{array}$$

$$\left.\begin{array}{l} S_{PL}(s \downharpoonright \Sigma_{PL}, \aleph_P) \\ T_{DL}(s \downharpoonright \Sigma_{DL}, \aleph_D) \\ \aleph \downharpoonright A_{PL} = (\aleph_P \cup \aleph_D) \downharpoonright A_{PL} \\ \aleph \setminus A_{PL} = (\aleph_P \cap \aleph_D) \setminus A_{PL} \end{array}\right\} \Rightarrow \; S_{DL}(s, \aleph) \mathbin{\backslash\!\!\backslash} H_{DL}$$

$$T_{PL}(s, \aleph) \;\Rightarrow\; S_{PL}(s, \aleph) \mathbin{\backslash\!\!\backslash} H_{PL}$$

There are no layers beneath the physical layer; the service specification S_{PL} must be a consequence of the total specification T_{DL}.

7.2 The Data Link Service

The data link layer accepts *packets* of data from the client layer at each node. It then attempts to transmit this data to all of the other nodes, via the broadcast medium of the physical layer. Each packet of data is encapsulated in a *frame* before transmission, and transmitted bit by bit to the physical layer. We assume that collisions are the only source of data corruption; if the data link succeeds in transmitting the whole frame, then the data link component of any receiving node will be able to pass the data to its client.

If the data link is interrupted by a collision during the transmission of a frame, it will *back off* and attempt to send the entire frame again. If sixteen consecutive transmissions are interrupted, the frame is discarded and the data link informs the client layer of its failure. Successful transmissions are also reported by the transmitter. Our assumption that collisions are the only source of corruption means that there is no need for error reporting at the receiver; collision-damaged frame fragments are simply discarded.

The data link is always ready to receive data from the physical layer, decapsulating whole frames and passing them to the client layer whenever the packet is addressed to the current node. The data link does not store packets: any buffering of data is the responsibility of the client layer. As a result, there is a limit on the time between the successful transmission of a frame and its delivery to the client layer at the destination node. There are also upper limits upon the time spent waiting to start a transmission, and the time spent transmitting. These are determined by the time taken to transmit one bit and the cable propagation delay, both parameters of the system.

Abstraction

We must establish which of the observable events in a possible history of the system are of interest. Our service specification will be expressed as a constraint on the occurrence and availability of these events. The data link layer is an interleaving of data link processes, one for each node on the network. The set of nodes in the network and the datatype of packets are parameters of the system.

$$\begin{array}{lllll} i & \in & NODE & & \text{nodes in the network} \\ p & \in & PKT & & \text{data packets} \\ r & \in & REP & ::= & succ \mid fail \end{array}$$

The datatype of reports has two elements, representing success and failure.

The interface between the data link layer and the client layer at node i will consist of three channels:

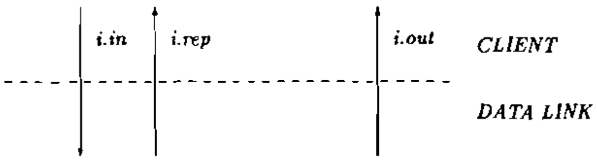

Figure 7.2: the service provided by the data link layer

Packets of data are received from the client layer at node i along channel $i.in$, and successful transmissiou is reported on channel $i.rep$. Valid frames received from the physical layer at node i will be decapsulated and passed as packets to the client layer, along channel $i.out$.

Formal Specification

The service specification of the data link will consist of constraints upon the behaviour at a single node—*local* requirements—and constraints upon the behaviour of the entire layer—*network* requirements. Each class of constraint will include both safety and liveness conditions. This classification is largely for the convenience of the reader; in section 7.3, we will employ a more systematic approacb.

The alphabet of the iuterface between the data link and client layers, across the whole network, is given by

$$A_{DL} \mathrel{\widehat{=}} \{i.in.p, i.rep.r, i.out.p \mid i : NODE\,;\, r : REP\,;\, p : PKT\}$$

The following subsets of A_{DL} will be useful in our service specification:

$$\begin{aligned} IN_i &\mathrel{\widehat{=}} \{i.in.p \mid p : PKT\} \\ REP_i &\mathrel{\widehat{=}} \{i.rep.r \mid r : REP\} \\ OUT_i &\mathrel{\widehat{=}} \{i.out.p \mid p : PKT\} \end{aligned}$$

These denote the set of all possible input events at node i, the set of possible reports at node i, and the set of possible output events at node i, respectively.

The service provided by the data link layer to the client is parametrised by the following time constants:

t_{in}	maximum delay between report and readiness for input

t_{rep}	maximum delay between input and availability of report
t_{out}	client layer maximum response time for output
t_{cs}	maximum delay in preparing a frame for transmission
t_{con}	length of the contention interval
t_{max}	maximum delivery delay for a successful transmission

The length of the contention interval, t_{con}, is an upper bound on the time taken to *acquire* the broadcast medium: if a node i transmits for time t_{con} without detecting a collision, then we can be sure that the other nodes will wait for i to finish before attempting to transmit.

We will assume the existence of a function *dest*, which returns the destination of a packet or frame. This will be a single node, although we could model broadcast packets by making the result of *dest* a set of nodes.

Local Conditions

We wish to ensure that the data link layer at each node alternates between accepting packets and reporting on their transmission. We can capture this requirement as a simple specification on the traces of the system, by counting the occurrences of events from the sets IN_i and REP_i:

$$DS1 \;\widehat{=}\; \forall\, i \bullet (\text{count}\, IN_i = \text{count}\, REP_i + 1) \vee (\text{count}\, IN_i = \text{count}\, REP_i)$$

There must be an input before each report, and there can be no more than one report following each input.

If the data link layer at node i accepts a packet at time t, it should be prepared to report the success or failure of its transmission within time t_{rep}.

$$\begin{aligned} DS2 \;\widehat{=}\; \forall\, i, t \bullet IN_i \text{ at } t \Rightarrow\; & i.rep.succ \text{ from } (t, t + t_{rep}) \\ & \vee \\ & i.rep.fail \text{ from } (t, t + t_{rep}) \end{aligned}$$

We also require that the data link should be ready for input, unless it is currently attempting to transmit a packet or waiting for a report to be collected.

$$DS3 \;\widehat{=}\; \forall\, i, t \bullet IN_i \text{ from } 0 \wedge (REP_i \text{ at } t \Rightarrow IN_i \text{ from } (t, t + t_{in}))$$

At each node i, the data link is willing initially to accept any valid packet on channel $i.in$. Subsequently, the data link becomes ready for a new packet on this channel within time t_{in} of any report on channel $i.rep$.

Network Conditions

If a success is reported at node i, then the last packet input at i will be safely delivered to its destination.

$$
\begin{aligned}
DS4 \;\; \widehat{=} \;\; & \forall\, i, j, t, t' \bullet \\
& \quad i.rep.succ \text{ at } t \;\wedge \\
& \quad dest(p) = j \;\wedge \\
& \quad \text{last } IN_i \text{ during } [0, t) = (t', i.in.p) \\
& \qquad \Rightarrow j.out.p \text{ from } (t', t' + t_{max})
\end{aligned}
$$

If the value $succ$ is passed to the client layer at node i, and the last packet input at that node was p at time t', with destination j, then that packet is made available for output at node j within time t_{max}.

Without a probabilistic argument, we can *guarantee* a successful transmission only when no other node is entrusted with a packet for the length of the contention interval. If a packet is input at node i at time t, and no other node is entrusted with a packet during the interval $(t - t_{rep}, t + t_{con} + t_{cs})$, then that packet will be delivered safely, and a success report will be available within time t_{rep}.

$$
\begin{aligned}
DS5 \;\; \widehat{=} \;\; & \forall\, i, p, t \bullet \\
& \quad i.in.p \text{ at } t \;\wedge \\
& \quad \forall\, k \bullet k \neq i \Rightarrow \neg\, (IN_k \text{ at } (t - t_{rep}, t + t_{con} + t_{cs})) \\
& \qquad \Rightarrow i.rep.succ \text{ from } (t, t + t_{rep})
\end{aligned}
$$

The bonnds of the time interval ensure that all previous packets have been dealt with, and that no other frames are ready until the period of contention is over.

Environmental Assumptions

The data link cannot provide the service specified above without the cooperation of the client layer, in the following respect. At each node, the client layer is responsible for the buffering of output packets; if the data link offers a packet to the client layer on channel $i.out$, the client will accept it within time t_{out}.

To express this environmental assumption, we define the following projection mapping on timed failures:

$$
\begin{aligned}
& \text{response } A\ (s, \aleph) \;\widehat{=} \\
& \quad inf\{ t \mid \forall\, t', I \bullet I \subseteq [0, end(s, \aleph)) \wedge (t' \in I \Rightarrow A \not\subseteq \aleph \uparrow t') \Rightarrow length(I) < t \}
\end{aligned}
$$

This function yields the least time t such that, if I is an interval contained in $[0, end(s, \aleph))$ and there is no time t' dnring I at which the whole of A is offered by the environment, then I must be shorter than t. Observe that

$$\begin{aligned} \text{response}\, A < t \;\; (s, \aleph) \;\; &\equiv \\ &\forall I \bullet I \subseteq [0, end(s, \aleph)) \wedge length(I) \geqslant t \Rightarrow \exists t' \in I \bullet A \subseteq \aleph \uparrow t' \end{aligned}$$

If the response time is less than t, then the whole of A must be offered by the environment at least once dnring any interval of length t.

Our assumption abont the client layer may be expressed as follows:

$$EA \;\; \widehat{=} \;\; \forall i \bullet \text{response}\, OUT_i < t_{out}$$

We assume that the data link at node i never has to wait longer than t_{out} for an offer of outpnt to be accepted. Onr specification of the data link service is

$$S_{DL} \;\; \widehat{=} \;\; EA \;\Rightarrow\; DS1 \wedge DS2 \wedge DS3 \wedge DS4 \wedge DS5$$

This is not a complete specification, by any means. We have shown that certain important aspects of the data link service may be rendered as timed failures specifications. In the following sections, we will see how the data link and physical layers interact to provide this service.

7.3 The Data Link Specification

The data link layer accepts packets of data from the client layer, and adds framing information. If the physical layer signals that the broadcast medium is clear, then the data link begins transmission. If the physical layer signals a collision, then the transmission is interrnpted as soon as possible. In this case, another attempt is made after a random period of time has elapsed. If sixteen attempts have been made to transmit the same frame, the data link signals that the transmission has failed, and awaits a new packet. If no collision occnrs during the transmission of a frame, then the data link signals a success.

Reception is less complicated. The data link receives bits of data from the physical layer, and stores them until the broadcast medium falls silent. When this occurs, the data stored is tested to see whether it corresponds to a valid frame intended for the cnrrent node. If this is so, then the data is stripped of its framing information and passed to the client layer. If not, then the data is discarded. In either case, the data link shonld be ready to receive new data before the inter-frame spacing time has elapsed.

Abstraction

The data link layer at node i accepts data packets along channel $i.in$, and passes the data to the physical layer along channel $i.put$ as a stream of bits. Data is collected from the physical layer at node i aloug channel $i.get$, and passed to the client layer along channel $i.out$. Reports are made available to the client layer on chanuel $i.rep$.

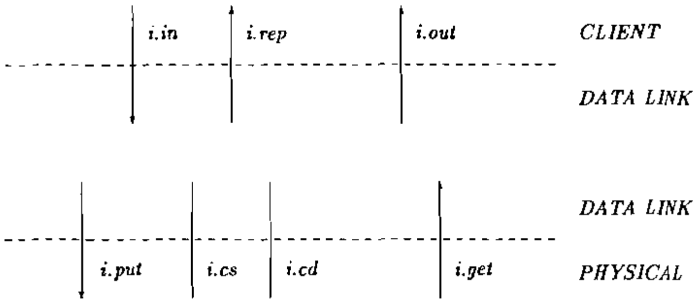

Figure 7.3: the two interfaces of the data link layer.

The physical layer is ready to synchronise upon the event $i.cs$ whenever the broadcast medium is clear at node i. Similarly, whenever a collision is detected at node i, the physical layer will make the event $i.cd$ available to the data link. If this event is observed, then the data link has been informed that a collision is takiug place. The alphabet of the data link layer is thus

$$\begin{aligned}\Sigma_{DL} \;\widehat{=}\; & \{i.in.p, i.rep.r, i.out.p \mid i : NODE \,;\, p : PKT \,;\, r : REP\} \\ & \cup \\ & \{i.put.b, i.get.b, i.cs, i.cd \mid i : NODE \,;\, b : BIT\}\end{aligned}$$

where the datatype of bits is given by

$$BIT \;::=\; 0 \mid 1$$

Any other events considered during implementation must be hidden before the data link is combined with the physical and client layers. Such events will not form a part of the data link specification.

Formal Specification

The total specification of the data link must be satisfiable by the data link layer itself, without the cooperation of the physical layer. The data link layer has only virtual communication between nodes, so our constraints must not require interaction between data link components at different nodes. All of our specifications will correspond to local requirements.

The following subsets of Σ_{DL} will be used in our specification:

$$\begin{aligned} IN_i &\mathrel{\hat=} \{i.in.p \mid p : PKT\} \\ PUT_i &\mathrel{\hat=} \{i.put.b \mid b : BIT\} \\ GET_i &\mathrel{\hat=} \{i.get.b \mid b : BIT\} \end{aligned}$$

These denote the set of possible inputs at node i, the set of possible bit transmissions at node i, and the set of possible bit receptions at node i, respectively.

We abbreviate a set of functions, each of which returns the time of occurrence of the most recent event from a particular set. If no event from that set has been observed, they will return the value *0*.

$$\begin{aligned} lastin_i(t) &\mathrel{\hat=} \text{time of last } IN_i \text{ before } t \\ lastcs_i(t) &\mathrel{\hat=} \text{time of last } i.cs \text{ before } t \\ lastcd_i(t) &\mathrel{\hat=} \text{time of last } i.cd \text{ before } t \\ lastput_i(t) &\mathrel{\hat=} \text{time of last } PUT_i \text{ before } t \\ lastget_i(t) &\mathrel{\hat=} \text{time of last } GET_i \text{ before } t \end{aligned}$$

and assume the existence of a function $frame : PKT \to \text{seq } BIT$ such that $frame(p)$ is the sequence of bits corresponding to packet p, together with the framing information required for transmission.

The following abbreviations will also appear in our specification:

$$\begin{aligned} lastpacket_i(t) &\mathrel{\hat=} \text{name of last } IN_i \text{ before } t \\ lastframe_i(t) &\mathrel{\hat=} frame(lastpacket_i(t)) \\ lasttrans_i(t) &\mathrel{\hat=} \text{data } i.put \text{ during } (lastcs_i(t), t) \\ attempts_i(t) &\mathrel{\hat=} \text{count } i.cs \text{ during } (lastin_i(t), t) \end{aligned}$$

These correspond to: the last packet input at node i; the last frame prepared for transmission at node i; the sequence of bits transmitted since the last cs signal at node i; the number of attempts made by node i to transmit the current frame.

Our specification of the data link layer will be parametrised by a number of time constants. The correct operation of the protocol will depend upon the relationships between these aud the constants defined in the previous section.

t_{succ}	maximum delay between a successful transmission and the offer of a report
t_{fail}	maximum delay between a failed transmission and the offer of a report
t_{slot}	minimum backoff delay — the *slot time* is the scheduling quantum for retransmission of a frame — $512\mu s$
t_{back}	maximum backoff delay — $524ms$
t_{bit}	time taken to trausmit one bit — $100ns$
t_{int}	inter-frame spacing delay — $9.6\mu s$
t_{rec}	maximum delay betweeu reception of a valid frame and the offer of an output

Each of these constants will be discussed in greater detail in section 7.5, where we discuss the interaction between the data link and physical layers. The values listed above are those given in the Ethernet specification document [Xerox 80].

Inputs

Our first constraint has already been presented as part of the data link service specification:

$$DT1 \;\hat{=}\; DS1$$

This stated that, at any node, inputs and reports should occur in strict alternation. The data link layer should be able to provide this part of the service without tbe assistance of the physical layer. This is true of another of our local conditions, which required that the input chaunel at any node is ready within time t_{in} of a report being collected.

$$DT2 \;\hat{=}\; DS3$$

Reports

If a success is reported at node i, then the last packet input at that node must have been transmitted successfully.

$$DT3 \;\hat{=}\; \forall\, i, t \bullet i.rep.succ \text{ at } t \Rightarrow lasttrans_i(t) = lastframe_i(t)$$

A success may be reported only if the sequence of bits transmitted since the last *i.cs* event is equal to the last frame readied for transmission at i. Conversely, if this packet has been successfully transmitted, a report should be made available:

$$\begin{aligned} DT4 \;\; \widehat{=} \;\; & \forall\, i, t \bullet PUT_i \text{ at } t \wedge lasttrans_i(t) = lastframe_i(t) \\ & \quad \Rightarrow i.rep.succ \text{ from } (t, t + t_{succ}) \end{aligned}$$

If a bit is passed along channel *i.put* at time t to complete the transmission of a frame, then a success will be reported on channel *i.rep* within time t_{succ}.

For a failure to be reported at node i, sixteen attempts must have been made to transmit the current packet.

$$DT5 \;\; \widehat{=} \;\; \forall\, i, t \bullet i.rep.fail \text{ at } t \Rightarrow attempts_i(t) = 16$$

Conversely, if the sixteenth consecutive attempt to transmit the same packet is interrupted, then the data link should report a failure:

$$\begin{aligned} DT6 \;\; \widehat{=} \;\; & \forall\, i, t \bullet i.cd \text{ at } t \wedge attempts_i(t) = 16 \\ & \quad \Rightarrow i.rep.fail \text{ from } (t, t + t_{fail}) \end{aligned}$$

If the sixteenth attempt to transmit the current frame at node i is interrupted by the observation of a collision at time t, then the data link should offer a failure report on channel *i.rep* within time t_{fail}.

Carrier-Sense

If the data link at node i requests the carrier-sense information, then either

* a packet has been received since the last report at node i, or
* a transmission has been interrupted, the minimnm backoff period has expired, and fewer than sixteen attempts have been made to transmit the current frame.

This requirement may be expressed as follows:

$$\begin{aligned} DT7 \;\; \widehat{=} \;\; \forall\, i, t \bullet i.cs \text{ at } t \Rightarrow \; & lastin_i(t) > lastput_i(t) \\ & \vee \\ & lastcd_i(t) < t - t_{slot} \wedge attempts_i(t) < 16 \end{aligned}$$

The event *i.cs* may be observed only if a packet has been input since the last bit transmission, or time t_{slot} has elapsed since the last collision was detected. Recall that, in our interpretation of the protocol, carrier-sense information is requested only as a prelude to bit transmission.

Conversely, the data link at each node should be ready to synchronise upon the *cs* event within time t_{cs} of receiving a packet for transmission, and within time t_{back} of a transmission being interrupted.

$$\begin{aligned} DT8 \;\hat{=}\; & \forall\, i, t \bullet IN_i \text{ at } t \Rightarrow i.cs \text{ from } (t, t + t_{cs}) \\ & \wedge \\ & i.cd \text{ at } t \wedge attempts_i(t) < 16 \Rightarrow i.cs \text{ from } (t + t_{slot}, t + t_{back}) \end{aligned}$$

If a packet is input at node i at time t, the event $i.cs$ should be made available by time $t + t_{cs}$. Further, if a collision is observed at node i, and fewer than sixteen attempts have been made to transmit the current frame, then $i.cs$ will be made available within time t_{back}.

Collision Detection

For a collision to be observed at node i, that node must be currently transmitting. We say that a node i is transmitting at t if a bit is broadcast at some time during the interval $(t - t_{bit}, t + t_{bit})$. This constraint is captured by the following behavioural specification:

$$DT9 \;\hat{=}\; \forall\, i, t \bullet i.cd \text{ at } t \Rightarrow lastput_i(t) > t - 2t_{bit}$$

Conversely, the data link should be ready to observe a collision at any time during frame transmission:

$$DT10 \;\hat{=}\; \forall\, i, t \bullet PUT_i \text{ at } t \Rightarrow i.cd \text{ from } t \text{ until } t + 2t_{bit}$$

Transmission

The data link should not pause during a transmission. If a bit is transmitted by node i, then either another bit was sent exactly t_{bit} ago, or the signal $i.cs$ was observed exactly time t_{int} ago. This constraint may be expressed as follows:

$$\begin{aligned} DT11 \;\hat{=}\; \forall\, i, t \bullet PUT_i \text{ at } t \Rightarrow\; & lastcs_i(t - 2t_{bit}) > lastcd_i(t - 2t_{bit}) \\ & \wedge \\ & PUT_i \text{ at } t - t_{bit} \vee i.cs \text{ at } t - t_{int} \end{aligned}$$

Note that the data link should not continue the transmission if a collision has occurred since the last *cs* event; we allow a short period of time ($2t_{bit}$) for transmission to cease. Further, we require that

$$DT12 \;\hat{=}\; \forall\, i, t \bullet lasttrans_i(t) \leqslant lastframe_i(t)$$

The sequence of data transmitted at node i must be a prefix of the last packet framed at node i.

The data link should be ready to transmit the first bit of a sequence time t_{int} after observing the *cs* event, and the subsequent bits at intervals of t_{bit}, providing that no collision is observed.

$$\begin{aligned} DT13 \;\hat{=}\; \forall\, i, t \bullet\; & i.cs \text{ at } t \Rightarrow \exists\, b \bullet i.put.b \text{ from } t + t_{int} \\ & \wedge \\ & (PUT_i \text{ at } t \;\wedge \\ & \;lastcs_i(t) > lastcd_i(t) \;\wedge \\ & \;lasttrans_i(t) \neq lastframe_i(t)) \Rightarrow \exists\, b \bullet i.put.b \text{ from } t + t_{bit} \end{aligned}$$

The data link may stop transmitting as soon as a collision is observed, or when the transmission is complete.

Reception

The data link should be ready to receive data within one bit time of the last bit arriving, unless two bit times have expired without a signal. If two bit times have elapsed since the last *get* event, then the data link receiver does not need to be ready until time t_{int} has passed.

$$\begin{aligned} DT14 \;\hat{=}\; \forall\, i, t \bullet GET_i \text{ at } t \Rightarrow\; & GET_i \text{ from } (t, t + t_{bit}) \text{ until } t + 2t_{bit} \\ & \wedge \\ & GET_i \text{ from } (t + 2t_{bit}, t + t_{int}) \end{aligned}$$

The data link becomes ready for a bit on channel $i.get$ within time t_{bit} of the last bit being received at i. The offer of GET_i may be withdrawn after two bit times have elapsed, but must be renewed before time t_{int} has elapsed; the data link must be ready before the next frame arrives.

Output

For a packet to be output at a node, it must have been received as an intact frame with the correct address. Valid frames will be preceded by an inter-frame space of duration t_{int}, and a transmission has ceased once two bit times have elapsed since the last *get* event. With these assumptions, we may identify the last frame fragment received by node i.

$$lastrec_i(t) \;\hat{=}\; \text{data } i.get \text{ during } [lastspace_i(t), lastgap_i(t)]$$

where $lastspace_i(t)$ is the endpoint of the last inter-frame space at node i:

$$lastspace_i(t) \;\hat{=}\; max\{t' \mid t' < t \wedge \neg\,(GET_i \text{ at } (t' - t_{int}, t')) \wedge GET_i \text{ at } t'\}$$

and $lastgap_i(t)$ denotes the beginning of the last gap of length $> 2t_{bit}$ observed at that node before time t.

$$lastgap_i(t) \;\widehat{=}\; max\{t' \mid t' < t - 2t_{bit} \land \neg\,(GET_i \text{ at } (t', t'+2t_{bit})) \land GET_i \text{ at } t'\}$$

This marks the end of the last contiguous bit sequence received at node i. Observe that both $lastspace_i$ and $lastgap_i$ are undefined if no data has arrived at the node i. We may complete the definitions by setting both to 0 in this case, assuming that predicate *validframe* is defined upon the empty trace of data values.

If we output a packet to the client layer, then we have received a stream of data on the *get* channel that corresponds to a valid frame with the correct address.

$$\begin{aligned} DT15 \;\widehat{=}\; \forall i, t, p \bullet i.out.p \text{ at } t \Rightarrow \exists t', f \bullet\; & t - t_{int} < t' < t \;\land \\ & lastrec_i(t') = f \;\land \\ & validframe(f) \;\land \\ & address(f) = i \;\land \\ & unframe(f) = p \end{aligned}$$

If a packet is output at time t, then the last bit of the corresponding frame must have arrived at some time during the interval $(t - 2t_{bit}, t)$. The data link layer should not buffer frames or packets.

The data link should be ready to output a valid frame within time t_{rec} of the last bit being received:

$$\begin{aligned} DT16 \;\widehat{=}\; \forall i, t, t', f, p \bullet\; & (lastrec_i(t) = f \;\land \\ & validframe(f) \;\land \\ & unframe(f) = p \;\land \\ & dest(p) = i \;\land \\ & lastgap_i(t) = t'\,) \\ & \quad \Rightarrow i.out.p \text{ from } (t', t' + t_{rec}) \text{ until } t' + t_{int} \end{aligned}$$

Note that we are assuming the existence of a suitable function for testing the validity of a frame.

We may now present the total specification of the data link layer. With no environmental assumptions to consider, it is simply the conjunction of the requirements specified above:

$$T_{DL} \;\widehat{=}\; \bigwedge_{n:1..16} DTn$$

7.4 The Physical Service

The physical layer provides a means of communication between distinct data link processes. The data link layer at a node may pass bits to the physical layer at at a rate of 10 megabits per second; the physical layer at that node will place a corresponding signal upon the broadcast medium, transmitting the data to the other nodes of the network. Signals received without interference are decoded and passed to the data link layer.

The physical layer also provides information about the state of the broadcast medium. A *carrier-sense* signal allows the data link layer to determine whether or not there is activity on the broadcast medium at the current node. Further, if a node i is transmitting bit signals, and interference is detected upon the broadcast channel, the physical layer will report that a *collision* is taking place.

Abstraction

The physical layer accepts bits from the data link layer at node i along channel $i.put$. Bit signals received from other nodes are passed to the data link along channel $i.get$. If a collision is occurring at node i, then the physical layer will make the event $i.cd$ available to the data link.

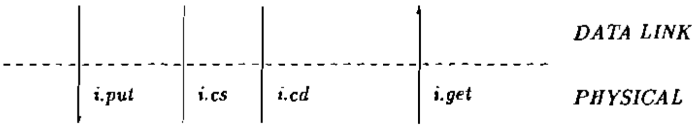

Figure 7.4: the service provided by the physical layer

The presence of a signal upon the broadcast medium at node i will be represented by the *unavailability* of the event $i.cs$. This choice of abstraction is compatible with our represeutation of the data link layer; we wish to synchronise with the physical layer when there is *no* activity upon the broadcast medium, as a prelude to data transmission.

The alphabet of the data link–physical interface is given by

$$A_{PL} \mathrel{\hat{=}} \{i.put.b, i.cs, i.cd, i.get.b \mid i : NODE ; b : BIT\}$$

Formal Specification

We present a formal description of the service provided by the physical layer, in terms of the occurrence and availability of the events in the set A_{PL}. To reason about the availability of collision detect and carrier-sense signals, we must identify the time at which the last signal arrived at a particular node. Recall that $lastput_i(t)$ denotes the time of the last signal transmitted at node i. We define

$$lastsig_i(t) \ \widehat{=}\ max\{lastput_j(t - t_{ij}) + t_{ij} \mid j : NODE\}$$

where t_{ij} is the time taken for a signal to travel from node i to node j on the broadcast medium. With this definition, $lastsig_i(t)$ is the arrival time of the last signal at node i before time t.

Transmission

The medium should be capable of receiving bits as fast as the data link layer can transmit them:

$$\begin{aligned} PS1 \ \widehat{=}\ \forall i, t \bullet\ & PUT_i \text{ from } 0 \\ & \wedge \\ & PUT_i \text{ at } t \Rightarrow PUT_i \text{ from } (t, t + t_{bit}) \end{aligned}$$

At each node i, the physical layer is prepared initially to accept a bit signal. Further, if a bit is accepted at time t, then the physical layer must be ready for another before time $t + t_{bit}$.

Carrier-Sense

We intend that the physical and data link layers should synchronise upon the event $i.cs$ only when the broadcast medium is silent at node i. The physical layer may take up to two bit times to respond to the presence of signals on the broadcast medium; if a synchronisation occurs, then the broadcast medium must have been silent less than two bit times ago.

$$\begin{aligned} PS2 \ \widehat{=}\ \forall i, t \bullet i.cs \text{ at } t \Rightarrow \exists t' \bullet\ & t' > t - 2t_{bit} \\ & \wedge \\ & lastsig_i(t') < t' - 2t_{bit} \end{aligned}$$

If an $i.cs$ synchronisation occnrs at time t, then there must be a time $t' > t - 2t_{bit}$ snch that the last signal before t' arrived before $t' - 2t_{bit}$.

The physical layer should make the event $i.cs$ available within two hit times of activity ceasing on the broadcast medium at node i. This offer should remain open at least until activity resumes.

$$\begin{aligned} PS3 \;\widehat{=}\; & \forall\, i, t, t' \bullet \mathit{lastsig}_i(t) = t' \wedge t' < t - 2t_{bit} \\ & \qquad \Rightarrow i.cs \text{ from } (t', t' + 2t_{bit}) \text{ until } t \end{aligned}$$

If the last signal to arrive at node i before time t arrives at time t', then event $i.cs$ should be made available before time $t' + 2t_{bit}$, and remain available at least until time t.

Collision Detection

A collision should be reported at node i only if a signal arrives from another node during a transmission.

$$\begin{aligned} PS4 \;\widehat{=}\; \forall\, i, t \bullet i.cd \text{ at } t \Rightarrow\; & \mathit{lastput}_i(t) > t - 2t_{bit} \\ & \wedge \\ & \mathit{lastsig}_i(t) > t - 2t_{bit} \end{aligned}$$

If a collision is reported at time t, then the interval $(t - 2t_{bit}, t)$ must contain a transmission, and the arrival of a signal from another node. Conversely, if a signal arrives from another node during transmission, then the physical layer should make $i.cd$ available within two bit times.

$$\begin{aligned} PS5 \;\widehat{=}\; & \forall\, i, t \bullet (\mathit{lastput}_i(t) > t - t_{bit} \;\wedge\; \mathit{lastsig}_i(t) > t - t_{bit}) \\ & \qquad \Rightarrow i.cd \text{ from } (\mathit{lastcd}_i(t), t + 2t_{bit}) \text{ until } t + 3t_{bit} \end{aligned}$$

If node i has transmitted a data bit less than time t_{bit} ago, and another node j transmitted approximately time t_{ij} ago, then $i.cd$ should be offered to the data link. This signal may already be available, hence the lower bound of $\mathit{lastcd}_i(t)$. Unless the collision continues, the offer may be withdrawn after one additional bit time.

Reception

In our idealised description of the physical layer, no bit should be received unless it has previously been transmitted at the appropriate time.

$$PS6 \;\widehat{=}\; \forall\, i, t \bullet i.get.b \text{ at } t \Rightarrow \exists\, j \bullet j.put.b \text{ at } t - t_{ij}$$

For data to be received without corruption, the receiving node must synchronise with the incoming sequence of bits. To facilitate this synchronisation, nodes observing the protocol described in [Xerox 80] must transmit a fixed bit sequence as a preamble to each frame. This is a responsibility of the physical layer, and will not form part of our service specification.

The corresponding liveness condition is given by:

$$\begin{aligned} PS7 \ \widehat{=}\ & \forall\, i, j, t \bullet i.put.b \text{ at } t - t_{ij} \ \wedge \\ & \qquad \forall\, k, t' \bullet (k \neq i \ \wedge \ PUT_k \text{ at } t' \ \wedge \ t' \in [t - t_{ij}, t]) \\ & \qquad\qquad \Rightarrow t' \geqslant t - t_{ij} + 2t_{bit} \ \wedge \ t_{jk} > t_{ij} \\ & \qquad \Rightarrow j.get.b \text{ from } t \text{ until } t + 0.1t_{bit} \end{aligned}$$

The reception of a bit signal b at destination node j is guaranteed if any transmission that occnrs during its journey takes place

* at least two bit times after b was transmitted

* at a node further away from j than the sender i

Under these conditions, no signal can interfere with the reception of b. Note that we have assumed that the broadcast medium is reliahle, and that signals propagate at a constant rate.

7.5 Combination

We are obliged to demonstrate that the service provided hy the data link layer is a conseqnence of that layer's total specification, together with the service provided by the physical layer, nnder the assumption that synchronisations from the set H_{DL} are concealed from the client layer. Our proof requirement is given by:

$$\left.\begin{array}{l} S_{PL}(s \downharpoonright \Sigma_{PL}, \aleph_P) \\ T_{DL}(s \downharpoonright \Sigma_{DL}, \aleph_D) \\ \aleph \downharpoonright A_{PL} = (\aleph_P \cup \aleph_D) \downharpoonright A_{PL} \\ \aleph \setminus A_{PL} = (\aleph_P \cap \aleph_D) \setminus A_{PL} \end{array}\right\} \Rightarrow \ S_{DL}(s, \aleph) \mathbin{\backslash\!\backslash} H_{DL}$$

It is sufficient to show that this result holds for each conjunct of the service specification S_{DL}; we will provide a brief justification in each case.

Each specification is given in terms of functions of timed failures, e.g. 'count', 'from', and 'response'. In these specifications, the connectives are *lifted* operators, whose actions on functions are defined by extension:

$$(\mathsf{f}\ op_\uparrow\ \mathsf{g})\ (s, \aleph) \ \widehat{=}\ \mathsf{f}\,(s, \aleph)\ op\ \mathsf{g}\,(s, \aleph)$$

where $op_\uparrow$ is the lifted form of connective op. The interpretation of connectives such as $+$, $=$, and $\wedge$ is obvious from the context; we have written $op_\uparrow$ as op.

Local Safety

Recall the first component of the data link service specification:

$$\begin{aligned} DS1\ (s,\aleph) \ \hat{=}\ & (\forall i \bullet \text{count}\, IN_i = \text{count}\, REP_i + 1 \\ & \quad\vee \\ & \quad \text{count}\, IN_i = \text{count}\, REP_i\ )\quad (s,\aleph) \end{aligned}$$

We are given that $(IN_i \cup REP_i) \subseteq \Sigma_{DL}$, and that

$$\begin{aligned} DT1\ (s \restriction \Sigma_{DL}, \aleph_D) \ \equiv\ & (\forall i \bullet \text{count}\, IN_i = \text{count}\, REP_i + 1 \\ & \quad\vee \\ & \quad \text{count}\, IN_i = \text{count}\, REP_i\ )\quad (s \restriction \Sigma_{DL}, \aleph_D) \end{aligned}$$

It is easy to see that, for any s and $\aleph$,

$$A \subseteq B \quad\Rightarrow\quad \text{count}\, A\ (s,\aleph) = \text{count}\, A\ (s \restriction B, \aleph)$$

and hence that

$$DT1\ (s \restriction \Sigma_{DL}, \aleph_D) \quad\Rightarrow\quad DS1\ (s,\aleph)$$

Input

The second requirement in the data link specification was

$$\begin{aligned} DS2 \ \hat{=}\ \forall i,t \bullet IN_i \text{ at } t \Rightarrow\ & i.rep.succ \text{ from } (t, t+t_{rep}) \\ & \vee \\ & i.rep.fail \text{ from } (t, t+t_{rep}) \end{aligned}$$

In the total specification of the data link layer, we insisted that no failure may he reported without sixteen attempts at transmission. As a result, the provision of this part of the data link service relies upon the following lemma:

Lemma 7.1 At any particular node i, the data link never has to wait more than t_{long} for a carrier-sense synchronisation $i.cs$, where t_{long} is the duration of the longest valid frame. This is *1.2ms* for the Ethernet protocol. ♡

Proof Suppose that the event $i.cs$ is not available at time t. From *PS3* we may deduce that $lastsig_i(t) > t - 2t_{bit}$, and hence that there is at least one other node j on the network already transmitting. There are two possibilities to consider:

- node j transmits without contention for time t_{con}, and *acquires* the broadcast channel, or
- node j is interrupted by another node k

If node j transmits a bit once every t_{bit} during an interval length t_{con} without interruption, then the *cs* synchronisation must be unavailable at every node on the network. This is because the length of the contention interval is greater than the round-trip signal propagation time for the network:

$$t_{con} \quad > \quad 2\ max\{t_{ij} \mid i, j : NODE\}$$

Suppose that node j has been transmitting for t_{con} at time t, with no collisions observed at j thus far. Then by *PS5* we know that $lastsig_j(t) < t - t_{con}$, and hence that

$$\forall k \quad \bullet \quad lastput_k(t - t_{jk}) < t - t_{jk} - t_{con}$$

Recall that the data link layer at each node must satisfy the requirement

$$PUT_k \text{ at } t \quad \Rightarrow \quad PUT_k \text{ at } t - t_{bit} \vee k.cs \text{ at } t - t_{int}$$

and observe that it is not possible for node k to begin transmission after $t - t_{jk}$, as $k.cs$ will not be available until time t_{jk} after j ceases transmission, by *PS2*. Hence j will not be interrupted.

In this case, *DT13* guarantees that node j will continue transmission until it has exhausted the current frame. It must then wait at least t_{int} before transmitting another bit, because of requirement *DT11*. If j transmits the last bit of the current frame at time t, then for each node k on the network we may assume that

$$\neg\ PUT_k \text{ at } (t + t_{jk} - t_{short}, t + t_{jk} + t_{int})$$

where t_{short} is the length of the shortest valid transmission; this is $57.6\mu s$ in the Ethernet protocol. From this we infer that

$$\forall k \quad \bullet \quad lastput_k(t + t_{jk} + t_{int}) < t + t_{jk} - t_{short}$$

It is an obvious property of the network that

$$t_{ij} - t_{ik} \quad \leqslant \quad t_{jk}$$

and from the definition of *lastput* we may deduce that

$$\forall k, t_1, t_2, t_3 \quad \bullet \quad t_3 < t_2 \leqslant t_1 \wedge lastput_k(t_1) < t_3 \ \Rightarrow \ lastput_k(t_2) < t_3$$

Recalling the definition of $lastsig_i$, we observe

$$lastsig_i(t + t_{ij} + t_{int}) \quad \equiv \quad max\{lastput_k(t + t_{ij} + t_{int} - t_{ik}) + t_{ik} \mid k : NODE\}$$

If we assume that $t_{short} > t_{con}$, remembering that t_{con} is more than twice t_{ij} for any i and j, then we may combine these results to yield

$$lastsig_i(t + t_{ij} + t_{int}) \leqslant t + t_{ij}$$

Given that $t_{int} > 2t_{bit}$, requirement *PS3* guarantees that the carrier-sense signal is made available at node i before $t + t_{ij} + 2t_{bit}$. Our assumption that such events are hidden from the client layer means that this signal will occur.

Now consider the other possibility: that j is interrupted within t_{con} of starting to transmit. In this case, no node will acquire the broadcast channel during the current time slot, and the event cs will be offered at node i within time t_{con}, the length of the contention interval. If j is to be interrupted, then at least one other node k must begin transmission before signals from j reach it, by *DT11* and *PS2*. Suppose that j and k start to transmit at times t_j and t_k, respectively. Observe that these nodes cause signals to arrive at i during the interval

$$[min\{t_j + t_{ij}, t_k + t_{ik}\}, max\{t_k + t_{jk} + t_{ij}, t_j + t_{jk} + t_{ik}\}]$$

Node j is interrupted by k, and thus ceases transmission, at time $t_k + t_{jk}$, and vice versa. This follows from requirements *PS5*, *DT10* and *DT11*.

If the medium at i does not fall silent at or before the end of this interval, then another node l must have started to transmit before signals from j or k could reach it. Signals from this node will cease to arrive at i before time

$$max\{t_j + t_{jl} + t_{il}, t_k + t_{kl} + t_{il}\}$$

A brief sketch of the situation should reassure the reader that l must be further from node i than j or k. An inductive argument will confirm that activity on the medium at node i must cease within time t_{con}, nnder the assumption that the network is finite, and $t_{con} > 2\ max\{t_{ij} \mid i, j : NODE\}$. □

Having justified the lemma, we may deduce that, if the data link at node i is entrusted with a packet for transmission at time t, then the cs synchronisation is available at time $t + t_{cs}$, or will be offered for at least two bit times before time $t + t_{cs} + t_{long}$.

$$\begin{aligned} \forall i, t \ \bullet \ IN_i \text{ at } t \Rightarrow \exists\, t' \bullet\ & t + t_{cs} < t' < t + t_{cs} + t_{long} \\ & \wedge \\ & i.cs \text{ from } (lastcs_i(t + t_{cs}), t') \text{ until } t' + 2t_{bit} \end{aligned}$$

Recall that t_{cs} is the maximum delay between the input of a packet and readiness for transmission, from *DT8*.

Similarly, if a transmission is interrupted and rescheduled for time t, then $i.cs$ will occur within time t_{long}, allowing another attempt to begin after the inter-frame spacing t_{int}. During the proof of the previous lemma, we established that if a node has been transmitting for time t_{con}, then it will not be interrupted. The worst case delay for status reporting may be calculated as follows:

	t_{cs}	preparing frame for transmission
$16\times$	t_{long}	waiting for the channel to clear
$16\times$	t_{int}	inter-frame spacing
$15\times$	t_{con}	almost succeeding
$15\times$	t_{back}	backing off
	t_{long}	successful transmission
	t_{succ}	delay in reporting success to data link
	$< t_{rep}$	in total, if we are to provide service S_{DL}

The response time may be reduced by increasing the backoff delay with each collision. The protocol described in [Xerox 80] insists that a node cannot delay for time t_{back} unless the current frame has been interrupted at least nine times.

Reports

The third component of the data link service specification is more easily established. Recall that this insisted that

$$\forall\, i, t \bullet IN_i \text{ from } 0 \wedge REP_i \text{ at } t \Rightarrow IN_i \text{ from } (t, t + t_{in})$$

At any node i, the data link layer must initially be ready to accept any packet on channel $i.in$. Further, should a report be accepted at time t, the data link will be ready to accept another packet before time $t + t_{in}$.

Recall that the total specification of the data link layer included precisely the same requirement:

$$DT2 \;\widehat{=}\; \forall\, i, t \bullet IN_i \text{ from } 0 \wedge REP_i \text{ at } t \Rightarrow IN_i \text{ from } (t, t + t_{in})$$

We must show that, if this requirement is true of the data link layer, then it must remain true when the data link is placed in parallel combination with the physical layer. We may assume that

$$DT2\,(s \downharpoonright \Sigma_{DL}, \aleph_D)$$
$$IN_i \cup REP_i \subseteq \Sigma_{DL}$$
$$\aleph \setminus A_{PL} = (\aleph_P \cap \aleph_D) \setminus A_{PL}$$

From the alphabet constraints, we may infer that

$$\begin{aligned}\forall i, t, a \bullet a \in IN_i \cup REP_i \Rightarrow (a \notin \sigma(\aleph_D \uparrow t) \;&\Rightarrow\; a \notin \sigma(\aleph \uparrow t)\\ &\wedge\\ \langle(t,a)\rangle \text{ in } s \;&\Leftrightarrow\; \langle(t,a)\rangle \text{ in } (s \downharpoonright \Sigma_{DL}))\end{aligned}$$

If an event from $IN_i \cup REP_i$ is offered by the data link layer, then it must be offered by the parallel combination. Further, the data link layer will perform an event from this set whenever the parallel combination does so. From the definitions of 'from' and 'at' given in chapter 6, we may infer that, for any time t

$$IN_i \text{ from } t\ (s \downharpoonright \Sigma_{DL}, \aleph_D) \;\Rightarrow\; IN_i \text{ from } t\ (s, \aleph)$$

$$REP_i \text{ at } t\ (s \downharpoonright \Sigma_{DL}, \aleph_D) \;\Rightarrow\; REP_i \text{ at } t\ (s, \aleph)$$

The result follows easily from the laws of the predicate calculus.

It will not always be necessary to expand the definitions of functions such as 'at' and 'from'. The timed failures proof system may be used to derive rules for reasoning about specifications expressed using these functions. For example, the proof of *DS3* could have made use of the following rule:

$$\frac{P \text{ sat } e \text{ from } t}{P \underset{A}{\parallel} Q \text{ sat } e \text{ from } t} \quad [\, e \notin \sigma(Q),\ A = \sigma(P) \cap \sigma(Q) \,]$$

A useful library of derived inference rules like this could be built up by pursuing further case studies in specification with Timed CSP.

Network Considerations

If a success is reported at node i, then the last packet input at i will be safely delivered to its destination.

$$\begin{aligned} DS4 \;\hat{=}\; \forall i, j, t, t' \bullet\\ & i.rep.succ \text{ at } t \;\wedge\\ & dest(p) = j \;\wedge\\ & \text{last } IN_i \text{ during } [0, t) = (t', i.in.p)\\ & \quad \Rightarrow j.out.p \text{ from } (t', t' + t_{max})\end{aligned}$$

We establish that the parallel combination of the data link and physical layers meets this requirement with the following argument.

For a success to be reported at node i, the last frame input at i must have been transmitted to the physical layer without interruption. Recall that

$$DT3 \equiv \forall i, t \bullet i.rep.succ \text{ at } t \Rightarrow lasttrans_i(t) = lastframe_i(t)$$

From requirement *PS5*, we know that any other transmissions would cause the event $i.cd$ to be offered to the data link layer. However, requirement *DT10* insists that the data link layer be ready to accept $i.cd$ whenever i is transmitting, and *DT11* insists that transmission should cease if $i.cd$ is observed.

We may conclude that no other node k transmits before $t_l - t_{ik}$, where t_l is the time at which the last bit of the frame was transmitted:

$$t_l \mathrel{\hat{=}} \text{last } PUT_i \text{ before } t$$

Assuming that the minimum frame length t_{short} is greater than the length of the contention interval t_{con}, we may infer that node i acquired the broadcast medium during the transmission of the current frame. We may apply the argument used in the proof of lemma 7.1 to establish that no other node k can begin transmission before time $t_l + t_{ik} + t_{int}$. Thus for any bit b of the frame in question,

$$\begin{aligned} i.put.b \text{ at } t \;\Rightarrow\; & \forall k, t' \bullet (k \neq i \;\wedge\; PUT_k \text{ at } t' \;\wedge\; t' \in [t - t_{ij}, t]) \\ & \qquad \Rightarrow t' \geqslant t - t_{ij} + t_{int} \;\wedge\; t_{jk} > t_{ij} \end{aligned}$$

From component *PS7* of the physical layer service specification, we know that any bit transmitted by node i is received by any node j, providing that the above condition is met:

$$\begin{aligned} PS7 \equiv\; & \forall i, j, t \bullet i.put.b \text{ at } t - t_{ij} \;\wedge \\ & \qquad \forall k, t' \bullet (k \neq i \;\wedge\; PUT_k \text{ at } t' \;\wedge\; t' \in [t - t_{ij}, t]) \\ & \qquad\qquad \Rightarrow t' \geqslant t - t_{ij} + 2t_{bit} \;\wedge\; t_{jk} > t_{ij} \\ & \qquad \Rightarrow j.get.b \text{ from } t \text{ until } t + 0.1t_{bit} \end{aligned}$$

We may conclude that each bit of the frame was received by the physical layer at node j, and passed to the data link layer along channel $j.get$.

We now appeal to the output part of the data link specification. At time $t_l + t_{ij}$, the last bit of the frame arrived at node j. We must establish that

$$lastrec_j(t_l + 3t_{bit} + t_{ij}) = lasttrans_i(t_l)$$

The last frame fragment received at node j up to and including time $t_l + t_{ij}$ should be identical to the last frame fragment transmitted at i. Recall that

$$lastrec_j(t) \mathrel{\hat{=}} \text{data } i.get \text{ during } [lastspace_j(t), lastgap_j(t)]$$

where $lastspace_j(t)$ is the endpoint of the last inter-frame space at node i:

$$lastspace_j(t) \;\hat{=}\; max\{t' \mid t' < t \wedge \neg\,(GET_j \text{ at } (t' - t_{int}, t')) \wedge GET_j \text{ at } t'\}$$

and $lastgap_j(t)$ denotes the beginning of the last gap of length $> 2t_{bit}$ observed at that node before time t.

$$lastgap_j(t) \;\hat{=}\; max\{t' \mid t' < t - 2t_{bit} \wedge \neg\,(GET_j \text{ at } (t', t' + 2t_{bit})) \wedge GET_j \text{ at } t'\}$$

From our observations above, we know that no other node k transmits during the interval $[t_0 - t_{ik}, t_1 + t_{ik} + t_{int}]$, where t_0 is the time at which node i began to transmit the frame in question:

$$t_0 \;\hat{=}\; t_{int} + \text{time of last } i.cs \text{ before } t_1$$

If another node k had transmitted after $t_0 - t_{ik}$, then a collision would have been observed at i. From the bounds of this interval, and requirements *PS6* and *PS7*, we may infer that

$$\begin{aligned} lastgap_j(t_1 + t_{ij} + 3t_{bit}) &= t_1 + t_{ij} \\ lastspace_j(t_1 + t_{ij} + 3t_{bit}) &= t_0 + t_{ij} \end{aligned}$$

providing that $t_{int} > 3t_{bit}$. We may then prove, by induction upon the length of the bit sequence transmitted at node i from time t_0 onwards, that

$$\begin{aligned} lastrec_j(t_1 + 3t_{bit} + t_{ij}) &= lasttrans_i(t_1) \\ &= lastframe_i(t_1) \\ &= frame(lastpacket_i(t_1)) \end{aligned}$$

The last contiguous bit sequence received at node j before time $t_1 + t_{ij} + 3t_{bit}$ is a valid frame containing the last packet input at node i before time t_1.

We appeal to property *DT16* of the data link layer:

$$\begin{aligned} \forall\, t, t', f, p \bullet (\, & lastrec_j(t) = f \;\wedge \\ & validframe(f) \;\wedge \\ & unframe(f) = p \;\wedge \\ & dest(p) = j \;\wedge \\ & lastgap_j(t) = t'\,) \\ & \quad \Rightarrow j.out.p \text{ from } (t', t' + t_{rec}) \text{ until } t' + t_{int} \end{aligned}$$

We know that $dest(p) = j$, and we may assume that the function *frame* always yields valid frames for transmission. Hence we have that

$$j.out.p \text{ from } (t_1 + t_{ij}, t_1 + t_{ij} + t_{rec}) \text{ until } t_1 + t_{ij} + t_{int}$$

However, our data link service requirement states that this output should remain available until it occurs. Fortunately, this service is provided subject to an environmental assumption:

$$EA \;\cong\; \forall\, i \bullet \text{response}\, OUT_i < t_{out}$$

If we assume that

$$t_{out} \;<\; t_{int} - t_{rec}$$

then this assumption allows us to infer that $j.out.p$ is observed before time $t_l + t_{ij} + t_{int}$. This is because

* we have shown that this event is available until it occurs during the interval $[t_l + t_{ij} + t_{rec}, t_l + t_{ij} + t_{int}]$, and
* we may infer from EA that the data link is seen to refuse $j.out.p$ at some time during any interval of length $> t_{out}$, corresponding to our knowledge that the event is offered by the environment.

We may conclude that

$$j.out.p \text{ from } (t_l + t_{ij}, t_l + t_{ij} + t_{rec})$$

If a packet p input at time t is to be successfully transmitted by node i, then the successful attempt at transmission may end no later than

$t\,+$		t_{cs}	preparing frame for transmission
$+$	$16\times$	t_{long}	waiting for the channel to clear
$+$	$16\times$	t_{int}	inter-frame spacing
$+$	$15\times$	t_{con}	almost succeeding
$+$	$15\times$	t_{back}	backing off
$+$		t_{long}	successful transmission of longest valid frame

This places an upper bound upon the value of t_l. We conclude that this component of the data link service is provided if

$$t_{max} \;>\; t_{cs} + 17t_{long} + t_{rec} + 15t_{con} + 16t_{int} + 15t_{back} + max\{t_{ij} \mid i, j : NODE\}$$

The maximum delay between input and the offer of output for a successful transmission will be less than t_{max}, providing that t_{max} is greater than the value of the expression to the right.

Success

The data link at node i guarantees to deliver a packet at time t, providing that no other nodes are entrusted with a packet during the interval

$$[t - t_{rep}, t + t_{con} + t_{cs}]$$

The lower bound of this interval allows us to infer that no signals arrive at node i during the packet transmission. We know that

$$\forall k \quad \bullet \quad k \neq i \Rightarrow lastin_k(t) < t - t_{rep}$$

In the proof of requirement *DS2*, we established that no node will ever wait for longer than time t_{con} for a chance to begin transmission. From requirement *DT7*, we know that the latest time that a carrier-sense synchronisation may be observed at each node k is given by

$lastin_k(t)$ +		t_{cs}	preparing frame for transmission
+	$16 \times$	t_{long}	waiting for the channel to clear
+	$16 \times$	t_{int}	inter-frame spacing
+	$15 \times$	t_{con}	almost succeeding
+	$15 \times$	t_{back}	backing off

From requirement *DT12*, we know that the last bit sequence transmitted at node k must be a subsequence of the last packet framed at that node:

$$\forall k, t \quad \bullet \quad lasttrans_k(t) \leqslant lastframe_k(t)$$

From this we may infer that each node k must cease transmission before time t_k, where t_k is given by

$$t_k \;\widehat{=}\; lastin_k(t) + t_{cs} + 15t_{con} + 16t_{int} + 15t_{back} + 17t_{long}$$

No more bits may be transmitted if the current frame is exhausted, or has been interrupted sixteen times, by *DT7*, *DT8*, and *DT11*. Assuming that t_{rep} exceeds the lower bound placed upon it earlier, we may infer that

$$lastsig_i(t + t_{con} + t_{cs}) \; < \; t$$

Appealing to the argument of lemma 7.1 yet again, we conclude that node i acquires the broadcast medium and thus succeeds in transmitting the frame. The lower bound upon the value of t_{rep} is enough to ensure that this success is reported before time $t + t_{rep}$.

This completes our semi-formal proof that the parallel combination of the data link and physical layers is enough to satisfy the data link service specification, under the following assumptions:

$$\begin{aligned}
t_{short} &> t_{con} \\
t_{con} &> 2\ max\{t_{ij} \mid i,j : NODE\} \\
t_{req} &> t_{cs} + 15t_{con} + 16t_{int} + 15t_{back} + 17t_{long} + t_{succ} \\
t_{int} &> 3t_{bit} \\
t_{out} &< t_{int} - t_{rec} \\
t_{max} &> t_{cs} + 15t_{con} + 16t_{int} + 15t_{back} + 17t_{long} + t_{rec} + max\{t_{ij} \mid i,j : NODE\}
\end{aligned}$$

Similar constraints are applied in the Ethernet specification document [Xerox 80]. For example, Appendix E of that document states that

> *It is important that data link controller implementations be able to receive a frame that arrives immediately after another frame has been transmitted or received. Here, "immediately" means 9.6μs, based on the minimum inter-frame spacing provided as recovery time for the data link. It is important that the data link controller be able to resume reception within that time.*

This particular requirement corresponds to the following component of the data link total specification:

$$\begin{aligned}
DT14 \ \hat{=}\ \forall\, i, t \bullet GET_i \text{ at } t \Rightarrow\ & GET_i \text{ from } (t, t + t_{bit}) \text{ until } t + 2t_{bit} \\
& \wedge \\
& GET_i \text{ from } (t + 2t_{bit}, t + t_{int})
\end{aligned}$$

If a bit signal is received at any node i, then the data link should offer to accept another within time t_{bit}. This offer may be withdrawn if no bit arrives within two bit times, which will be the case if a valid frame has just arrived, providing that the data link is ready to resume reception within time t_{int}.

7.6 Implementation

The Xerox specification document [Xerox 80] makes no recommendation about the implementation of the Ethernet protocol, stating that it may consist of any combination of hardware, firmware, or software. However, a concurrent variant of the language Pascal [Brinch Hansen 75] is used to describe the behaviour of the data link layer. The resulting program is presented as a definitive statement of the intended behaviour of the data link layer.

The precision of our specification language means that we have no need of an algorithmic description for specification purposes. However, such a description is useful as a guide to implementation, and as an aid to understanding the details of the timed failures specification. Accordingly, we represent the data link layer as a Timed CSP process, which must satisfy the data link total specification.

Structure

The data link layer at a single node will be implemented as a parallel combination of four processes: two sending, and two receiving. The transmit data encapsulation process inserts a packet of data into an appropriate data frame and hands it to the transmit link manager. This process connects to the physical layer at the node, receiving collision and carrier-sense signals, and sending bits for transmission.

The receiver processes complement this action: the receive link manager collects bits from the physical layer, and passes complete frames to the data decapsulation process for validation. Valid frames intended for the current node are stripped and passed to the client layer.

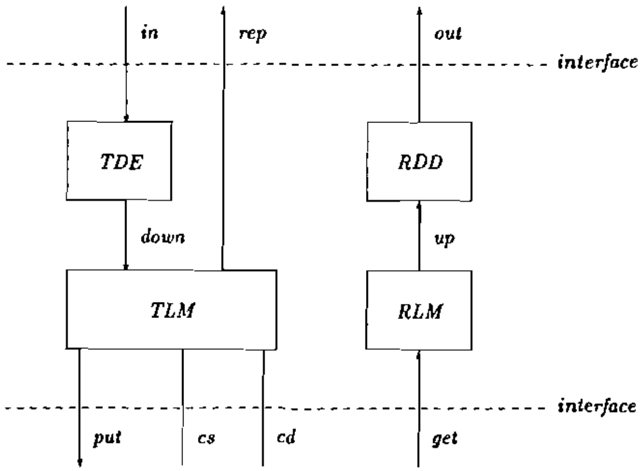

Figure 7.5: the internal structure of the data link layer.

The labelled arrows in figure 7.5 correspond to channels of communication between the processes, while the two lines labelled *cd* and *cs* represent synchronisations with the physical layer.

Implementation

The data link layer is an interleaving parallel combination of data link processes, one for each node in the network:

$$DL \;\widehat{=}\; \mathop{|||}_{i:NODE} DL_i$$

At each node, there are processes to receive and transmit data. There is no reason for the two processes to synchronise with each other in an ideal implementation.

$$DL_i \;\widehat{=}\; i : (TRANSMIT \;|||\; RECEIVE)$$

We label the processes with the appropriate *NODE* identifier. The process descriptions below are independent of the node identity.

Transmission

Data transmission is handled by two processes, connected by a single channel:

$$TRANSMIT \;\widehat{=}\; (TDE \mathop{\parallel}_{down} TLM) \setminus down$$

The data encapsulation process accepts a packet from the client layer, frames it, and passes the frame to the link management process:

$$TDE \;\widehat{=}\; in?m \xrightarrow{t_1} down!frame(m) \xrightarrow{t_2} TDE$$

Once the link manager has accepted a frame for transmission, it waits for a signal from that the medium is clear, and then passes the frame to the physical layer, bit by bit. Initially the process must listen for a frame on the appropriate channel:

$$TLM \;\widehat{=}\; down?f \xrightarrow{t_3} HOLD_{f,1}$$

The *cs* eveut is hidden from the layers above, so it will occur as soon as both layers are ready. Once *cs* has been observed, transmission may begin.

$$HOLD_{f,n} \;\widehat{=}\; cs \xrightarrow{t_{int}} (SEND_f \mathop{\nabla}_{cd} HANDLE_{f,n}) \,;\, TLM$$

Recall that the data link must wait for time t_{int}, the inter-frame spacing, before starting to transmit. The process $HOLD_{f,n}$ holds a frame f until the *cs* synchronisation occurs, and the subsequent transmission is performed by a process $SEND_f$. The second subscript to the holding process is used to record the number of attempts made to transmit the current frame.

The sending process transmits the bits of the frame at intervals of t_{bit} to the physical layer, terminating successfully if it should succeed in transmitting the entire frame.

$$\begin{aligned} SEND_{\langle\rangle} &\mathrel{\widehat{=}} rep!succ \xrightarrow{t_4} SKIP \\ SEND_{\langle x\rangle^\frown s} &\mathrel{\widehat{=}} send!x \xrightarrow{t_{bit}} SEND_s \end{aligned}$$

Before terminating, the process informs the client layer that a successful transmission has occurred. If the sending process terminates without being interrupted, or the handler terminates, then the trausmitter returns to its original state.

The transmission of a frame f may be interrupted at any time by the collision detect signal. If this occurs during the n^{th} attempt to transmit the frame, then control is passed to the process $HANDLE_{f,n}$.

$$\begin{aligned} HANDLE_{f,n} &\mathrel{\widehat{=}} BACKOFF_n \,; HOLD_{f,n+1} \quad \text{if } n < 16 \\ HANDLE_{f,16} &\mathrel{\widehat{=}} rep!fail \xrightarrow{t_5} SKIP \end{aligned}$$

If fewer than sixteen attempts have been made to transmit the current frame, then the transmitter will wait for a certain period of time before making another attempt to transmit the frame. If the sixteenth attempt is interrupted, then the transmitter informs the client layer of its failure, and terminates.

According to our data link specification, the $BACKOFF_n$ process may be implemented by any delay between t_{slot} and t_{back}. In [Xerox 80], it is implemented by a random delay process, terminating at time $r * t_{slot}$ after being started, where time r is taken from a uniform distribution of integers in the range $1 \leqslant r \leqslant 2^{max\{10,n\}}$, where n is the number of the current attempt. This allows the data link to modify its behaviour as the load upon the broadcast medium varies.

Reception

Data reception is handled by two process, also connected by a single channel:

$$RECEIVE \mathrel{\widehat{=}} (RLM \underset{up}{\parallel} RDD) \setminus up$$

The receive link manager accepts from the physical layer, and passes plausible frame fragments to the data decapsulatiou process. A fragment is plausible if its length exceeds 64 octets, the minimum frame size. The data decapsulation passes valid frames intended for the current node to the client layer along channel *out*.

The bit reception component of the link manager is prepared to accept bits from the physical layer at intervals of t_{bit}. If some bits have been received and no bits arrive for two bit times, control transferred to a simple validation process by a timeout operator:

$$\begin{aligned} RLM &\mathrel{\widehat{=}} rec?x \xrightarrow{t_{bit}} LISTEN_{\langle x\rangle} \\ LISTEN_s &\mathrel{\widehat{=}} (rec?x \xrightarrow{t_{bit}} LISTEN_{s\frown\langle x\rangle}) \overset{2t_{bit}}{\rhd} PASS_s \end{aligned}$$

The length of bit sequence s determines the behaviour of $PASS_s$.

$$PASS_s \mathrel{\widehat{=}} \text{if } \#(s) \geqslant 512 \text{ then } up!s \xrightarrow{t_6} RLM \text{ else } RLM$$

If the sequence is longer than *64* octets then it is passed to the decapsulation process via channel *up*. Shorter sequences should be discarded without further consideration.

If a bit sequence is passed to the decapsulation process, then the address field is matched against the address of the current node. We assume the existence of a function *address* that returns the appropriate information. If the bit sequence is a frame intended for the current node, then it is stripped of framing information and offered to the client layer.

$$\begin{aligned} RDD \mathrel{\widehat{=}} up?f \xrightarrow{t_7} &\text{ if } address(f) = here \\ &\text{ then } out!unframe(f) \xrightarrow{t_8} RDD \\ &\text{ else } RDD \end{aligned}$$

We have assumed that all errors are due to collisions, and our assumption that $t_{short} > t_{con}$ means that all collision-damaged frame fragments, which take less than time t_{con} to transmit, will be shorter than *64* octets. With this assumption, no error checks are required during the decapsulation process.

By applying the inference rules of the timed failures proof system, we may confirm that our implementation meets the requirements of the data link total specification, providing that

$$\begin{aligned} t_1 + t_3 &< t_{cs} & \qquad t_6 &< t_{int} \\ t_2 &< t_{rep} + t_{in} & t_7 &< t_{rec} \\ t_4 &< t_{in} & t_8 &< t_{short} - t_{out} \\ t_5 &< t_{in} \end{aligned}$$

If all the above constraints are satisfied, then we have produced a satisfactory implementation of the data link layer.

7.7 Discussion

The specification in this chapter is not a complete description of the service provided by the Ethernet protocol. We have captured some of the most important aspects of this service, and suggested a suitable implementation of the data link component of the protocol; this was sufficient to demonstrate the specification and design qualities of our notation.

We can produce a more detailed study of the protocol without changing the method of specification employed in this chapter.

* We may expand our description of the data link service by adopting a more systematic approach to the capture of requirements, as illustrated by the service specification of the physical layer. For each event a visible at the current level of abstraction, we considered the conditions under which a may occur, and the circumstances in which it must be offered to the environment. The resulting conjunction of safety and liveness constraints produced a more detailed specification.

* We may address other aspects of the data link service by adding new events to our interface. For example, to include error detection at receiving nodes, we might add a channel *err* to the datalink interface:

  $$\{i.err.e \mid i : NODE \,;\, e : ERR\} \quad \subset \quad A_{DL}$$

  where *ERR* is a datatype of error reports. Alternatively, we may choose to consider the events on the channels *in* and *out* in greater detail, specifying the format of data and addressing information in packets and frames.

The lack of a suitable model for CSP prevents us from addressing the probabilistic aspects of the Ethernet protocol within our formal specification. However, the data link implementation and the physical service provide a basis for reasoning about the performance of the protocol. For example, results such as lemma 7.1 could be used to estimate the probability of a successful frame transmission, given suitable probability distributions for the length of packets, and the frequency at which they are submitted for transmission.

Chapter 8

Signals

Wben describing the behaviour of a real-time process, we may wish to include observable events that are not synchronisations. These *signals* may make it easier to describe and analyse certain aspects of behaviour, providing useful reference points in a history of the system. For example, an audible bell might form part of the nser interface to a telephone network, even though the bell may ring without the cooperation of the user. This is incompatible with our existing view of communication.

In some cases, suitable environmental assumptions—discussed in section 4.3—will allow us to describe such behaviour within the existing Timed Failures model. However, if we intend that these signals should be used to trigger other events or behaviours, then we mnst extend our semantic model to include an element of *broadcast concurrency*: some output events may occur without the cooperation of the environment.

In our model, signal events will occnr as soon as they become available, and will propagate through parallel combination. A process may ignore any signal $\widehat{a}$ performed by another process, unless it is waiting to perform the corresponding synchronisation a. If this is the case, then both $\widehat{a}$ and a will occur. Of these, only the signal will be observed outside the parallel combination; it makes no sense to propagate a synchronisation.

We will define a denotational semantic model, representing each process as a set of possible behaviours. Each behaviour is represented by a triple $(s, \aleph, t)$, corresponding to the knowledge that the process may perform trace s while refusing synchronisations in $\aleph$, if observed up until time t. The time component is included to simplify the semantic equations for concurrency. Two component bebaviours may give rise to a behaviour of a parallel combination only if they represent observations np until the same moment in time.

8.1 The Timed Signals Model

We will represent signals as distinguished events in a extended alphabet, adopting a *hatting* convention to differentiate signals from synchronisations. If we use $\widehat{\Sigma}$ to denote the set of all signal events, then the set of all events is given by

$$\widetilde{\Sigma} \mathrel{\widehat{=}} \Sigma \cup \widehat{\Sigma}$$

For each synchronisation event a in Σ, we may add a signal event $\widehat{a}$.

We use $T\widetilde{\Sigma}$ to denote the set of all timed synchronisations and signals, and $T\widetilde{\Sigma}^*_\leqslant$ to denote the set of all timed traces that may include signals:

$$\begin{aligned} T\widetilde{\Sigma} &\mathrel{\widehat{=}} TIME \times \widetilde{\Sigma} \\ T\widetilde{\Sigma}^*_\leqslant &\mathrel{\widehat{=}} \{s \in \operatorname{seq} T\widetilde{\Sigma} \mid (t, e) \text{ precedes } (t', e') \text{ in } s \Rightarrow t \leqslant t'\} \end{aligned}$$

Signal events may not be refused if offered, so there is no reason to include them in timed refusal sets:

$$\begin{aligned} TINT &\mathrel{\widehat{=}} \{[b, e) \mid 0 \leqslant b < e < \infty\} \\ RTOK &\mathrel{\widehat{=}} \{I \times A \mid I \in TINT \wedge A \in \mathbf{P}\,\Sigma\} \\ RSET &\mathrel{\widehat{=}} \{\bigcup C \mid C \in \mathbf{F}\,RTOK\} \end{aligned}$$

The set of possible refusal sets in the Timed Signals model is given by $RSET$, as before. The set of possible observations in this model is given by $T\widetilde{F}$, where

$$T\widetilde{F} \mathrel{\widehat{=}} T\widetilde{\Sigma}^*_\leqslant \times RSET \times TIME$$

Each possible behaviour is a triple, consisting of a timed trace from $T\widetilde{\Sigma}^*_\leqslant$, a timed refusal set, and a time value.

We will give a new semantics to our language of Timed CSP terms, mapping each construct to an element of $TS_{\widetilde{F}}$, where

$$TS_{\widetilde{F}} \mathrel{\widehat{=}} \mathbf{P}\,T\widetilde{F}$$

As before, we employ a domain of environments to record the values of term variables, and define a semantic function for terms:

$$\begin{aligned} \widetilde{ENV} &\mathrel{\widehat{=}} VAR \rightarrow TS_{\widetilde{F}} \\ \mathcal{F}_S &\in TCSP \rightarrow \widetilde{ENV} \rightarrow TS_{\widetilde{F}} \end{aligned}$$

We write $\mathcal{F}_S[\![P]\!]\rho$ to denote the semantics of term P in an environment ρ. As in the Timed Failures model, we omit the environment parameter when we give the semantics of a closed term.

The Timed Signals model $TM_{\tilde{F}}$ is defined to be those elements S of $TS_{\tilde{F}}$ which satisfy a set of eight healthiness conditions, enshrined as axioms of the model:

1. $(s, \aleph, t) \in S \Rightarrow t \geqslant end(s, \aleph)$

2. $(s, \aleph, t) \in S \wedge t' \geqslant t \Rightarrow \exists\, s' \bullet \sigma(s') \subseteq \hat{\Sigma} \wedge (s^\frown(s' + t), \aleph, t') \in S$

3. $(\langle\rangle, \{\}, 0) \in S$

4. $(s^\frown w, \aleph, t) \in S \wedge end(s) \leqslant t' \leqslant min\{t, begin(w)\} \Rightarrow (s, \aleph \upharpoonright t', t') \in S$

5. $(s, \aleph, t) \in S \wedge s \cong w \Rightarrow (w, \aleph, t) \in S$

6. $(s, \aleph, t) \in S \Rightarrow \exists\, \aleph' \in RSET \bullet \aleph \subseteq \aleph' \wedge (s, \aleph', t) \in S \wedge$
   $\forall\, t' : TIME\, ;\, a : \Sigma \bullet (t' \leqslant t \wedge (t', a) \notin \aleph')$
   $\Rightarrow (s \upharpoonright t'^\frown\langle(t', a)\rangle, \aleph' \upharpoonright t', t') \in S$

7. $\forall\, t : TIME \bullet \exists\, n(t) \in \mathbb{N} \bullet (s, \aleph, t) \in S \Rightarrow \#(s) \leqslant n(t)$

8. $(s, \aleph, t) \in S \wedge \aleph' \in RSET \wedge \aleph' \subseteq \aleph \Rightarrow (s, \aleph', t) \in S$

The first axiom insists that no trace or refusal information is recorded after the end of the current observation. The second states that any observation can be extended into the future; the only events that *must* be observed are signals. The remaining six conditions are inherited from the underlying Timed Failures model, modified slightly to reflect the possible presence of signal events in a process trace.

The third axiom allows us to infer that all processes have at least one possible behaviour: the empty failure, observed until time 0. The fourth axiom states that any behavionr of S gives rise to another if truncated, while the fifth states that the set of traces of a process should be closed under timed trace equivalence.

The sixth axiom is a finitary condition upon refusal information. For any observation $(s, \aleph, t)$, there exists a maximal refusal set $\aleph'$ such that any timed synchronisation (t', a) not in $\aleph'$ is a possible extension of $s \upharpoonright t'$. The seventh axiom places a similar coudition upon traces, asserting the existence of an upper bound $n(t)$ npon the number of signals or synchronisations that may be observed before time t in any behaviour of S. The remaining axiom states that if a process may refuse a set $\aleph$, then it may refuse any subset of $\aleph$.

For any $S \in TS_{\tilde{F}}$ and $t \in TIME$, we define

$$S(t) \;\widehat{=}\; \{(s, \aleph, t') \in S \mid t' \leqslant t\}$$

This yields the set of observations from S that end strictly before time t, and suggests a distance metric on the space $TS_{\widetilde{F}}$

$$\tilde{d}(S,T) \;\widehat{=}\; inf\{\{2^{-t} \mid S(t) = T(t)\} \cup \{1\}\}$$

In section A.3, we show that our model is a *complete* metric space under $\tilde{d}$.

Notation

We define a new alphabet operator for timed traces, to match the synchronisation set operator defined in chapter 2.

$$\hat{\sigma}(s) \;\widehat{=}\; \{\hat{a} \in \hat{\Sigma} \mid \exists\, t \bullet \langle(t, \hat{a})\rangle \textbf{ in } s\}$$

As before, we overload the definition of this operator,

$$\hat{\sigma}(P) \;\widehat{=}\; \{\hat{a} \in \hat{\Sigma} \mid \exists (s, \aleph, t) \in \mathcal{F}_S[\![P]\!] \bullet \hat{a} \in \hat{\sigma}(s)\}$$

to return the set of signal events that may be performed by a process P.

We define an operator *sync*, which may be applied to a trace or set of timed events. For any $s \in T\tilde{\Sigma}^*_{\leqslant}$ or $A \subseteq T\tilde{\Sigma}$,

$$\begin{aligned}
sync(\langle\rangle) &\;\widehat{=}\; \langle\rangle \\
sync(\langle(t,a)\rangle^\frown s) &\;\widehat{=}\; \langle(t,a)\rangle^\frown sync(s) \\
sync(\langle(t,\hat{a})\rangle^\frown s) &\;\widehat{=}\; \langle(t,a)\rangle^\frown sync(s) \\
sync(A) &\;\widehat{=}\; \{a \in \Sigma \mid a \in A \vee \hat{a} \in A\}
\end{aligned}$$

This operator returns the set of synchronisation events that are mentioned in the set or trace, as synchronisations or signals.

The semantics of parallel combination in the Timed Signals model will require a new subsequence relation between timed traces:

$$s_1 \subseteq s_2 \;\Leftrightarrow\; \forall\, t, a \bullet \langle(t,a)\rangle \textbf{ in } s_1 \Rightarrow \langle(t,a)\rangle \textbf{ in } s_2$$

We say that a trace s_1 is a subset of trace s_2 if and only if each timed event in s_1 is also present in s_2.

The failure subtraction operator may be applied to behaviours in this model:

$$\begin{aligned}
(s,\aleph,t) - t' \;\widehat{=}\;& (s \dot{-} t', \aleph \dot{-} t', t - t') && \text{if } t \geqslant t' \\
& (\langle\rangle, \{\}, 0) && \text{otherwise}
\end{aligned}$$

Subtracting time t from a behaviour discards the part of the behaviour that lies before time t; the remaining part is shifted backwards through time.

8.2 Sequential Processes

Atoms

As in the Timed Failures model, the divergent process $\perp$ is identified with the deadlocked process *STOP*. Any trace of this process must be empty,

$$\mathcal{F}_S[\![STOP]\!]\rho \;\widehat{=}\; \{(s,\aleph,t) \mid s = \langle\rangle \wedge t \geqslant end(\aleph)\}$$

and any refusals must be recorded before the observation ends.

The synchronising termination process *SKIP* is ready to perform a single instance of the special event ✓ at any time.

$$\begin{aligned}\mathcal{F}_S[\![SKIP]\!]\rho \;\widehat{=}\;& \{(\langle\rangle,\aleph,t) \mid \checkmark \notin \sigma(\aleph) \wedge t \geqslant end(\aleph)\}\\ & \cup\\ & \{(\langle(t,\checkmark)\rangle,\aleph,t') \mid \checkmark \notin \sigma(\aleph \upharpoonright t) \wedge t' \geqslant max\{t, end(\aleph)\}\}\end{aligned}$$

If no events have been observed, then ✓ is available, and any refusals were recorded before the end of the observation. Otherwise, ✓ is observed at some time t and was available beforehand.

We may wish to use a signal to indicate that the successful termination of a process. Such an event would be propagated to the environment, causing termination in any process that is waiting to synchronise upon ✓. We define

$$\mathcal{F}_S[\![\widehat{SKIP}]\!]\rho \;\widehat{=}\; \{(\langle\rangle,\{\},0)\} \cup \{(\langle(0,\hat{\checkmark})\rangle,\aleph,t) \mid\wedge t \geqslant end(\aleph)\}$$

If no events have been observed, then we have watched only until time *0*. If our observation extends beyond this time, then a termination signal will be observed. We also define two forms of delayed termination:

$$\begin{aligned}\mathcal{F}_S[\![WAIT\ t]\!]\rho \;\widehat{=}\;& \{(\langle\rangle,\aleph,t') \mid \checkmark \notin \aleph \upharpoonleft t \wedge t' \geqslant end(\aleph)\}\\ & \cup\\ & \{(\langle(t'',\checkmark)\rangle,\aleph,t') \mid t'' \geqslant t \;\wedge t' \geqslant max\{t'', end(\aleph)\}\\ &\qquad\qquad\qquad\qquad \wedge \checkmark \notin \aleph \uparrow [t,t'')\}\end{aligned}$$

If the process is to synchronise upon the termination event, then ✓ is made available from time t onwards. If the process is to signal termination, then the termination signal will be observed at time t.

$$\begin{aligned}\mathcal{F}_S[\![\widehat{WAIT}\ t]\!]\rho \;\widehat{=}\;& \{(\langle\rangle,\aleph,t') \mid end(\aleph) \leqslant t' \leqslant t\}\\ & \cup\\ & \{(\langle(t,\hat{\checkmark})\rangle,\aleph,t') \mid t' \geqslant max\{t, end(\aleph)\}\}\end{aligned}$$

In either case, any event may be refused before time t.

Prefix

The event prefix operator transfers control to a process following the observation of an event. If this event is a synchronisation, then it should be continuously available until it occurs:

$$\begin{array}{rcll}
\mathcal{F}s[\![a \to P]\!]\rho & \widehat{=} & \{(\langle\rangle, \aleph, t) \mid a \notin \sigma(\aleph) \wedge t \geqslant end(\aleph)\} & \\
& & \cup & \\
& & \{(\langle(t,a)\rangle^\frown s, \aleph, t') \mid & a \notin \sigma(\aleph \restriction t) \wedge \\
& & & begin(s) \geqslant t + \delta \wedge \\
& & & t' \geqslant max\{t, end(s, \aleph)\} \wedge \\
& & & (s, \aleph, t') - (t+\delta) \in \mathcal{F}s[\![P]\!]\rho\}
\end{array}$$

If the event is a signal, then it should occur immediately.

$$\begin{array}{rcll}
\mathcal{F}s[\![\hat{a} \to P]\!]\rho & \widehat{=} & \{(\langle\rangle, \{\}, 0)\} & \\
& & \cup & \\
& & \{(\langle(0,\hat{a})\rangle^\frown s, \aleph, t) \mid & t \geqslant end(s,\aleph) \wedge \\
& & & begin(s) \geqslant \delta \wedge \\
& & & (s, \aleph, t) - \delta \in \mathcal{F}s[\![P]\!]\rho\}
\end{array}$$

A delay of time δ is associated with the transfer of control to P.

Sequential Composition

In the sequential composition $P\,;\,Q$ control passes from P to Q as soon as P offers to synchronise upon the termination event $\checkmark$, or sends a termination signal $\hat{\checkmark}$. In either case, there is no delay associated with the transfer of control, and the termination event is concealed from the environment.

$$\begin{array}{rcl}
\mathcal{F}s[\![P\,;Q]\!]\rho & \widehat{=} & \{(s,\aleph,t) \mid \checkmark \notin \sigma(s) \wedge \hat{\checkmark} \notin \hat{\sigma}(s) \\
& & \qquad \wedge\, (s, \aleph \cup ([0,t) \times \{\checkmark\}), t) \in \mathcal{F}s[\![P]\!]\rho\} \\
& & \cup \\
& & CL_{\cong}\{(s^\frown w, \aleph, t) \mid \\
& & \quad \checkmark \notin \sigma(s) \wedge \hat{\checkmark} \notin \hat{\sigma}(s) \wedge (w, \aleph, t) - t' \in \mathcal{F}s[\![Q]\!]\rho \\
& & \quad \wedge \\
& & \quad (\,(s^\frown\langle(t',\checkmark)\rangle, \aleph \restriction t' \cup ([0,t') \times \{\checkmark\}), t') \in \mathcal{F}s[\![P]\!]\rho \\
& & \quad \vee \\
& & \quad (s^\frown\langle(t',\hat{\checkmark})\rangle, \aleph \restriction t' \cup ([0,t') \times \{\checkmark\}), t') \in \mathcal{F}s[\![P]\!]\rho)\}
\end{array}$$

Any observation of this seqnential composition may be an observation of P in which no termination events occnr, or a terminating observation of P followed by an observation of Q.

The sequential composition operator does not distinguish between the two forms of the termination event. This is illustrated by the following equivalence:

$$(P \mid\mid\mid WAIT\ t) ; Q \;\; \equiv \;\; (P \mid\mid\mid \widehat{WAIT}\ t) ; Q$$

In the presence of the sequential composition operator, the termination event ✓ occurs as soon as it is made available, and is concealed from the environment.

Choice

The semantics of the nondeterministic choice operator is the usual union of possible component behaviours:

$$\mathcal{F}_S[\![P \sqcap Q]\!]\rho \;\; \widehat{=} \;\; \mathcal{F}_S[\![P]\!]\rho \cup \mathcal{F}_S[\![Q]\!]\rho$$

and can be extended to give a meaning to indexed nondeterministic choice provided that the set of alternatives is *uniformly bounded*, in the sense of section 2.4.

If either of the components of a deterministic choice is ready to perform a signal event, then that choice will be resolved immediately.

$$\begin{aligned}\mathcal{F}_S[\![P \,\Box\, Q]\!]\rho \;\; \widehat{=} \;\; & \{(\langle\rangle, \aleph, t) \mid (\langle\rangle, \aleph, t) \in \mathcal{F}_S[\![P]\!]\rho \cap \mathcal{F}_S[\![Q]\!]\rho\} \\ & \cup \\ & \{(s, \aleph, t) \mid s \neq \langle\rangle \wedge (s, \aleph, t) \in \mathcal{F}_S[\![P]\!]\rho \cup \mathcal{F}_S[\![Q]\!]\rho \\ & \qquad\qquad \wedge \\ & \qquad\qquad (\langle\rangle, \aleph \upharpoonright begin(s), begin(s)) \in \mathcal{F}_S[\![P]\!]\rho \cap \mathcal{F}_S[\![Q]\!]\rho\}\end{aligned}$$

Any event refused by a deterministic choice before any events have been observed must be refused by both components, and the behaviour following the first event must stem from just one of the components.

As an example, consider a process which is initially prepared to participate in the synchronisation a,

$$(a \rightarrow SKIP) \;\; \Box \;\; (WAIT\ t ; \widehat{b} \rightarrow STOP)$$

If this event occurs, the process terminates successfully. However, if a has not been observed by time t, the process sends a signal $\widehat{b}$ and then deadlocks. The occurrence of the signal event resolves the choice, and withdraws the offer of a.

The prefix choice operator may be used to offer the environment an infinite choice of inputs to a process. Signals correspond to output events; there is no reason to include signals in a prefix choice construct.

$$\begin{aligned}\mathcal{F}_S[\![a : A \to P_a]\!]\rho \;\hat{=}\; & \{(\langle\rangle, \aleph, t) \mid A \cap \sigma(\aleph) = \{\}\} \\ & \cup \\ & \{(\langle(t', a)\rangle^\frown s, \aleph, t) \mid a \in A \wedge t' \geqslant 0 \;\wedge \\ & \qquad\qquad A \cap \sigma(\aleph \restriction t') = \{\} \;\wedge \\ & \qquad\qquad (s, \aleph, t) - (t' + \delta) \in \mathcal{F}_S[\![P_a]\!]\rho\}\end{aligned}$$

We assume that the set A contains only synchronisation events, and that the set of alternatives $\{P_a \mid a \in A\}$ is uniformly bounded.

Relabelling

The relabelling functions may be used to rename the events in a process, while preserving aspects of the control structure. We do not permit the use of such functions to transform signals into synchronisations, or vice versa. We insist that for any relabelling function f.

$$\begin{aligned}\forall a : \Sigma \;\bullet\; f(a) \in \Sigma \\ \forall \hat{a} : \hat{\Sigma} \;\bullet\; f(\hat{a}) \in \hat{\Sigma}\end{aligned}$$

With this restriction, the effect of applying such a function is given by

$$\begin{aligned}\mathcal{F}_S[\![f^{-1}(P)]\!]\rho \;&\hat{=}\; \{(s, \aleph, t) \mid (f(s), f(\aleph), t) \in \mathcal{F}_S[\![P]\!]\rho\} \\ \mathcal{F}_S[\![f(P)]\!]\rho \;&\hat{=}\; \{(f(s), \aleph, t) \mid (s, f^{-1}(\aleph), t) \in \mathcal{F}_S[\![P]\!]\rho\}\end{aligned}$$

where $f^{-1}(\aleph)$ denotes the inverse image of refusal set $\aleph$ under f.

Abstraction

We may conceal signal events from the environment by removing them from the trace. Because the process cannot stop signals occurring, the internal behaviour is independent of any signals performed. We extend the definition of the hiding operator on traces:

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

We may hide any combination of signals and synchronisations from the environment of a process:

$$\mathcal{F}_S[\![P \setminus A]\!]\rho \;\widehat{=}\; \{(s \setminus A, \aleph, t) \mid (s, \aleph \cup ([0,t) \times (A \downharpoonright \Sigma)), t) \in \mathcal{F}_S[\![P]\!]\rho\}$$

Observe that only synchronisations may be added to the refusal set; signal events already occur as soon as they become available.

Recursion

As before, we may regard the semantics of a term P with free variable X as a function defined upon $TS_{\widetilde{F}}$, mapping a set of behaviours S to the semantics of P in an environment obtained by associating X with S.

Definition 8.1 If P is a *TCSP* term, and X and Y are variables such that Y does not occur free in P, then

$$\begin{aligned} \widetilde{M}(X,P)\rho &\;\widehat{=}\; \lambda\, Y \bullet \mathcal{F}_S[\![P]\!]\rho[Y/X] \\ \widetilde{M}_\delta(X,P)\rho &\;\widehat{=}\; \widetilde{W}_\delta \cdot \lambda\, Y \bullet \mathcal{F}_S[\![P]\!]\rho[Y/X] \end{aligned}$$

where $\widetilde{W}_\delta$ is the mapping defined by

$$\widetilde{W}_\delta \;\widehat{=}\; \lambda\, Y \bullet \mathcal{F}_S[\![\mathit{WAIT}\,\delta\,;X]\!]\rho[Y/X]$$

◇

The definition of $\widetilde{W}_\delta$ reflects the delay associated with the second form of recursion operator. The semantics of each recursion operator is given by the fixed point of the corresponding mapping:

$$\begin{aligned} \mathcal{F}_S[\![\mu\, X \circ P]\!]\rho &\;\widehat{=}\; \text{the unique fixed point of the mapping } \widetilde{M}(X,P)\rho \\ \mathcal{F}_S[\![\mu\, X \bullet P]\!]\rho &\;\widehat{=}\; \text{the unique fixed point of the mapping } \widetilde{M}_\delta(X,P)\rho \end{aligned}$$

The signals model $TM_{\widetilde{F}}$ is a complete metric space under the metric $\widetilde{d}$ defined earlier in this chapter. Following the arguments of chapter 3, we can show that the semantics of delayed recursion is always well-defined.

Further, the addition of signals to our computational model does not affect the notion of a constructive term. As in section 3.2, we may establish that the semantics of the immediate recursion $\mu\, X \bullet P$ is well-defined whenever term P is constructive for variable X.

8.3 Signals and Concurrency

We intend that signals should be propagated through a parallel combination, and that available synchronisations are *triggered* by the corresponding signal events: if a signal $\widehat{a}$ is observed, then any process waiting to perform synchronisation a will be allowed to proceed. Observable synchronisations require the participation of the environment, so if a signal forms part of the interface between two processes then the corresponding synchronisation must be concealed.

It would be enough to conceal only those synchronisations which occur at the same time as the corresponding signal events, allowing a process to signal and synchronise upon the same event. For example, we might define a process

$$a \rightarrow STOP \overset{t}{\rhd} \widehat{a} \rightarrow STOP$$

which waits to synchronise upon event a, but will send a signal $\widehat{a}$ instead if no progress has been made by time t. However, it can be argued that no process may signal and synchronise upon the same event; we obtain a simpler, more intuitive, semantics for concurrency if we proscribe dynamic reconfiguration of input and output channels.

Accordingly, we place a simple restriction upon the sets presented as arguments to the alphabet parallel operator. In the parallel combination

$$P \;{}_{A}\|_{B}\; Q$$

the sets A and B determine which synchronisations may be performed by processes P and Q, respectively. By adding signals to these sets, we may also determine which signals are propagated. We will insist that

$$\begin{aligned} A \cap sync(A \cap \widehat{\Sigma}) &= \{\} \\ B \cap sync(B \cap \widehat{\Sigma}) &= \{\} \end{aligned}$$

No event a may appear in the same set as a synchronisation a and a signal $\widehat{a}$.

As an example, consider the following choices for A and B:

$$\begin{aligned} A &\mathrel{\widehat{=}} \{a, b, \widehat{c}\} \\ B &\mathrel{\widehat{=}} \{\hat{a}, b, \widehat{c}\} \end{aligned}$$

In this case, either component may broadcast signal $\widehat{c}$ to the environment, and Q may broadcast $\widehat{a}$. If Q broadcasts $\widehat{a}$, then P may perform synchronisation a, but only the signal will be propagated to the environment. As before, both components must cooperate upon any synchronisation in $A \cap B$.

We may now derive the semantic equation for $P\ {}_A\|_B\ Q$. Suppose that the traces performed by components P and Q are s_P and s_Q respectively. Any synchronisation common to both sets must be performed by botb components:

$$s \upharpoonright \Sigma \upharpoonright (A \cap B) \;=\; s_P \upharpoonright \Sigma \upharpoonright (A \cap B) \;=\; s_Q \upharpoonright \Sigma \upharpoonright (A \cap B)$$

and any synchronisation that is exclusive to one component will be observed if it is performed by that component, and not hidden by a corresponding signal. If we identify the sets of signals

$$\begin{aligned} C &\mathrel{\widehat{=}} A \cap \widehat{\Sigma} \\ D &\mathrel{\widehat{=}} B \cap \widehat{\Sigma} \end{aligned}$$

then we may capture this requirement as

$$\begin{aligned} s \upharpoonright \Sigma \upharpoonright (A - B) &= s_P \upharpoonright (\Sigma - sync(D)) \\ s \upharpoonright \Sigma \upharpoonright (B - A) &= s_Q \upharpoonright (\Sigma - sync(C)) \end{aligned}$$

Synchronisations of P tbat are also signals of Q are removed from the trace, and can only occur when Q performs the corresponding signal:

$$\begin{aligned} s_P \upharpoonright (A \cap sync(D)) &\subseteq sync(s_Q \upharpoonright D) \\ s_Q \upharpoonright (B \cap sync(C)) &\subseteq sync(s_P \upharpoonright C) \end{aligned}$$

If a is such a synchronisation, then (t, a) may appear in s_P only if $(t, \widehat{a})$ appears in s_Q. This will be true whenever (t, a) appears in the trace $sync(s_Q \upharpoonright D)$. A similar condition applies for synchronisations of Q.

The parallel combination will propagate any signals that lie in A or B and are performed by the corresponding process:

$$s \upharpoonright \widehat{\Sigma} \;\in\; s_P \upharpoonright C \mathbin{|||} s_Q \upharpoonright D$$

Each component may perform signals from outside these sets, but they will not be passed to the other components, nor to the environment of the parallel combination. Combining these conditions, we obtain that

$$\begin{aligned} s \in s_P\ {}_A\widetilde{\|}_B\ s_Q \;\Leftrightarrow\; & s \upharpoonright \Sigma \upharpoonright (A \cap B) = s_P \upharpoonright \Sigma \upharpoonright (A \cap B) = s_Q \upharpoonright \Sigma \upharpoonright (A \cap B) \;\wedge \\ & s \upharpoonright \Sigma \upharpoonright (A - B) = s_P \upharpoonright (\Sigma - sync(D)) \;\wedge \\ & s \upharpoonright \Sigma \upharpoonright (B - A) = s_Q \upharpoonright (\Sigma - sync(C)) \;\wedge \\ & s_P \upharpoonright (A \cap sync(D)) \subseteq sync(s_Q) \;\wedge \\ & s_Q \upharpoonright (B \cap sync(C)) \subseteq sync(s_P) \;\wedge \\ & s \upharpoonright \widehat{\Sigma} \in s_P \upharpoonright C \mathbin{|||} s_Q \upharpoonright D \;\wedge\; s \upharpoonright \Sigma = s \upharpoonright \Sigma \upharpoonright (A \cup B) \end{aligned}$$

If P performs a signal $\hat{a}$ at time t, then the corresponding synchronisation should be offered to component Q. We may examine the effect of such an offer by including (t, a) in the appropriate refusal set. If s_P and s_Q are the traces performed by components P and Q, then we insist that for any time t'

$$\begin{aligned} sync(\sigma(s_Q \restriction D \uparrow t')) &\subseteq \sigma(\aleph'_P \restriction A \uparrow t') \\ sync(\sigma(s_P \restriction C \uparrow t')) &\subseteq \sigma(\aleph'_Q \restriction B \uparrow t') \end{aligned}$$

The synchronisations corresponding to the signals performed by Q at time t must be offered to component P at time t, if they are contained in set A. A similar condition applies to signals performed by P.

The behaviours of each component will take the following form:

$$\begin{aligned} (s_P, \aleph_P \cup \aleph'_P, t) &\in \mathcal{F}_S[\![P]\!]\rho \\ (s_Q, \aleph_Q \cup \aleph'_Q, t) &\in \mathcal{F}_S[\![Q]\!]\rho \end{aligned}$$

Any synchronisation from A that is refnsed by component P must be refused by the parallel combination. A similar condition applies to events from B.

$$\begin{aligned} \aleph_P &\subseteq \aleph \restriction A \\ \aleph_Q &\subseteq \aleph \restriction B \end{aligned}$$

We consider only those refusals of P and Q which correspond to events from A and B: any other synchronisation will be impossible. Conversely, any event refused from $A \cup B$ must be refused by at least one of the components, or concealed by the inclusion of the corresponding signal in the interface set.

$$\aleph \restriction (A \cup B) \setminus sync(C \cup D)) = (\aleph_P \setminus sync(D)) \cup (\aleph_Q \setminus sync(C))$$

For convenience, we define

$$\begin{aligned} \aleph \in \aleph_P \;{}_A\tilde{\|}_B\; \aleph_Q \Leftrightarrow \; & \aleph_P \subseteq \aleph \restriction A \;\wedge\; \aleph_Q \subseteq \aleph \restriction B \;\wedge \\ & \aleph \restriction (A \cup B) \setminus sync(C \cup D)) \\ & \quad = (\aleph_P \setminus sync(D)) \cup (\aleph_Q \setminus sync(C)) \end{aligned}$$

We may now give the semantic equation for alphabet parallel combination in the signals model:

$$\begin{aligned} \mathcal{F}_S[\![P \;{}_A\|_B\; Q]\!]\rho \;\widehat{=}\; \{(s, \aleph, t) \mid \exists\, s_P, \aleph_P, \aleph'_P, s_Q, \aleph_Q, \aleph'_Q \bullet \forall\, t' \bullet \\ & s \in s_P \;{}_A\tilde{\|}_B\; s_Q \;\wedge\; \aleph \in \aleph_P \;{}_A\tilde{\|}_B\; \aleph_Q \;\wedge \\ & sync(\sigma(s_Q \restriction D \uparrow t')) \subseteq \sigma(\aleph'_P \restriction A \uparrow t') \;\wedge \\ & sync(\sigma(s_P \restriction C \uparrow t')) \subseteq \sigma(\aleph'_Q \restriction B \uparrow t') \;\wedge \\ & (s_P, \aleph_P \cup \aleph'_P, t) \in \mathcal{F}_S[\![P]\!]\rho \;\wedge \\ & (s_Q, \aleph_Q \cup \aleph'_Q, t) \in \mathcal{F}_S[\![Q]\!]\rho \;\} \end{aligned}$$

If the two components are to synchronise upon every event from Σ, then no signals may he propagated to the environment. The semantics of a lockstep parallel construct $P \parallel Q$ is thus

$$\begin{aligned}\mathcal{F}_S[\![P \parallel Q]\!]\rho \;\;\hat{=}\;\; \{(s, \aleph_P \cup \aleph_Q, t) \mid s = s \downarrow \Sigma \;&\wedge\; (s, \aleph_P, t) \in \mathcal{F}_S[\![P]\!]\rho \\ &\wedge\; (s, \aleph_Q, t) \in \mathcal{F}_S[\![Q]\!]\rho\}\end{aligned}$$

No such restriction need be applied to the interleaving parallel operator:

$$\begin{aligned}\mathcal{F}_S[\![P \mid\mid\mid Q]\!]\rho \;\;\hat{=}\;\; \{(s, \aleph, t) \mid \exists\, s_P, s_Q \bullet s \in s_P \mid\mid\mid s_Q \;&\wedge\; (s_P, \aleph, t) \in \mathcal{F}_S[\![P]\!]\rho \\ &\wedge\; (s_Q, \aleph, t) \in \mathcal{F}_S[\![Q]\!]\rho\}\end{aligned}$$

Signals and synchronisations are simply interleaved.

8.4 Consistency

The Timed Signals model is an extension of the Timed Failures model. If a process description P does not mention signal events, the semantics of P in $TM_{\widehat{F}}$ will be equivalent to the semantics of P in TM_F. If we use π to denote the natural projection mapping between the two models, we may assert that

$$\mathcal{F}_T[\![P]\!] \;\;=\;\; \pi(\mathcal{F}_S[\![P]\!])$$

for any closed term P constructed without signal events.

Definition 8.2 A *TCSP* term is said to be signal-free if it contains no occurrences of events from $\widehat{\Sigma}$. This will be true whenever the term contains no subterms which match any of

$$\widehat{SKIP}\;,\; \widehat{WAIT}\; t,\;\; \widehat{a} \rightarrow P\;, \text{ or }\;\; P\;{}_A\|_B\; Q$$

where $(A \cup B) \cap \widehat{\Sigma} \neq \{\}$. ◇

With this definition, our consistency result is expressed by the theorem below:

Theorem 8.3 If P is closed and signal-free, and any recursive subterms in P are constructive for the corresponding variables, theu

$$\mathcal{F}_T[\![P]\!] \;\;=\;\; \pi(\mathcal{F}_S[\![P]\!])$$

where projection mapping $\pi : TM_{\widehat{F}} \rightarrow TM_F$ is given by

$$\pi(S) \;\;\hat{=}\;\; \{(s, \aleph) \mid \exists\, t \bullet (s, \aleph, t) \in S\}$$

♡

Proof We proceed by structural induction. To show that our result is true for recursively-defined processes, we must adopt a slightly stronger result for our inductive hypothesis. We begin by extending our definition of *signal-free* to semantic sets and environments:

Definition 8.4 If S is an element of $TS_{\tilde{F}}$, we say that S is signal-free if

$$\forall (s, \aleph, t) \in S \quad \bullet \quad \widehat{\sigma}(s) = \{\}$$

Further, if $\rho \in \widetilde{ENV}$, we say that ρ is signal-free if

$$\forall X \in VAR \quad \bullet \quad \rho[\![X]\!] \text{ is signal-free}$$

$$\Diamond$$

It is easy to see that the semantic set $\mathcal{F}_S[\![P]\!]\rho$ will be signal-free whenever both P and ρ are signal-free. If we extend our projection mapping to environments with

$$\pi\rho \;\widehat{=}\; \{X \mapsto \pi(\rho[\![X]\!]) \mid X \in VAR\}$$

for any ρ in $\widetilde{ENV}$, we may also conclude that

$$\rho \text{ is signal-free} \;\Rightarrow\; \pi\rho \in ENV$$

We may now state our inductive hypothesis: if P is a signal-free term, then

$$\forall \rho \in \widetilde{ENV} \quad \bullet \quad \rho \text{ is signal-free} \Rightarrow \mathcal{F}_T[\![P]\!](\pi\rho) = \pi(\mathcal{F}_S[\![P]\!]\rho)$$

base case It is sufficient to consider the case of the deadlock process. For any environment ρ in $\widetilde{ENV}$, we have

$$\mathcal{F}_S[\![STOP]\!]\rho \;\widehat{=}\; \{(s, \aleph, t) \mid s = \langle\rangle \wedge t \geqslant end(\aleph)\}$$

whence

$$\begin{aligned} \pi(\mathcal{F}_S[\![STOP]\!]) &= \pi(\{(s, \aleph, t) \mid s = \langle\rangle \wedge t \geqslant end(\aleph)\}) \\ &= \{(s, \aleph) \mid s = \langle\rangle\} \\ &= \mathcal{F}_T[\![STOP]\!] \end{aligned}$$

Although trace s is an element of $T\tilde{\Sigma}^*_{\leqslant}$, we know that $s \downarrow \widehat{\Sigma} = \langle\rangle$, and hence that $s \in T\Sigma^*_{\leqslant}$.

inductive step Consider the case of the alphabet parallel operator. If the parallel combination $P \;{}_A\|_B\; Q$ is signal-free, then the same must be true of components

P and Q. If s_P and s_Q are the traces corresponding to each component, we may conclude that $s_P = s_P \upharpoonright \Sigma$ and $s_Q = s_Q \upharpoonright \Sigma$. Iu this case,

$$\begin{aligned} s \in s_P \;{}_A\tilde{\|}_B\; s_Q \;\Leftrightarrow\;& s \upharpoonright \Sigma \upharpoonright (A \cap B) = s_P \upharpoonright (A \cap B) = s_Q \upharpoonright (A \cap B) \;\wedge \\ & s \upharpoonright \Sigma \upharpoonright (A - B) = s_P \setminus sync(D) \;\wedge \\ & s \upharpoonright \Sigma \upharpoonright (B - A) = s_Q \setminus sync(C) \;\wedge \\ & s_P \upharpoonright (A \cap sync(D)) \subseteq sync(s_Q) \;\wedge \\ & s_Q \upharpoonright (B \cap sync(C)) \subseteq sync(s_P) \;\wedge \\ & s \upharpoonright \hat{\Sigma} = \langle\rangle \;\wedge\; s \upharpoonright \Sigma = s \upharpoonright \Sigma \upharpoonright (A \cup B) \end{aligned}$$

From our assumption that the parallel combination is signal-free, we have that

$$A \cap \hat{\Sigma} = B \cap \hat{\Sigma} = \{\}$$

and hence that sets C and D are empty. From this, we deduce that

$$\begin{aligned} s \in s_P \;{}_A\tilde{\|}_B\; s_Q \;&\Leftrightarrow\; s \upharpoonright (A - B) = s_P \wedge s \upharpoonright (B - A) = s_Q \wedge s = s \upharpoonright (A \cup B) \\ &\Leftrightarrow\; s \in s_P \;{}_A\|_B\; s_Q \end{aligned}$$

We may also infer that

$$\begin{aligned} \aleph \in \aleph_P \;{}_A\tilde{\|}_B\; \aleph_Q \;\Leftrightarrow\;& \aleph_P \subseteq \aleph \upharpoonright A \;\wedge\; \aleph_Q \subseteq \aleph \upharpoonright B \;\wedge \\ & \aleph \upharpoonright (A \cup B) \setminus sync(C \cup D)) \\ & \quad = (\aleph_P \setminus sync(D)) \cup (\aleph_Q \setminus sync(C)) \\ \Leftrightarrow\;& \aleph_P \subseteq \aleph \upharpoonright A \;\wedge\; \aleph_Q \subseteq \aleph \upharpoonright B \;\wedge\; \aleph \upharpoonright (A \cup B) = \aleph_P \cup \aleph_Q \end{aligned}$$

We may use these results to simplify the semantics given for alphabet parallel combination in the signals model:

$$\begin{aligned} \mathcal{F}_S[\![P \;{}_A\|_B\; Q]\!]\rho \;\equiv\; \{(s, \aleph, t) \mid \exists\, s_P, \aleph_P, s_Q, \aleph_Q \bullet\; & s \in s_P \;{}_A\|_B\; s_Q \;\wedge \\ & \aleph \upharpoonright (A \cup B) = \aleph_P \cup \aleph_Q \;\wedge \\ & \aleph_P \subseteq \aleph \upharpoonright A \wedge \aleph_Q \subseteq \aleph \upharpoonright B \;\wedge \\ & (s_P, \aleph_P, t) \in \mathcal{F}_S[\![P]\!]\rho \;\wedge \\ & (s_Q, \aleph_Q, t) \in \mathcal{F}_S[\![Q]\!]\rho\} \end{aligned}$$

If we assume that ρ is signal-free, we may apply our inductive hypothesis to components P and Q, yielding

$$\begin{aligned} \pi(\mathcal{F}_S[\![P \;{}_A\|_B\; Q]\!]\rho) \;=\; \{(s, \aleph) \mid \exists\, s_P, \aleph_P, s_Q, \aleph_Q \bullet\; & s \in s_P \;{}_A\|_B\; s_Q \;\wedge \\ & \aleph \upharpoonright (A \cup B) = \aleph_P \cup \aleph_Q \;\wedge \\ & \aleph_P \subseteq \aleph \upharpoonright A \wedge \aleph_Q \subseteq \aleph \upharpoonright B \;\wedge \\ & (s_P, \aleph_P) \in \mathcal{F}_T[\![P]\!]\pi\rho \;\wedge \\ & (s_Q, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\pi\rho\} \\ \equiv\; \mathcal{F}_T[\![P \;{}_A\|_B\; Q]\!]\pi\rho \qquad\qquad & \end{aligned}$$

The case of the recursion operator requires the following lemma:

Lemma 8.5 The set of signal-free semantic sets,

$$SF \;\widehat{=}\; \{S \in TS_{\widetilde{F}} \mid S \text{ is signal-free}\}$$

is a complete subspace of $TS_{\widetilde{F}}$ under metric $\widetilde{d}$. ♡

The following definitions are taken from [Sutherland 75]:

Definition 8.6 A sequence $\{S_n\}$ in a metric space (M, d) is a *Cauchy sequence* if given $\epsilon > 0$, there exists N such that $d(S_n, S_m) < \epsilon$ for any $m, n > N$. ◇

Definition 8.7 A metric space (M, d) is said to be *complete* iff every Cauchy sequence in (M, d) converges to a point in M. ◇

Proof of lemma 8.5 [after Reed] Suppose that $\{S_n\}$ is a Cauchy sequence in metric space $(SF, \widetilde{d})$, and let $\{n_i\}$ be a sequence of positive integers such that

$$\begin{aligned} \forall\, i \geqslant 0 \;&\bullet\; i < n_i < n_{i+1} \\ \forall\, m \geqslant n \;&\bullet\; \widetilde{d}(S_m, S_{n_i}) \end{aligned}$$

Recall that the metric $\widetilde{d}$ was given by

$$\widetilde{d}(S, T) \;\widehat{=}\; \inf\{\{2^{-t} \mid S(t) = T(t)\} \cup \{1\}\}$$

where $S(t)$ denotes the set of observations from S that end no later than time t:

$$S(t) \;\widehat{=}\; \{(s, \aleph, t') \in S \mid t' \leqslant t\}$$

With such a metric, the limit of the Cauchy sequence S_n is equal to

$$S \;\widehat{=}\; \bigcup_{i \geqslant 0} S_{n_i}(i)$$

By our choice of sequence n_i we have that

$$0 \leqslant t \leqslant i \;\Rightarrow\; S(t) = S_{n_i}(t)$$

and each S_n is signal-free. Hence we may observe that

$$\begin{aligned} (s, \aleph, t) \in S \;&\Rightarrow\; (s, \aleph, t) \in S(t) \\ &\Rightarrow\; \exists\, i \bullet (s, \aleph, t) \in S_{n_i}(t) \\ &\Rightarrow\; \exists\, i \bullet (s, \aleph, t) \in S_{n_i} \\ &\Rightarrow\; \widehat{\sigma}(s) = \{\} \end{aligned}$$

and conclude that the limit S is also signal-free. The set TF is thus a complete subspace of $TS_{\widetilde{F}}$. □

The semantics of immediate recursion in the signals model is given by:

$$\mathcal{F}_S[\![\mu X \circ P]\!]\rho \;\hat{=}\; \text{the unique fixed point of the mapping } \widetilde{M}(X,P)\rho$$

where tbe mapping $\widetilde{M}(X,P)\rho$ is given by

$$\widetilde{M}(X,P)\rho \;\hat{=}\; \lambda\, Y \bullet \mathcal{F}_S[\![P]\!]\rho[Y/X]$$

In onr indnctive hypothesis, we have assumed that $\mu X \circ P$ and ρ are both signal-free. From tbis, we dednce that

$$Y \text{ is signal-free } \Rightarrow \mathcal{F}_S[\![P]\!]\rho[Y/X] \text{ is signal-free}$$

and hence that subspace SF is closed under the mapping $\widetilde{M}(X,P)\rho$. If we suppose that P is constructive for variable X, then this mapping has a unique fixed point, and this fixed point must lie in SF. We may conclude that the semantic set

$$\mathcal{F}_S[\![\mu X \circ P]\!]\rho$$

is signal-free.

Applying the fixed point property, we may obtain that

$$\begin{aligned}\mathcal{F}_S[\![\mu X \circ P]\!]\rho &= \widetilde{M}(X,P)\rho\,(\mathcal{F}_S[\![\mu X \circ P]\!]\rho)\\ &= \mathcal{F}_S[\![P]\!]\rho[\mathcal{F}_S[\![\mu X \circ P]\!]\rho/X]\end{aligned}$$

Term P, environment ρ, and semantic set $\mathcal{F}_S[\![\mu X \circ P]\!]\rho$ are all signal-free, so we may apply our inductive hypothesis to yield that

$$\begin{aligned}\pi(\mathcal{F}_S[\![\mu X \circ P]\!]\rho) &= \mathcal{F}_T[\![P]\!]\pi(\rho[\mathcal{F}_S[\![\mu X \circ P]\!]\rho/X])\\ &= \mathcal{F}_T[\![P]\!]\pi\rho[\pi(\mathcal{F}_S[\![\mu X \circ P]\!]\rho)/X]\\ &= M(X,P)\pi\rho\,(\pi(\mathcal{F}_S[\![\mu X \circ P]\!]\rho)\end{aligned}$$

where the mapping $M(X,P)\pi\rho$ is as defined in chapter 3:

$$M(X,P)\pi\rho \;=\; \lambda\, Y \bullet \mathcal{F}_T[\![P]\!]\pi\rho[Y/X]$$

We have shown that $\pi(\mathcal{F}_S[\![\mu X \circ P]\!]\rho)$ is a fixed point of this mapping, but we know that

$$\mathcal{F}_T[\![\mu X \circ P]\!]\pi\rho \;=\; \text{the } \textit{unique} \text{ fixed point of the mapping } M(X,P)\pi\rho$$

so we may conclude that, provided that P is constructive for variable X,

$$\pi(\mathcal{F}_S[\![\mu X \circ P]\!]\rho) \;=\; \mathcal{F}_T[\![\mu X \circ P]\!]\pi\rho$$

The remaining cases for the inductive step are comparatively simple. We conclude that the inductive hypothesis holds for all terms: if P is a signal-free term in *TCSP*, then

$$\forall \rho \in \widetilde{ENV} \quad \bullet \quad \rho \text{ is signal-free} \Rightarrow \mathcal{F}_T[\![P]\!](\pi\rho) = \pi(\mathcal{F}_S[\![P]\!]\rho)$$

under the assumption that any recnrsive terms are constrnctive for the corresponding variables. The conclusion of theorem 8.3 follows immediately: if P is closed and signal-free, then

$$\mathcal{F}_T[\![P]\!] \quad = \quad \pi(\mathcal{F}_S[\![P]\!])$$

□

Furthermore, it is easy to see that the defining axioms of $TM_{\tilde{F}}$ are consistent with those of TM_F, in the sense that

$$S \in TM_{\tilde{F}} \quad \Rightarrow \quad \pi(S) \in TM_F$$

If a set S satisfies the axioms of the Signals model, the projection $\pi(S)$ satisfies those of the Timed Failures model.

We conclude that the semantics given in this chapter are consistent with the equations and axioms presented earlier, and hence that the Signals model may be regarded as an extension of the Timed Failures model.

8.5 Example

As an application of the Signals model, we consider a Timed CSP implementation of the physical layer of an Ethernet-like protocol. This layer provides a means of commnnication between the nodes of a local area network; data bits are accepted from the data link at each node, and passed along a broadcast medium. In section 7.4, we saw that the service provided by the physical layer could be captured as a timed failures specification. With the addition of signal events, we can produce a *TCSP* description to satisfy that specification.

The service provided by the physical layer was described in terms of the availability and occurrence of synchronisation events from the following set:

$$A_{DL} \quad \widehat{=} \quad \{i.put.b, i.cs, i.cd, i.get.b \mid i : NODE\,;\, b : BIT\}$$

At each node i, the physical layer shares two channels and two simple synchrouisations with the data link component. The channels carry data bits between the two layers: bits are accepted from the data link layer along channel $i.put$, and transmitted to the data link along channel *i.get*. The synchronisation *i.cs* is made available to the data link whenever activity ceases on the broadcast medium, and the synchronisation *i.cd* is offered whenever a collision is taking place.

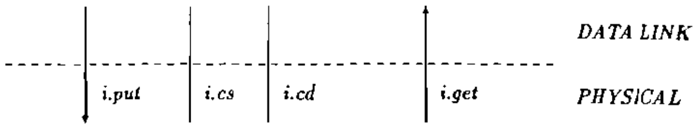

Figure 8.1: the service provided by the physical layer at node *i*

The event *i.put.b* models the acceptance of a data bit *b* by the physical layer at node *i*. If the physical layer is to meet the specification given in chapter 7, a corresponding signal must be placed upon the broadcast medium. Even so, there is no guarantee that the data bit will be received at another node *j*; if other nodes are transmitting, then this bit signal may be lost. This behaviour is easy to model if we introduce a set of signal events:

$$\hat{\Sigma}_{PL} \;\widehat{=}\; \{ i.\widehat{at}.b \mid i : NODE \,;\, b : BIT \}$$

The event $j.\widehat{at}.b$ models the arrival of a bit *b* on the broadcast medium at node *j*.

Transmission

If a data bit *b* is passed to the physical layer along channel *i.put*, it should be broadcast to every other node on the network.

$$TRANS_i \;\widehat{=}\; i.put.b \xrightarrow{t_{bit}} \left(TRANS_i \;\;|||\;\; \Big|\Big|\Big|_{j \neq i} WAIT\ d_{ij} \,;\, j.\widehat{at}.b \rightarrow STOP \right)$$

We have decided that signal $j.\widehat{at}.b$ should occur at a time $d_{ij} + t_{bit}$ after the input event *i.put.b*. The bit time t_{bit} is the duration of a bit transmission on the broadcast medium.

The behaviour following an input event is an interleaving of two processes. The first is a fresh copy of the broadcast process: the physical layer at node *i* is ready to accept a new bit for transmission once time t_{bit} has elapsed. The second is an interleaving of simple transmission processes: each of these will produce a signal $j.\widehat{at}.b$ at the correct time, and then terminate.

Reception

The arrival of a data bit b at node i is modelled by the signal event $i.\widehat{at}.b$. If the physical layer at node i is ready, it will synchronise upon this event, and offer data bit b to the data link.

$$LISTEN_i \;\widehat{=}\; i.at.b : AT_i \xrightarrow{t_1} i.get.b \xrightarrow{t_2} LISTEN_i$$

where

$$AT_i \;\widehat{=}\; \{i.at.0, i.at.1\}$$

The combination of delays $t_1 + t_2$ must be strictly less than t_{bit} if the physical layer is to function correctly. In a valid frame sequence, a data bit is transmitted every t_{bit}; the physical component must be capable of decoding a data bit signal and passing it to the data link within this time. If the above process is not ready to observe signal $i.\widehat{at}.b$, then data bit b will not be received.

Carrier Sense

If more than two bit times have elapsed since the last bit arrived at node i, then the synchronisation $i.cs$ should be offered to the client layer. This offer should be withdrawn if another event from AT_i is observed. The following process will meet these requirements:

$$SENSE_i \;\widehat{=}\; (i.cs \rightarrow STOP) \underset{AT_i}{\nabla} NOISE_i$$

Once an at event is observed, control is passed to a process which offers to synchronise upon events from AT_i.

$$NOISE_i \;\widehat{=}\; (a : AT_i \xrightarrow{t_3} NOISE_i) \overset{2t_{bit}}{\rhd} SENSE_i$$

If more than two bit times have elapsed since the last at event, this process withdraws the offer, and passes control to a copy of the original process $SENSE_i$.

Collision Detection

If a data bit arrives from another node while node i is transmitting, then synchronisation $i.cd$ should be offered to the data link layer. Accordingly, we define a process $DETECT_i$ which waits for i to start transmitting.

$$DETECT_i \;\widehat{=}\; a : PUT_i \xrightarrow{t_4} (MONITOR_i \underset{AT_i}{\nabla} COLLISION_i) \,;\, DETECT_i$$

Once a transmission has begun, control is passed to a monitor process. Observe that the delays t_4 and t_5 must be less than t_{bit}, or this process will interfere with frame transmission.

$$MONITOR_i \;\hat{=}\; (a : PUT_i \xrightarrow{t_5} MONITOR_i) \overset{2t_{bit}}{\rhd} SKIP$$

This process offers to engage in events from $\{i.put.0, i.put.1\}$, until two bit times elapse without a *put* event. When this happens, the monitor process terminates successfully.

If a data bit arrives during a transmission, control is passed to the process $COLLISION_i$, which behaves as $MONITOR_i$, except that it is ready to engage in the event $i.cd$.

$$COLLISION_i \;\hat{=}\; MONITOR_i \;|||\; i.cd \rightarrow STOP$$

As this is an interleaved parallel combination, it will terminate successfully when transmission ceases, and $MONITOR_i$ terminates. When this happens, control is passed to another copy of the original process.

Combination

The physical layer component at node i is the parallel combination of the processes defined above:

$$PL_i \;\hat{=}\; (TRANS_i \underset{PUT_i}{\|} DETECT_i) \;|||\; SENSE_i \;|||\; LISTEN_i$$

The transmission and collision detect processes must agree on each occurrence of an event from PUT_i, but no other synchronisation is required. The physical layer itself is a parallel combination of node processes

$$PHYSICAL \;\hat{=}\; \|_{ALL_i} PL_i$$

where ALL_i is the set of all events that are possible for node i:

$$ALL_i \;\hat{=}\; PUT_i \cup GET_i \cup AT_i \cup \{i.cs, i.cd\} \cup \{j.\widehat{at}.b \mid j \in NODE \wedge j \neq i\}$$

Only the broadcast signal events are seen by more than one node.

Chapter 9

Discussion

9.1 Conclusions

In this thesis, we have presented a formal method for the specification and development of real-time systems. We have exhibited a system description language with a number of useful programming features. We have introduced a formal specification language for the description and analysis of system behaviour. We have presented a complete, compositional proof system for relating the two languages, and formulated techniques for simplifying the proof obligations that arise during the development process. Finally, we have extended the method to include a treatment of broadcast communication.

A substantial case study was undertaken to demonstrate the applicability of the development method, and proved successful. To assess the performance of the method, it is necessary to consider the role of formal methods in systems development: initially, a set of informal requirements describing the intended behaviour of a system are translated into an abstract formal specification; this specification is then gradually refined towards some final implementation. If each refinement step is formally verified, then we may be certain that any behaviour of the implementation will be consistent with the original specification. However, as [Barringer 87] points out:

* the gap between informal requirements and formal specification means that there is no guarantee that the system performs as originally intended;

* as soon as realistically sized systems are considered, shortcuts have to be taken; the number of formal proofs required is just far too large.

These are valid criticisms, and must be addressed if our development method is to be of any practical use.

The specification language formulated in chapters four and six is an attempt to answer the first of these criticisms. If the intended behaviour of the system may be described in terms of the observation and availability of some set of communication events, then this language may be used to capture the system requirements in a clear and comprehensible fashion. The resulting specification may be translated into timed failures notation. Alternatively, we may derive inference rules which relate such specifications directly to the implementation language.

Once the system requirements have been formalised, the specification language may be used to reason about system properties, and to communicate the details of the design to others. At this stage of the development, we will often detect inconsistencies and ambiguities in the original requirements. Even if the development is then completed informally, the production of a formal specification will have improved the safety and reliability of the system.

The system description language presented in chapters two and three is significantly larger than that proposed in [Reed 88]. We have extended the language to include process variables, primitive timing operators, and new operators for sequential composition, parallel composition and recursion. Although the extended syntax is harder to reason about—there are more cases to consider—it is easier to reason *with*. In realistic applications, such as the case study of chapter seven, we find that the new operators correspond more closely with our requirements, resulting in an elegant implementation with a simple semantics.

The complete proof system introduced in chapter five provides a formal link between the specification language and the system description language. Given a proposed implementation of a system component, we may use the inference rules presented in chapter five to establish that it behaves as expected. The compositional nature of the proof system supports the hierarchical development of large, complex systems: we may reason about the behaviour of each component in isolation. The notion of environmental assumptions, introduced in chapter four, proves particularly useful in these circumstances.

The second criticism is more challenging: real-time systems are complicated entities, and the proof obligations generated during the development process are necessarily complex. The theory of timewise refinement presented in chapter five can be used to reduce any proof obligations which correspond to untimed safety conditions: if we can show that these requirements are satisfied in Reed's untimed Traces model, then we may conclude that they are also satisfied in the context of the Timed Failures model.

The treatment of scheduling and abstraction introduced in chapter six provides another method of reducing the complexity of proof obligations. By separating the concerns of scheduling and concealment, we are able to present our requirements in a clear and structured fashion, as illustrated by the development method for hierarchical protocols described at the beginning of chapter seven.

Even with the techniques described above, when we come to apply Timed CSP to the specification and development of complex real-time systems, we find that the number of formal proofs required is still uncomfortably large. However, we may replace many of these proofs with rigorous mathematical arguments, and still be reasonably sure that our implementation is correct. Where doubts remain, we may increase the degree of formality until the truth, or falsity, of the argument becomes apparent.

In a description of a real-time system, it is sometimes convenient to include observable events that are not synchronisations: output events which may occur without the cooperation of the environment. In chapter eight, we showed that our model of computation could be extended to include a treatment of broadcast communication. Not only does this make it easier to describe and analyse certain aspects of behaviour, but it may also be used as a basis for modelling assignment in our system description language. This is the subject of current research, and will be discussed at the end of this chapter.

This thesis has presented a formal development method for real-time systems, based upon the models proposed by Reed and Roscoe. This method supports both formal and rigorous reasoning at every stage of system development, and is applicable to systems of a realistic size. It is our hope that the results of the research described in this thesis may be used to improve the safety and reliability of real-time distributed systems.

9.2 Other Approaches

A wide variety of formal methods[1] have been proposed for the specification and development of real-time systems, based upon

* process algebras, such as Timed ACP [Baeten & Bergstra 89]
* temporal logics, such as that presented in [Barringer *et al.* 84]
* programming languages, such as ESTEREL [Berry & Gonthier 88]

Although much research has been carried out into the theory of timed concurrency, a consensus has yet to emerge concerning the applicability of the various formalisms to different types of system. A successful development method is likely to involve some combination of the features mentioned above. A notation that is well-suited

[1]A useful review is presented in [Joseph & Goswami 88].

to requirements capture is unlikely to be an efficient programming language, and *vice versa.*

The process algebras—

* SCCS [Milner 83]
* TCCS [Moller & Tofts 90]
* ATP [Nicollin *et al.* 90]
* Timed LOTOS [Quemada & Fernandez 87]
* Timed ACP [Baeten & Bergstra 89]
* Timed CCS [Wang 90, Hennessy & Regan 90]

—rely upon bisimulation relations to prove correctness. Two processes are said to be bisimilar if they exhibit the same behaviour according to the operational semantics for the language. To show that an implementation meets a given specification, we describe both as processes, and show that the two descriptions are bisimilar.

This approach has proved successful in an untimed context, but is difficult to apply to complex real-time systems. A great deal of information is present in each process description: as specifications, they are difficult to understand and unsuitable for rigorous, rather than formal, reasoning. We do not exploit the algebraic properties of Timed CSP: our method of proof is quite different, and we employ separate languages for system description and formal specification.

The ESTEREL programming language [Berry & Gonthier 88] is a deterministic language based upon a synchrony hypothesis: the outputs of a system are conceptually synchronous with its inputs. If it can be assumed that the system under consideration takes no time to execute the operations required of it, then that system may be programmed in ESTEREL, and compiled into a language of finite automata. The language is given a semantics in terms of rewrite rules; no development method comparable to ours has been presented.

More relevant to the development method outlined in this thesis is the work described in [Hooman & Widom 89]. In this paper, the authors present a compositional proof system relating an occam-like language to a quantitative temporal logic, similar to the one developed in [Koymans & de Roever 83]. Although the system description language is somewhat limited, it is clear that quantitative temporal logics are useful assertion languages—indeed, [Jackson 90] shows how such a logic may be employed as a specification language for Timed CSP. It would be interesting to see the proof system applied in the development of a large, complex system.

In [Shasha *et al.* 83], the authors use a quantitative temporal logic to prove the correctness of a carrier-sense broadcast protocol, similar to the one described in chapter seven. By assuming a simplified version of the service provided by the physical layer, and an internal specification of the data link layer, the authors are able to establish that certain desirable properties hold of the network. The sketch proof provided is similar to the rigorous justification of the data link service presented in chapter seven.

In terms of complexity of specification, and support for formal reasoning, there is little to choose between quantitative temporal logic and the notation presented in this thesis. However, the structuring mechanisms of Timed CSP, and the exclusive treatment of communication, are of some advantage when large systems are considered. We have yet to see a large-scale application of qnantitative temporal logic to the hierarchical development of complex real-time systems.

9.3 Future Work

If the development method described in this thesis is to support formal reasoning at every stage of the development process, we must bridge the gap between the system description language and executable code. We are fortunate in that there exists a powerful programming language based upon CSP, the occam language of [Inmos 88]. We propose to establish a refinement relation between a subset of Timed CSP, corresponding to occam-implementable processes, and a subset of occam. To provide a formal basis for this refinement relation, we must give a denotational semantics to occam in the style of the Timed Failures model.

Towards this end, we may use the model for broadcast communication presented in chapter eight to provide a basis for modelling assignment in Timed CSP. Instead of adding a signal event for every synchronisation, we extend the alphabet Σ with a set of assignment events Γ.

$$\begin{aligned} \tilde{\Sigma} &\mathrel{\widehat{=}} \Sigma \cup \Gamma \\ \Gamma &\mathrel{\widehat{=}} Var \times Val \end{aligned}$$

Each assignment event is a pair $x.\alpha$, representing the assignment of value α to variable x. If we choose Ψ to denote the set of possible states,

$$\Psi \mathrel{\widehat{=}} Var \rightarrow Val$$

then we may define a semantic function

$$\mathcal{F}_S \in TCSP \rightarrow \Psi \rightarrow TM_{\tilde{F}}$$

This function takes a language construct, and an initial state ψ, and returns a set of possible observations. For example, the semantics of the assignment statement $x := e \,;\, P$ would be given by

$$\begin{array}{rcl}\mathcal{F}_S[\![x := e \,;\, P]\!]\psi & \mathrel{\widehat{=}} & \{(\langle\rangle, \{\}, 0)\} \\ & & \cup \\ & & \{(\langle(0, x.\alpha)\rangle^\frown s, \aleph, t) \mid \alpha = \psi[\![e]\!] \;\wedge \\ & & \qquad begin(s) \geqslant \delta \;\wedge \\ & & \qquad \psi' = \psi \oplus \{x \mapsto \alpha\} \;\wedge \\ & & \qquad (s, \aleph, t) - \delta \in \mathcal{F}_S[\![P]\!]\psi' \,\}\end{array}$$

We may use the state component to give a semantics to conditional statements, as well as input and output instructions.

As we discovered in chapter seven, Timed CSP lacks any mechanism for reasoning about probabilistic aspects of system behaviour. Such a mechanism would allow us to analyse the performance of communication protocols. However, a semantic model which allows us to formalise statements such as *the system responds within 5 time units, with a probability of 0.5* will be complex indeed. Although substantial progress has been made towards an nntimed probabilistic model for CSP [Seidel 90], little has been done to combine probability and time. This is an area for future research.

Another area for research is the development of a simulated time model for CSP: a real-time model which supports an algebra of processes. If we discard the realism assumption of our computational model, which places a bound upon the rate of progress of a process, we may exhibit algebraic laws for the elimination of concurrency. These laws may be useful in establishing the correctness of compilers for a language with timing constructs, which must simulate the flow of time. Such a model might be based upon the operational semantics for Timed CSP given in [Schneider 91].

The operational semantics may also be used to define an infinite Timed Failures model, in which process behaviours are represented hy infinite traces and infinite refusal sets. Such a model would support a theory of timewise refinement hased npon the untimed Failures model, and provide a more straightforward semantics for the hiding operator: we might distiuguish an A-active behaviour by the inclusion of the set $[0, \infty) \times A$ in the timed refusal.

Finally, if we wish Timed CSP to be adopted by industrial users, it is essential that the development process is supported by reliable software tools—to manipulate formal specifications, and to assist in verification—we cannot expect methods to reach maturity without leaving their home environment.

References

[Baeten & Bergstra 89]

J.C.M. Baeten and J.A. Bergstra, *Real Time Process Algebra,* Report P8916, Programming Research Group, University of Amsterdam 1989.

[Barringer 87]

H. Barringer, *The Use of Temporal Logic in the Compositional Specification of Concurrent Systems,* in *Temporal Logics and their Applications,* Academic Press 1987.

[Barringer *et al.* 84]

H. Barringer, R. Kuiper, and A. Pneuli, *Now You May Compose Temporal Logic Specifications,* Proceedings of the Sixteenth ACM Symposium on Theory of Computing (1984).

[Barringer *et al.* 85]

H. Barringer, R. Kuiper, and A. Pneuli, *A Compositional Temporal Approach to a CSP-like Language,* in *Formal Models in Programming* E.J. Neuhold and G. Chroust (eds.), North-Holland 1985.

[Berry & Gonthier 88]

G. Berry and G. Gonthier, *The Esterel Synchronous Programming Language: Design, Semantics, Implementation,* Rapports de Recherche 842, INRIA Sophia-Antipolis 1988.

[Berry 89]

G. Berry, *Real Time Programming: Special Purpose or General Purpose Languages,* in *Information Processing 89,* G.X. Ritter (ed.), North-Holland 1989.

[Brookes 83]

S.D. Brookes, *A Model for Communicating Sequential Processes,* Oxford University D.Phil thesis 1983.

[Bergstra & Klop 84]

J.A. Bergstra and J.W. Klop, *Process Algebra for Synchronous Communication,* Information and Control 60 (1984).

[Brinch Hansen 75]

Per Brinch Hansen, *Concurrent Pascal Report,* Technical Report CIT-IS-TR-17, California Institute of Technology 1975.

[Boucher & Gerth 87]

A. Boucher and R. Gerth, *A Timed Model for Extended Communicating Sequential Processes,* Proceedings of ICALP '87, Springer LNCS 267 (1987).

[Davies & Schneider 89]

J. Davies and S.A. Schneider, *Factorising Proofs in Timed CSP,* Proceedings of the Fifth Conference on the Mathematical Foundations of Programming Semantics, Springer LNCS 439 (1989).

[Enderton 77]

H.B. Enderton, *Elements of Set Theory,* Academic Press 1977.

[Hennessy & Regan 90]

M. Hennessy and T. Regan, *A Temporal Process Algebra,* Technical Report 2–90, University of Sussex 1990.

[Hoare 78]

C.A.R. Hoare, *Communicating Sequential Processes,* Communications of the ACM 21–8 (1978).

[Hoare 85]

C.A.R. Hoare, *Communicating Sequential Processes,* Prentice-Hall 1985.

[Hooman & de Roever 89]

J.J.M. Hooman and W.P. de Roever, *Design and verification in real-time distributed computing: an introduction to compositional methods,* Proceedings of the Ninth International Symposium on Protocol Specification, Testing and Verification, North-Holland 1989.

[Hooman & Widom 89]

J.J.M. Hooman and J. Widom, *A Temporal-Logic-Based Compositional Proof System for Real-time Message Passing,* Proceedings of PARLE 89, Springer LNCS 366 (1989).

[Hooman 90]

J.J.M. Hooman, *Compositional Proof Systems for Real-time Distributed Message Passing,* ESPRIT BRA-3096 (SPEC) deliverable, Eindhoven University of Technology 1990.

[Inmos 88]

Inmos Limited, *Occam 2 Reference Manual,* Prentice-Hall 1988.

[Jackson 90]

D.M.Jackson, *A Temporal Logic for Timed CSP,* Programming Research Group Technical Report TR-5-90, Oxford University 1990.

[Jahanian & Mok 86]

F. Jahanian and A.K. Mok, *Safety Analysis of Timing Properties in Real-Time Systems,* IEEE Transactions on Software Engineering, SE-12 (1986).

[Jeffrey 90]

A. Jeffrey, *Discrete Timed CSP*, Programming Methodology Group, Chalmers University of Technology (to appear).

[Jones 82]

G. Jones, *A Timed Model of Communicating Processes,* Oxford University D.Phil thesis 1982.

[Joseph & Goswami 88]

M. Joseph and A. Goswami, Formal Description of Real-time Systems: a review, Research Report 129, Department of Computer Science, University of Warwick 1988.

[Koymans & de Roever 83]

R. Koymans and W.P. de Roever, *Examples of a real-time temporal logic specification* in *The Analysis of Concurrent Systems*, Springer LNCS 207 (1983).

[Lamport 77]

L. Lamport, *Proving the Correctness of Multiprocess Programs,* Transactions on Software Engineering 3 (1977).

[Milner 80]

R. Milner, *A Calculus of Communicating Systems,* Springer LNCS 94 (1980).

[Milner 83]

R. Milner, *Calculi for Synchrony and Asynchrony,* Theoretical Computer Science 25 (1983).

[Milner 89]

R. Milner, *Communication and Concurrency,* Prentice-Hall 1989.

[Moller & Tofts 90]

F. Moller and C. Tofts, *A Temporal Calculus of Communicating Systems*, Proceedings of CONCUR 90, Springer LNCS 458 (1990).

[Nicollin *et al.* 90]

X. Nicollin, J.-L. Richier, J. Sifakis and J. Voiron, ATP: an Algebra for Timed Processes, Proceedings of the IFIP Working Conference on Programming Concepts and Methods, 1990.

[Quemada & Fernandez 87]

J. Quemada and A. Fernandez, *Introduction of Quantitative Relative Time into LOTOS,* in *Protocol Specification, Testing and Verification VII,* H. Rudin and C.H. West (eds.), North Holland 1987.

[Reed & Roscoe 86]

G.M. Reed and A.W. Roscoe, *A Timed Model for Communicating Sequential Processes,* Proceedings of ICALP '86, Springer LNCS 226 (1986); Theoretical Computer Science 58 (1988).

[Reed & Roscoe 87]

G.M. Reed and A.W. Roscoe, *Metric Spaces as Models for Real-time Concurrency,* Proceedings of the Third Workshop on the Mathematical Foundations of Programming Language Semantics, LNCS 298 (1987).

[Reed 88]

G.M. Reed, *A Uniform Mathematical Theory for Real-time Distributed Computing,* Oxford University D.Phil thesis 1988.

[Roscoe 82]

A.W. Roscoe, *A Mathematical Theory of Communicating Processes,* Oxford University D.Phil thesis 1982.

[Schneider 89]

S.A. Schneider, *Correctness and Communication in Real-time Systems,* Oxford University D.Phil thesis 1989.

[Schneider 91]

S.A. Schneider, *An Operational Semantics for Timed CSP*, Programming Research Group Technical Report TR-1-91, Oxford University 1991.

[Seidel 90]

K. Seidel, *Probabilistic CSP: Work in Progress,* Programming Research Group, Oxford University 1990.

[Shasha *et al.* 83]

D.E. Shasha, A. Pneuli, and W. Ewald, *Temporal Verification of Carrier-Sense Local Area Network Protocols,* Proceedings of the 11th ACM Symposium on the Principles of Programming Languages (1983).

[Sutherland 75]

W.A. Sutherland, *Introduction to Metric and Topological Spaces*, Oxford University Press 1975.

[Tanenbaum 81]

A.S. Tanenbaum, *Computer Networks,* Prentice-Hall International 1981.

[Wang 90]

Wang Yi, *Real-time Behaviour of Asynchronous Agents,* Proceedings of CONCUR 90, Springer LNCS 458 (1990).

[Woodcock 90]

J.C.P. Woodcock, *Using Z,* Lecture Notes, Programming Research Group, Oxford University 1990.

[Xerox 80]

The Ethernet Specification, available from the Xerox Corporation, reprinted in ACM Computer Communication Review July 1981.

[Zwarico 86]

A.E. Zwarico, *A Formal Model of Real-Time Computing,* University of Pennsylvania Technical Report 1986.

Appendix A

Mathematical Proofs

A.1 Lemmata

We give derivations for two of the lemmata presented without proof in the body of the thesis. The first result requires a proof of semantic equivalence, while the second is representative of a series of results about constructive terms, presented at the beginning of chapter 3.

Communicating Parallel

In chapter 2, we claimed that

$$P \underset{C}{\|} Q \;\equiv\; c\,(l(P)\;{}_A\|_B\; r(Q))$$

where the process relabelling functions l, r, and c are given by:

$$\begin{array}{rclcrcll}
l(a) & \mathrel{\widehat=} & a & \text{if } a \in C & & & & \\
 & & l.a & \text{otherwise} & c(a) & \mathrel{\widehat=} & a & \text{if } a \in C \\
r(a) & \mathrel{\widehat=} & a & \text{if } a \in C & c(l.a) & \mathrel{\widehat=} & a & \text{if } a \notin C \\
 & & r.a & \text{otherwise} & c(r.a) & \mathrel{\widehat=} & a & \text{if } a \notin C
\end{array}$$

and

$$\begin{array}{rcl}
A & \mathrel{\widehat=} & l(\Sigma - C) \cup C \\
B & \mathrel{\widehat=} & r(\Sigma - C) \cup C
\end{array}$$

and we choose l and r such that

$$l(\Sigma) \cap C \;=\; r(\Sigma) \cap C \;=\; \{\}$$

We may establish this equivalence by demonstrating that, for any environment ρ

$$\mathcal{F}_T\llbracket P \underset{C}{\|} Q\rrbracket\rho \;\equiv\; \mathcal{F}_T\llbracket c\,(l(P)\ {}_A\|_B\ r(Q))\rrbracket\rho$$

Suppose that $(s, \aleph)$ is an element of $\mathcal{F}_T\llbracket P \underset{C}{\|} Q\rrbracket\rho$. In this case, we know that

$$\begin{aligned}\exists\, s_P, \aleph_P, s_Q, \aleph_Q \;\bullet\; s \in s_P \underset{C}{\|} s_Q \;&\wedge\; \aleph \downharpoonright C = (\aleph_P \cup \aleph_Q) \downharpoonright C\\ &\wedge\; \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C\\ &\wedge\; (s_P, \aleph_P) \in \mathcal{F}_T\llbracket P\rrbracket\rho\\ &\wedge\; (s_Q, \aleph_Q) \in \mathcal{F}_T\llbracket Q\rrbracket\rho\end{aligned}$$

Appealing to the semantic equation for the renaming operator, we see that the statement above is equivalent to

$$\begin{aligned}\exists\, s'_P, \aleph'_P, s'_Q, \aleph'_Q \;\bullet\; s \in s_P \underset{C}{\|} s_Q \;&\wedge\; s'_P = l(s_P) \wedge s'_Q = r(s_Q)\\ &\wedge\; \aleph_P = l^{-1}(\aleph'_P) \wedge \aleph_Q = r^{-1}(\aleph'_Q)\\ &\wedge\; \aleph \downharpoonright C = (\aleph_P \cup \aleph_Q) \downharpoonright C\\ &\wedge\; \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C\\ &\wedge\; (s'_P, \aleph'_P) \in \mathcal{F}_T\llbracket l(P)\rrbracket\rho\\ &\wedge\; (s'_Q, \aleph'_Q) \in \mathcal{F}_T\llbracket r(Q)\rrbracket\rho\end{aligned}$$

which is true if and only if

$$\begin{aligned}\exists\, s', \aleph', s'_P, \aleph'_P, s'_Q, \aleph'_Q \;\bullet\; s \in s_P \underset{C}{\|} s_Q \;&\wedge\; s = c(s') \wedge \aleph = c^{-1}(\aleph')\\ &\wedge\; s'_P = l(s_P) \wedge s'_Q = r(s_Q)\\ &\wedge\; \aleph_P = l^{-1}(\aleph'_P) \wedge \aleph_Q = r^{-1}(\aleph'_Q)\\ &\wedge\; \aleph \downharpoonright C = (\aleph_P \cup \aleph_Q) \downharpoonright C\\ &\wedge\; \aleph \setminus C = (\aleph_P \cap \aleph_Q) \setminus C\\ &\wedge\; (s'_P, \aleph'_P) \in \mathcal{F}_T\llbracket l(P)\rrbracket\rho\\ &\wedge\; (s'_Q, \aleph'_Q) \in \mathcal{F}_T\llbracket r(Q)\rrbracket\rho\end{aligned}$$

From our choice of s', s'_P, s'_Q, and the definitions given for A, B, l, r, and c, we may deduce that

$$s \in s_P \underset{C}{\|} s_Q \;\Leftrightarrow\; s' \in s'_P \underset{C}{\|} s'_Q \;\Leftrightarrow\; s' \in s'_P\ {}_A\|_B\ s'_Q$$

We may also deduce that

$$\aleph'_P \subseteq \aleph' \downharpoonright A \;\wedge\; \aleph'_Q \subseteq \aleph' \downharpoonright B \;\wedge\; \aleph' \downharpoonright (A \cup B) = \aleph'$$

If we recall the semantics of the alphabet parallel operator, it is clear that the previous existentially quantified statement is equivalent to

$$\exists\, s', \aleph' \quad \bullet \quad s = c(s') \wedge \aleph = c^{-1}(\aleph') \wedge (s', \aleph') \in \mathcal{F}_T[\![l(P)\ {}_A\|_B\ r(Q)]\!]\rho$$

which is true if and only if

$$(s, \aleph) \ \in \ \mathcal{F}_T[\![c\ (l(P)\ {}_A\|_B\ r(Q))]\!]\rho$$

We conclude that the two terms are semantically equivalent. □

Constructive Terms

Section 3.1 included several lemmas about constructive terms. Each of these may be derived from the semantic equations given in chapters 2 and 3. As an example, consider the first clause of lemma 3.5:

Lemma 3.5 If P is t-constructive for X,

1. $a \xrightarrow{t_0} P$ and $WAIT\ t_0\ ; P$ are $(t + t_0)$-constructive for X

◇

Proof Term P is t-constructive for variable X if and only if

$$\begin{aligned}&\forall\, t' : TIME\, ; \rho : ENV \bullet \\ &\qquad \mathcal{F}_T[\![P]\!]\rho \upharpoonright t' + t = \mathcal{F}_T[\![P]\!]\rho[\rho[\![X]\!] \upharpoonright t'/X] \upharpoonright t' + t\end{aligned}$$

Suppose that $(s, \aleph)$ is an element of $\mathcal{F}_T[\![a \xrightarrow{t_0} P]\!]\rho \upharpoonright t' + t + t_0$. From the semantics of the delayed prefix operator, we may infer that $end(s, \aleph) \leqslant t' + t + t_0$ and

$$\begin{aligned}&s = \langle\rangle \wedge a \notin \sigma(\aleph) \\ &\vee \\ &\exists\, s' \bullet s = \langle(t'', a)\rangle ^\frown s' \wedge \\ &\qquad t'' \geqslant 0 \wedge a \notin \sigma(\aleph \upharpoonright t'') \wedge \\ &\qquad (s', \aleph) - (t'' + t_0) \in \mathcal{F}_T[\![P]\!]\rho\}\end{aligned}$$

Timed traces are sequences of timed events arranged in chronological order; it is a simple matter to establish that

$$\begin{aligned}&end(s, \aleph) \leqslant (t' + t + t_0) \wedge t'' \geqslant 0 \wedge s = \langle(t'', a)\rangle ^\frown s' \\ &\qquad\qquad \Rightarrow \quad end(s', \aleph) - (t'' + t_0) \leqslant t' + t\end{aligned}$$

From our assumption that P is t-constructive for X, we may infer that

$$\begin{aligned}(s', \aleph) - (t'' + t_0) \in \mathcal{F}_T[\![P]\!]\rho \upharpoonright t' + t \\ \Rightarrow \quad (s', \aleph) - (t'' + t_0) \in \mathcal{F}_T[\![P]\!]\rho[\rho[\![X]\!] \upharpoonright t'/X] \upharpoonright t' + t\end{aligned}$$

By the semantics of the prefix operator, this is equivalent to

$$\begin{aligned}& a = \langle\rangle \wedge a \notin \sigma(\aleph) \\ & \vee \\ & \exists\, s' \bullet s = \langle(t'', a)\rangle^\frown s' \wedge \\ & \qquad t'' \geqslant 0 \wedge a \notin \sigma(\aleph \upharpoonright t'') \wedge \\ & \qquad (s', \aleph) - (t'' + t_0) \in \mathcal{F}_T[\![P]\!]\rho[\rho[\![X]\!] \upharpoonright t'/X]\end{aligned}$$

which is true if and only if

$$(s, \aleph) \quad \in \quad \mathcal{F}_T[\![a \xrightarrow{t_0}]\!]\rho[\rho[\![X]\!] \upharpoonright t'/X]$$

This argument may be reversed to establish that

$$\mathcal{F}_T[\![P]\!]\rho \upharpoonright t' + t + t_0 = \mathcal{F}_T[\![P]\!]\rho[\rho[\![X]\!] \upharpoonright t'/X] \upharpoonright t' + t + t_0$$

for any time t' and environment ρ. The term $a \xrightarrow{t_0} P$ is thus $t + t_0$-constructive for X whenever P is t-constructive for X. The proof that $WAIT\ t_0\ ;\ P$ is also $t + t_0$-constructive is entirely similar. □

A.2 The Finite Dependency Theorem

To establish the Finite Dependency Theorem, we will establish a stronger result by structural induction on the syntax of Timed CSP. We recall the statement of the theorem:

Finite Dependency Theorem If P is a $TCSP$ term, possibly containing free occurrences of process variables drawn from the set $\{X_i \mid i \in I\}$, and ρ is an environment, then

$$\begin{aligned}(s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho \quad \Rightarrow \quad \exists\, N : \mathbb{F}\, I \bullet \forall\, \rho' : ENV \bullet \\ (\forall\, i : N \bullet \rho[\![X_i]\!] = \rho'[\![X_i]\!]) \Rightarrow (s, \aleph) \in \mathcal{F}_T[\![P]\!]\rho'\end{aligned}$$

◇

It is clear that this is a consequence of the following lemma, which will be established by structural induction on the syntax of Timed CSP terms:

Lemma A.1 If $(s_0, \aleph_0)$ is an element of $\mathcal{F}_T[\![P]\!]\rho$, then

$$\exists M \in \mathbb{F}(VAR \times TF) \;\bullet\; PROP(M, s_0, \aleph_0, P, \rho)$$

where

$$\begin{array}{rcl} PROP(M, s_0, \aleph_0, P, \rho) & \widehat{=} & (\forall (X,(s,\aleph)) \in M \bullet (s,\aleph) \in \rho[\![X]\!]) \\ & & \wedge \\ & & \forall \rho' \in ENV \bullet (\forall (X,(s,\aleph)) \in M \bullet (s,\aleph) \in \rho'[\![X]\!]) \\ & & \qquad \Rightarrow (s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho' \end{array}$$

♡

That is, we may find a *finite* set M of $(variable, behaviour)$ pairs such that the behaviour $(s_0, \aleph_0)$ depends only upon the elements of M. In the proof of this result, the following result will be useful. It states that if we can find a finite set M such that $PROP$ holds for M, then we can obtain a second set M' in which all of the behaviours corresponding to variables for which term P is t-constructive end at least t before $(s_0, \aleph_0)$. We establish this secondary result by showing that $PROP$ holds of the subset of M obtained by discarding those behaviours which do not meet this condition.

Lemma A.2 If $PROP$ is as defined in lemma A.1, then

$$\begin{array}{rcl} PROP(M, s_0, \aleph_0, P, \rho) & \Rightarrow & \exists M' \bullet PROP(M', s_0, \aleph_0, P, \rho) \\ & & \wedge \\ & & \forall (X,(s,\aleph)) \in M' \bullet (P \; t_X\text{-constructive for } X \\ & & \qquad \Rightarrow end(s,\aleph) + t_X \leqslant end(s_0,\aleph_0)) \end{array}$$

♡

To see that this is true, define

$$M' \mathrel{\widehat{=}} \{(X,(s,\aleph)) \in M \mid P \; t_X\text{-constructive for } X \Rightarrow end(s,\aleph) + t \leqslant end(s_0,\aleph_0)\}$$

and observe that $PROP(M', s_0, \aleph_0, P, \rho)$ holds. The first conjunct is immediate, as $M' \subseteq M$. To establish the second, let ρ' be such that

$$\forall (X,(s,\aleph)) \in M' \bullet (s.\aleph) \in \rho'[\![X]\!]$$

Then define ρ'' by

$$\rho''[\![X]\!] \;\; \widehat{=} \;\; \rho'[\![X]\!] \cup \{(s,\aleph) \mid (X,(s,\aleph)) \in M\}$$

In this case,

$$\forall (X, (s, \aleph)) \in M \bullet (s, \aleph) \in \rho''[\![X]\!]$$

and hence

$$(s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho''$$

If we choose $t_0 = end(s_0, \aleph_0)$, we obtain

$$(s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho'' \upharpoonright t_0$$

From this we may obtain

$$\begin{aligned}(s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho''[\rho''[\![X]\!] \upharpoonright t_0 - t_X / X]\\ \Rightarrow \quad (s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho'[\rho'[\![X]\!] \upharpoonright t_0 - t_X / X]\\ \Rightarrow \quad (s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho'\end{aligned}$$

The final implication above follows from the definition of t-constructive. We may now proceed to establish lemma A.1.

Proof of lemma A.1

We proceed by structural induction upon the syntax of *TCSP* terms, observing that the result is trivially true for all closed terms or processes—these have the same semantics in every environment, and the empty set is a suitable choice for set M. The remaining base case for our induction is the variable clause:

case X

Suppose that $(s_0, \aleph_0)$ is an element of $\rho[\![X]\!]$, and choose M to be the singleton set $\{(X, (s_0, \aleph_0))\}$. The result follows immediately.

The inductive step is straightforward in every case except that of mutual recursion; a typical example is the case of the parallel operator.

case $P \parallel Q$

Suppose that $(s_0, \aleph_0)$ is an element of $\mathcal{F}_T[\![P \parallel Q]\!]\rho$. From the semantics of the parallel operator we obtain that

$$\begin{aligned}\exists \aleph_P, \aleph_Q \bullet \quad & \aleph_0 = \aleph_P \cup \aleph_Q\\ & \wedge (s_0, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho\\ & \wedge (s_0, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho\end{aligned}$$

By our inductive hypothesis, there exist sets M_P and M_Q corresponding to $(s_0, \aleph_P)$ and $(s_0, \aleph_Q)$ to satisfy the proposition. If we take M to be the uuion of these sets, then we have that

$$\begin{aligned} (X,(s,\aleph)) \in M &\Rightarrow (X,(s,\aleph)) \in M_P \vee (X,(s,\aleph)) \in M_Q \\ &\Rightarrow (s,\aleph) \in \rho[\![X]\!] \vee (s,\aleph) \in \rho[\![X]\!] \end{aligned}$$

which establishes that M satisfies the first requirement. Now suppose that ρ' is such that

$$\forall (X,(s,\aleph)) : M \bullet (s,\aleph) \in \rho'[\![X]\!]$$

then, as $M_P \subseteq M$,

$$\forall (X,(s,\aleph)) : M_P \bullet (s,\aleph) \in \rho'[\![X]\!]$$

we apply the inductive hypothesis and deduce that $(s_0, \aleph_P) \in \mathcal{F}_T[\![P]\!]\rho'$. Similarly, we may deduce that $(s_0, \aleph_Q) \in \mathcal{F}_T[\![Q]\!]\rho'$. From the semantics of the parallel operator, we have that $(s_0, \aleph_P \cup \aleph_Q)$ in $\mathcal{F}_T[\![P \parallel Q]\!]\rho'$, which establishes the case.

case $\langle X_i = P_i \rangle_j$

Consider $(s_0, \aleph_0)$ in $\mathcal{F}_T[\![\langle X_i = P_i \rangle_j]\!]\rho$, where the recursive equations are indexed by set I. Unfolding the recursion, we see that

$$(s_0, \aleph_0) \in \mathcal{F}_T[\![P_j]\!]\rho_I$$

where

$$\rho_I = \rho[\mathcal{F}_T[\![\langle X_i = P_i \rangle_k]\!]\rho / X_k \mid k \in I]$$

Applying the inductive hypothesis to every term P_l, we know that for any $(s_l, \aleph_l)$ in $\mathcal{F}_T[\![P_l]\!]\rho$ there is a corresponding set $M(s_l, \aleph_l, l)$ such that

$$\begin{aligned} (i) \quad & \forall (X,(s,\aleph)) : M(s_l, \aleph_l, l) \bullet (s,\aleph) \in \rho[X] \\ (ii) \quad & \forall \rho' : ENV \bullet (\forall (X,(s,\aleph)) : M(s_l, \aleph_l, l) \bullet (s,\aleph) \in \rho'[\![X]\!]) \\ & \qquad \Rightarrow (s_l, \aleph_l) \in \mathcal{F}_T[\![P_l]\!]\rho' \end{aligned}$$

Applying lemma A.2, we obtain that there exists a set $M'(s_l, \aleph_l, l)$, a subset of $M(s_l, \aleph_l, l)$, satisfying (i) and (ii) above, such that

$$(iii) \quad \forall (X,(s,\aleph)) \in M' \bullet P \ t_X\text{-constructive for } X \Rightarrow end(s,\aleph) + t_X \leqslant end(s_0, \aleph_0)$$

We define a function $m : (VAR \times TF) \rightarrow \mathbf{P}(VAR \times TF)$ as follows:

$$m(X,(s,\aleph)) \mathrel{\widehat{=}} \begin{array}{ll} \{\} & \text{if } X \notin \{X_i \mid i \in I\} \\ M'(s,\aleph,l) & \text{if } X = X_l \wedge l \in I \end{array}$$

In the second case, if the variable X appears in the variable vector $\underline{X}$, we let $m(X,(s,\aleph))$ be the set whose existence is guaranteed by the inductive hypothesis applied to the corresponding term. We define a relation R on $VAR \times TF$ by

$$(X_1,(s_1,\aleph_1))\ R\ (X_2,(s_2,\aleph_2)) \ \Leftrightarrow\ (X_1,(s_1,\aleph_1)) \in m(X_2,(s_2,\aleph_2))$$

This is a well-founded finite-to-one relation. That is:

1. there are no infinite chains $\{C_n\}$ such that $\forall n \bullet C_{n+1}\ R\ C_n$
2. for any C, the set $\{C' \mid C'\ R\ C\}$ is finite

The second of these requirements follows immediately from the definition of m, and the first is established as follows:

Suppose that $\{(X_n,(s_n,\aleph_n)) \mid n : \mathbf{N}\}$ is such a infinite chain, then each X_n must be an X_i for some $i \in I$, for otherwise X_{n+1} cannot exist (by the definitions of R and m). For each index n, let i_n be the vector index such that $X_n = X_{i_n}$. Construct an infinite chain of natural numbers N_p by

$$\begin{array}{rcl} N_0 & \mathrel{\widehat{=}} & 0 \\ N_{p+1} & \mathrel{\widehat{=}} & min\{n : \mathbf{N} \mid n > N_p \wedge i_n \not\prec i_{n-1}\} \end{array}$$

that is, the successor N_{p+1} is defined to be the least number n greater than N_p for which the vector index i_n is not beneath i_{n-1} in the well-ordering of the vector indexing set I. This is a good definition: if the defining set is empty for N_{p+1}, the infinite sequence $\{i_k \mid k > N_p\}$ is strictly decreasing with respect to well-order $\prec$, forcing a contradiction. Let

$$t_k \mathrel{\widehat{=}} end(s_k,\aleph_k)$$

Recalling that property (iii) holds of M', which is used to define m, we have

$$\begin{array}{l} \forall p \bullet t_{p+1} \leqslant t_p \\ \forall p \bullet t_{N_{p+1}} + t \leqslant t_{N_{p+1}-1} \end{array}$$

hence

$$\forall p \bullet t_{N_{p+1}} + t \leqslant t_{N_p}$$

and thus t_{N_p} is a sequence tending to $-\infty$, contradicting the fact that each t_k is non-negative. Hence there can be no infinite chain C_n, and the relation R is indeed well-founded. We appeal to the following result from [Enderton 77]:

Kōnig's Lemma If R is a well-founded relation such that, for all y, the set $\{x \mid xRy\}$ is finite, then

$$\forall y \bullet \{x \mid x\ R^t\ y\} \text{ is finite}$$

$\diamond$

Applying this, the set $M = \{C \mid C\ R^t\ (X_j, (s_0, \aleph_0))\}$ is a finite set. We claim that

$$PROP(M, s_0, \aleph_0, \langle X_i = P_i\rangle_j, \rho)$$

Recall that

$$\begin{aligned} PROP(M, s_0, \aleph_0, P, \rho) \ \widehat{=}\ & (\forall (X, (s, \aleph)) \in M \bullet (s, \aleph) \in \rho[\![X]\!]) \\ & \wedge \\ & \forall \rho' \in ENV \bullet (\forall (X, (s, \aleph)) \in M \bullet (s, \aleph) \in \rho'[\![X]\!]) \\ & \qquad \Rightarrow (s_0, \aleph_0) \in \mathcal{F}_T[\![P]\!]\rho' \end{aligned}$$

and observe that

$$\begin{aligned} (X, (s, \aleph)) \in M \ &\Rightarrow\ \exists X', s', \aleph' \bullet (X, (s, \aleph)) \in m(X', (s', \aleph')) \\ &\Rightarrow\ \exists l, s', \aleph' \bullet (X, (s, \aleph)) \in M(s', \aleph', l) \end{aligned}$$

The first conjunct of $PROP(M, s_0, \aleph_0, \langle X_i = P_i\rangle_j, \rho)$ follows immediately from the corresponding result for $PROP(M(s', \aleph', l), s', \aleph', P_l, \rho)$. To see that the second conjunct is true, it is enough to show that, given any $(X, (s, \aleph))$ in M,

$$(\forall (X', (s', \aleph')) \in m(X, (s, \aleph)) \bullet HYP(X', (s', \aleph'))) \ \Rightarrow\ HYP(X, (s, \aleph))$$

where

$$HYP(X, (s, \aleph)) \ \widehat{=}\ (s, \aleph) \in \rho'[\mathcal{F}_T[\![\langle X_i = P_i\rangle_k]\!]\rho'/X_k][\![X]\!]$$

We establish this as follows: assume the left-hand side of the above implication, and consider the identity of variable X. If X is an element of $\{X_i \mid i \in I\}$ then

$$\rho'[\mathcal{F}_T[\![\langle X_i = P_i\rangle_k]\!]\rho'/X_k][\![X]\!] \ =\ \rho'[\![X]\!]$$

which contains $(s, \aleph)$, by the antecedent to the second conjunct of $PROP$. Otherwise, let $X = X_l$ for $l \in I$. In this case,

$$m(X, (s, \aleph)) \ =\ M(s, \aleph, l)$$

and for each $(X', (s', \aleph'))$ in $M(s, \aleph, l)$, we have that

$$\begin{aligned} & (s', \aleph') \in \rho'[\mathcal{F}_T[\![\langle X_i = P_i\rangle_k]\!]\rho'/X_k][\![X']\!] \\ \Rightarrow\ & (s, \aleph) \in \mathcal{F}_T[\![P_l]\!]\rho'[\mathcal{F}_T[\![\langle X_i = P_i\rangle_k]\!]\rho'/X_k] \\ \Rightarrow\ & (s, \aleph) \in \mathcal{F}_T[\![\langle X_i = P_i\rangle_l]\!]\rho' \end{aligned}$$

which establishes $HYP(X,(s,\aleph))$, but then the result holds for all elements of M, in particular we have that

$$HYP(X_j,(s_0,\aleph_0))$$

which says that

$$(s_0,\aleph_0) \in \rho'[\mathcal{F}_T[\![(X_i = P_i)_k]\!]\rho'/X_k][\![X_j]\!]$$

finally yielding

$$(s_0,\aleph_0) \in \mathcal{F}_T[\![(X_i = P_i)_j]\!]\rho'$$

the consequent of the second conjunct of *PROP*. This establishes the lemma, and hence the Finite Dependency Theorem. □

A.3 The Signals Model

In section 8.1, we claimed that the signals model $TM_{\tilde{F}}$ is a complete metric space under metric $\tilde{d}$ defined by

$$\tilde{d}(S,T) \triangleq inf\{\{2^{-t} \mid S(t) = T(t)\} \cup \{1\}\}$$

where

$$S(t) \triangleq \{(s,\aleph,t') \in S \mid t' \leqslant t\}$$

As in the proof of lemma 8.5, we take two definitions from [Sutherland 75]:

Definition A.3 A sequence $\{S_n\}$ in a metric space (M,d) is a *Cauchy sequence* if given $\epsilon > 0$, there exists N snch that $d(S_n,S_m) < \epsilon$ for any $m,n > N$. ◇

Definition A.4 A metric space (M,d) is said to be *complete* iff every Cauchy sequence in (M,d) converges to a point in M. ◇

and suppose that $\{S_n\}$ is a Cauchy sequence in metric space $(TM_{\tilde{F}},\tilde{d})$, and let $\{n_i\}$ be a sequence of positive integers such that

$$\forall i \geqslant 0 \bullet i < n_i < n_{i+1}$$
$$\forall m \geqslant n \bullet \tilde{d}(S_m,S_{n_i})$$

Under metric $\tilde{d}$, the limit of sequence S_n is equal to

$$S \triangleq \bigcup_{i \geqslant 0} S_{n_i}(i)$$

By our choice of sequence n, we have that

$$0 \leqslant t \leqslant i \;\Rightarrow\; S(t) = S_{n_i}(t)$$

For each axiom ax given in section 8.1, we must show that

$$\forall n \bullet S_n \in TM_{\tilde{F}} \;\Rightarrow\; S \text{ satisfies } ax$$

We consider the case of the second axiom:

$$(s, \aleph, t) \in S \wedge t' \geqslant t \Rightarrow \exists s' \bullet \sigma(s') \subseteq \hat{\Sigma} \wedge (s ^\frown (s' + t), \aleph, t') \in S$$

Suppose that

$$(s, \aleph, t) \in S \quad \wedge \quad t' \geqslant t$$

If we choose i such that

$$S(t') \;=\; S_{n_i}(t')$$

then we may infer that

$$(s, \aleph, t) \;\in\; S_{n_i}$$

and hence that

$$\exists s' \;\bullet\; \sigma(s') \subseteq \hat{\Sigma} \wedge (s ^\frown (s' + t), \aleph, t') \in S_{n_i}$$

However

$$\begin{aligned}(s ^\frown (s' + t), \aleph, t') \in S_{n_i} &\Rightarrow (s ^\frown (s' + t), \aleph, t') \in S_{n_i}(t') \\ &\Rightarrow (s ^\frown (s' + t), \aleph, t') \in S(t') \\ &\Rightarrow (s ^\frown (s' + t), \aleph, t') \in S\end{aligned}$$

We may conclude that

$$(s, \aleph, t) \in S \wedge t' \geqslant t \Rightarrow \exists s' \bullet \sigma(s') \subseteq \hat{\Sigma} \wedge (s ^\frown (s' + t), \aleph, t') \in S$$

and hence that the limit S satisfies the axiom. Similar reasoning allows us to establish that the limit satisfies the other seven axioms, and hence that the model $TM_{\tilde{F}}$ is a complete metric space. □

Glossary

Mathematical Symbols

$\mathbb{P}$	powerset operator	$\{\}$	the empty set
$\mathbb{F}$	set of all finite subsets of	$\cong$	semantic equivalence
seq	set of all finite sequences of	$\widehat{=}$	defined to be equal to
$\mathbb{N}$	set of natural numbers	$\prec$	a partial order
$\mathbb{Z}$	set of integers	seg	initial segment
$\mathbb{Q}$	set of rational numbers	$\underline{v}$	vector v
$\mathbb{R}$	set of real numbers	$\heartsuit$	end of theorem or lemma
$m\,..\,n$	integers from m to n	$\triangle$	end of rule
dom	domain of a function	$\square$	end of proof
ran	range of a function	$\diamondsuit$	end of definition

Syntax

$\bot$	divergence	$;$	sequential composition
$STOP$	deadlock	$\overset{\circ}{;}$	sequential composition (with δ delay)
$SKIP$	successful termination	$\backslash$	hiding
$WAIT$	delayed termination	$\mu X \bullet P$	delayed recursion
$\rightarrow$	prefix	$\mu X \circ P$	immediate recursion
$\twoheadrightarrow$	instant prefix	$P[Q/X]$	syntactic substitution
$\xrightarrow{t}$	delayed prefix	$\parallel$	lockstep parallel
$\square$	deterministic choice	${}_A\parallel_B$	alphabet parallel
$\sqcap$	nondeterministic choice	$\vert\vert\vert$	interleaving
$f(P)$	direct image	$\underset{A}{\parallel}$	sharing parallel
$f^{-1}(P)$	inverse image		

$\overset{t}{\rhd}$	timeout	$\langle X_i = P_i \rangle_j$	mutual recursion
$\lightning$	untimed interrupt	CSP	untimed CSP terms
$\underset{t}{\lightning}$	timed interrupt	$TCSP$	Timed CSP terms
$\underset{e}{\nabla}$	event interrupt	Θ	syntactic abstraction

Semantics

tr	untimed trace	Σ	all events
s	timed trace	Σ^*	all untimed traces
$\aleph$	refusal set	$T\Sigma$	all timed events
α	stability value	$T\Sigma^*_{\leqslant}$	all timed traces
ρ	environment	TF	all timed failures
δ	delay constant	TS_F	all sets of timed failures
$\checkmark$	termination event	TM_F	timed failures model
ε	non-event	d	distance metric
VAR	process variables	$M(X, P)$	mapping for $\mu X \circ P$
ENV	environments	$M_\delta(X, P)$	mapping for $\mu X \bullet P$
$TIME$	the time domain $[0, \infty)$	$\rho[Y/X]$	environment over-riding
$TINT$	half-open time intervals	TS_F^I	product space
$RTOK$	refusal tokens	TM_F^I	product model
$RSET$	timed refusal sets	$\underline{d}$	vector metric

Semantic Functions and Models

$\mathcal{T}$	traces	M_T	traces model
$\mathcal{S}$	stabilities	M_S	stabilities model
$\mathcal{F}$	failures	M_F	failures model
$\mathcal{E}$	failures-stabilities	M_{FS}	failures-stabilities model
$\mathcal{T}_T$	timed traces	TM_T	timed traces model
$\mathcal{S}_T$	timed stabilities	TM_S	timed stabilities model
$\mathcal{F}_T$	timed failures	TM_F	timed failures model
$\mathcal{E}_T^*$	untimed failures-timed stabilities	TM_{FS}^*	untimed failures-timed stabilities model
$\mathcal{E}_T$	timed failures-timed stabilities	TM_{FS}	timed failures-timed stabilities model

Timed Failures

$\langle\rangle$	the empty trace	*first*	first event
$\frown$	catenation of traces	*last*	last event
$\leqslant$	trace prefix	*begin*	start time
$\dot{-}$	time shift	*end*	end time
$-$	time shift (failures)	*head*	first timed event
$\downarrow$	count of events	*foot*	last timed event
$\uparrow$	before	*times*	time values present
$\uparrow$	during	*tstrip*	strip time values
$\upharpoonleft$	after	σ	events present
$\downharpoonleft$	restrict	$\cong$	trace equivalence
$\backslash$	hiding	$CL_{\cong}$	closure under $\cong$

Specification

sat	satisfies
$\mathbf{sat}_{\rho}$	satisfies in environment ρ
Φ	abstraction mapping for trace specifications
$\mathbf{act}_{A}$	active for every event in set A
$\backslash\!\backslash$	whenever these events are active

Signals Model

$\hat{a}$	signal event	$\hat{\Sigma}$	all signal events
$\hat{\checkmark}$	termination signal	$\tilde{\Sigma}$	all events
$\hat{\sigma}$	signals present	$T\tilde{\Sigma}$	all timed events
sync	possible synchronisations	$T\tilde{\Sigma}^{*}_{\leqslant}$	all timed traces
$\widetilde{ENV}$	environments with signals	$T\tilde{F}$	all timed failures
$\tilde{d}$	metric for $TM_{\tilde{F}}$	$TS_{\tilde{F}}$	sets of timed failures
$\mathcal{F}_S$	semantic function	$TM_{\tilde{F}}$	signals model

Thanks

For inspiration, friendship, practical assistance, and legal advice during the preparation of this thesis:

Greg Abowd, Geoff Barrett, Howard Barringer, Matthew Blakstad, Meghan Burke, Mark Bush, Sue Charlett, Ching-Hua Chow, Peter Coesmans, Katie Cooke, Damian Cugley, Naiem Dathi, Will Davies, Kate Davis, Vicky Elphicke, Susan Even, Mike Field, Paul Gardiner, Dave Gavaghan, Jeremy Gibbons, Steve Giess, Michael Goldsmith, Malcolm Harper, Guy Hart-Davis, Alison Harvey, Claire Henderson, Tony Hoare, John Iwnicki, Dave Jackson, Jeremy Jacob, Alan Jeffrey, Liz Johns, Geraint Jones, Mathai Joseph, Mike Kalougin, Andrew Kay, Steve King, Alice King-Farlow, Nick Lawrence, Jo Leggett, Florence Maraninchi, David Mayers, Quentin Miller, Colin Millerchip, Charlie Morcom, David Murphy, Andrew Newman, Duncan Oliver, Monica Payne, Mike Reed, Joy Reed, Phil Richards, Gordon Riddell, Bill Roscoe, Anne Ryan, Ib Sørensen, Elizabeth Schneider, Steve Schneider, Jenni Scott, Jess Search, Karen Seidel, Julie Sheppard, Mike Spivey, Richard Stamper, Joe Stoy, Bernard Sufrin, Wilson Sutherland, Jacqui Thornton, Tim White, Rob Woiccak, Ken Wood, and Jim Woodcock.