A MATHEMATICAL THEORY OF SYNCHRONOUS COMMUNICATION

by

Janet E. Barnes

Technical Monograph PRG-112
ISBN 0902928899

Hilary Term 1993

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

A Mathematical Theory of Synchronous Communication

Janet E. Barnes

Hilary Term 1993

Abstract

A mathematical theory of synchronous communication is presented. The process algebra SCSP, shares many of its constructors with Hoare's Communicating Sequential Processes. It models components of a distributed system as processes evolving in lockstep. A synchronous variant, SRPT, of Josephs' Receptive Process Theory, which distinguishes between input and output events in its model of communication, is also investigated.

The language SCSP is given a denotational semantics. The semantic model captures the behaviour of processes using failures-divergences information. SCSP exhibits sufficient algebraic laws to form a sound and complete proof system with respect to the semantics. This allows reasoning about concurrent systems by means of algebraic manipulation of process expressions. The notation is extended to capture communication of data via channels and is used to specify a token ring protocol. SCSP is sufficiently expressive to establish temporal details of the protocol.

SRPT can be interpreted as a receptive sublanguage of SCSP. This is demonstrated by embedding both the language and its semantic model in those of SCSP. The embedding allows many of the mathematical results concerning SRPT to be deduced from their counterparts in SCSP. SRPT is shown to be applicable to the modelling of synchronous circuits. The notion of discrete time in the algebra captures the clock's behaviour while the receptive nature of SRPT matches the communication of signals in a circuit. By introducing a notion of timewise abstraction, the effect of variation in the speed at which circuits are clocked can be analysed. Timewise abstraction is also applied to the analysis of pipes.

To my Parents and Dave
for their love and support.

Acknowledgements

Firstly, I must thank my supervisor, Mark Josephs, for his encouragement, advice and guidance throughout the production of this thesis. This work has also profited from the comments and advice of John Barnes, Jim Davies, Tony Hoare, Dave Jackson, Geraint Jones, Steve Schneider, Brian Scott and other colleagues at the PRG to whom I extend my thanks.

This work was made possible by the financial support of the UK SERC for which I am grateful.

Contents

Chapter 1

Introduction

As technology advances and society places a greater reliance on computer systems in critical applications, verifying the correctness of these systems becomes more important and more difficult as the size of the systems increases. Furthermore, in an endeavour to increase efficiency more emphasis is being placed on concurrent systems which are harder to analyse than their sequential counterparts. A concurrent system can be viewed as a network of component processes interacting via some form of communication. Because of an awareness by computer scientists of the need for formal design and verification techniques for such systems, the last decade has seen the development of mathematical models (including so-called process algebras) of communication and concurrency which can be used to analyse the behaviour of networks of processes. Well-known process algebras are Hoare's CSP [Hoa85], Milner's CCS [Mil89] and Bergstra and Klop's ACP [BK84]. The usual approach of these algebraic methods is to view the system at an appropriate level of abstraction at which only key events are observed. At this level the system is described in an equational form which, by the use of algebraic laws, can be manipulated to give information concerning the interaction of these events.

The original process algebras do not display any concept of quantitative time; they restrict their concerns to the ordering of events. This makes them unsuitable for examining systems in which timing is critical, for example a nuclear reactor controller must insert the control rods within time t once an overtemperature signal has been detected, where t is typically a very short time. There are also situations in which systems, without time critical requirements, may be more satisfactorily modelled in a timed framework. In an idle token ring the token passes unhindered around the ring; there is no guarantee of the availability of a message for transmission around the ring and any model must be able to represent this situation. Consequently in an untimed model it may be difficult to hide the mechanism of the protocol (the token) while avoiding infinite chatter (the possibility of an arbitrary number of internal events occurring). In a timed model this problem can be

avoided by assuming that the token takes time (which cannot be hidden) to pass around the ring.

In order to extend the domain of problems which can be satisfactorily addressed by process algebra techniques, there has been interest in extending the original algebras to incorporate the concept of time. Both dense and discrete time domains have been considered for the measure of time. Dense time models typically use the real numbers, $\mathbb{R}$, as their time domain and view the passage of time as continuous. Dense time models include Reed and Roscoe's Timed CSP [DS89, RR86], Moller and Tofts' Temporal CCS [MT90] and Baeten and Bergstra's Timed ACP [BB91a]. Discrete time models typically use the natural numbers, $\mathbb{N}$, as their time domain and assume that time increases in a stepwise fashion, each step corresponding to the 'tick' of a global clock. Discrete time models include Milner's Synchronous CCS [Mil83], Hennessy and Regan's TPL [HR90], and Jeffrey's Discrete Timed CSP [Jef91a].

This thesis is devoted to the development of a new discrete time model, Synchronous CSP, and its 'receptive' submodel [Dil89, Jos92]. It is hoped that these process algebras will be widely applicable; case studies in protocol verification and digital logic design are described in this thesis.

1.1 Technical Overview

1.1.1 Synchronous CSP

The next chapter presents a discrete time algebra, Synchronous CSP. Whereas CCS and SCCS are based on an operational semantics, CSP and SCSP are based on a denotational semantics. In SCSP, the underlying model is a failures-divergences model [BR85] in which time is recorded implicitly. The failures have been encoded as traces (finite sequences) of sets, these sets consisting of occurrences and refusals of events. Divergences are also encoded in the trace sets as in [Jos92], giving us a very simple model.

In designing the language SCSP, the aim has been to achieve a formalism which is sufficiently expressive to capture the time dimension of systems under analysis, while at the same time maintaining a powerful algebraic structure. By choosing a discrete model of time we limit the applications of our algebra. It only applies to systems where a lowest common denominator on the delays between observations can be postulated. This is less restrictive than one might imagine; in fact for a majority of systems a discrete time model is the natural choice. This is particularly so in hardware which is often clocked at a speed which allows individual components to reach a stable state. In applications where the concept of discrete time does indeed suffice, the advantages of using an algebra which supports the same level of expressibility can be extensive. By using a discrete time model we are able to

obtain a complete axiomatisation within the algebra; this is not the case in the current models of Timed CSP, for example. Such an axiomatisation makes the algebra a powerful specification and design language.

Although many of the constructs within the algebra are constructs familiar to CSP, SCSP should not be regarded as a syntactic extension of CSP. Instead, advantage has been taken of the time dimension in setting up the model, while keeping the model close enough to that of CSP to draw from the wealth of experience that has been generated by the latter.

In SCSP, we assume that events which occur simultaneously are independent. This is a fairly natural assumption since it merely assumes that time must pass between cause and effect. We express the simultaneous occurrence of events by the use of sets in the traces, rather than artificially imposing an ordering on events occurring simultaneously. This resembles the 'true concurrency' approach and contrasts with the 'interleaving concurrency' approach of many other process algebras.

A second assumption of our model is that a given event can occur at most once at each time step. (We are not interested in pathological cases, such as that of an infinite number of occurrences of an event at a given time which Jeffrey finds himself considering [Jef91a]. Indeed, we restrict ourselves to finite alphabets of events and so we can be sure that only a finite number of events occur at any given time.) Again the assumption is less restrictive than it might first appear. For example, the voltage-level on a wire may change many times between clock pulses, but provided it always stabilises before the rising edge of the clock, say, at most one event (indicating the final voltage level associated with that wire) need be recorded on each tick.

1.1.2 Synchronous Receptive Process Theory

In the latter half of this thesis a second algebra, Synchronous Receptive Process Theory is presented. There are strong similarities between the design principles of this language and SCSP; namely SRPT is based on a discrete time frame, simultaneously occurring events are assumed to possess causal independence and simultaneous multiple occurrences of the same event are prohibited in the language.

The main difference between SCSP and SRPT is the assumed method of communication between processes and their environment. In SCSP we assume that events occur only on simultaneous co-operation of the process and its environment. In SRPT events performed by a process are classified into two types, input events and output events. The process is always receptive to input events, in that it is always willing to co-operate with the environment on the performance of input events. Symmetrically the environment is always prepared to allow the performance of output events from the process. The process has complete control over the performance of output events, while the environment has complete control over

the performance of input events. Naturally the behaviour of a process is influenced by the input events performed, so input to a process can affect subsequent output by that process. The method of communication adopted by SRPT, coupled with the choice of time frame, make SRPT particularly appropriate for problems which involve modelling clocked circuits.

Like SCSP, SRPT is based on a denotational semantics. The underlying model for SRPT is very simple: by distinguishing between input and output events in the model and considering the receptive nature of communication, it is no longer necessary to record refusal information. It is sufficient to consider a traces-divergences model in which time is recorded implicitly. Terms in the traces are sets of events. Divergences are again encoded in the traces, although a slightly different approach to this encoding results in us considering a different partial order on the model to the usual non-determinism ordering used in CSP [BR85] and SCSP.

The similarities in the design principles of SCSP and SRPT are borne out in our ability to embed SRPT and its associated model into SCSP and its associated model. The embedding demonstrates how SRPT can be viewed as a receptive sublanguage of SCSP. Moreover, many of the theoretical results of SRPT can be deduced via the embedding and corresponding results for SCSP.

A final consequence of the combination of the receptive model of communication and the discrete time frame in SRPT is the ability to perform timewise abstraction. Timewise abstraction provides a method for translating the time frame in which a system is represented. This can be particularly useful when the internal workings of components of a system are most appropriately verified in a time frame different to that appropriate to the ultimate interaction between the system and its environment. Timewise abstraction can be viewed as the time dimensional counterpart to communication abstraction which is already recognised as a powerful development tool.

1.2 Structure of this Thesis

This thesis can be divided into two parts. The first part consists of Chapters 2–4 and is concerned with the discrete time process algebra, SCSP. Chapter 2 introduces the language SCSP; each of the operators of the language is described, these descriptions are supported by axioms satisfied by processes in SCSP and some examples. A mathematical theory underpinning the algebra of SCSP is developed in Chapter 3; a failures-divergences model representing the behaviours of processes is presented, this enables the construction of a denotational semantics for SCSP. A proof system for SCSP, incorporating the axioms presented in Chapter 2, is shown to be sound and complete with respect to the semantics in Chapter 3. In Chapter 4 the language is enhanced to allow value passing in communication, this

enhancement is used in the specification of a token ring protocol in SCSP.

The second half of this thesis, Chapters 5–7, presents a synchronous receptive process theory (SRPT) and considers some of the features of this algebra. In Chapter 5 the language SRPT and its associated denotational model are presented and informally compared with those of SCSP. The comparison of SRPT with SCSP is formalised in Chapter 6 where, by embedding both the language and model of SRPT into that of SCSP, it is demonstrated how SRPT can be viewed as a receptive sublanguage of SCSP. The embedding makes it possible to deduce a sound and complete proof system for SRPT from the results of SCSP. The theory of timewise abstraction and its application to SRPT are considered in Chapter 7, the use of timewise abstraction is supported by several examples drawn from the field of digital logic design.

The final chapter outlines the main results of this thesis, makes comparisons with work by other authors and concludes with a discussion of possible future developments resulting from this work.

Chapter 2

Synchronous CSP

Like Hoare's CSP [Hoa85], Synchronous CSP is designed to provide a clear model for systems of concurrent processes interacting via synchronised communication. Moreover, SCSP allows us to model timing conditions by assuming that events occur at discrete points in time. It provides an algebra and associated algebraic laws allowing us to manipulate expressions into forms which make the consequences of the interaction explicit. Non-determinism and concurrency are handled in a manner familiar from CSP, although other constructs have been superseded by constructs which enable us to grasp better the considerations which are our concern: namely our implicit measure of absolute time and our true concurrency approach to events which occur simultaneously.

We shall see in this chapter that Synchronous CSP is a simple language which is, nevertheless, sufficiently expressive to capture the characteristics of many systems. The algebra and laws are underpinned by the denotational model which will be presented in Chapter 3.

2.1 The Language

As in Hoare's CSP, we assume that the system we wish to model can be viewed as performing, in cooperation with the environment, a selection of instantaneous events. We take the environment to be all components which may interact with the system under scrutiny. Such components will typically be other systems or a user. We choose to insist that events have no duration; actions with duration can be represented by two events symbolising the commencement and termination of the action.

We presuppose a universal alphabet of events Σ. We associate with each process an alphabet of events, $A \subseteq \Sigma$, in which it may participate. We require that A is finite and non-empty. We also presuppose a set of process variables, Var. These variables facilitate the definition of processes by recursion.

The abstract syntax of our language is similar to a subset of CSP, the obvious difference being the replacement of the event prefix construct by the set prefix construct. We take P to range over process terms*, $A \in \mathbb{F}\Sigma$, $x \in Var$ and S to range over bijective renaming functions $S : \Sigma \to \Sigma$. Then, with certain restrictions on the alphabets of the processes, the following grammar defines the syntax of our language.

$P ::=$	$\bot_A$	Chaos
	$\mid x$	process variable
	$\mid P \sqcap P$	non-deterministic choice
	$\mid [X \subseteq A \to P_X]$	set prefix
	$\mid P \parallel P$	parallel composition
	$\mid P \setminus A$	hiding
	$\mid P[S]$	renaming
	$\mid \mu\, x : A \bullet P$	recursion
	$\mid \langle x_i \mathrel{\widehat{=}} P_i \rangle_j$ with A	mutual recursion

We now consider the informal interpretation of each of these terms along with restrictions imposed upon the alphabets of the process terms.

2.1.1 Primitive processes and operators

In presenting the operators in the following sections we shall state various equational properties of these operators expressed in the form $P \equiv Q$ ($P \mathrel{\widehat{=}} Q$ means, on the other hand, P is defined equal to Q). When the equational property is stated as an axiom, it is an axiom of the proof system to be presented in Section 3.4. When the equational property is stated as a law then it is derivable within the proof system - the derivation of many of these laws is however tedious involving reduction and comparison of the terms on both sides of the equation, so will not be presented here.

Chaos

The process $\bot_A$ is the most undesirable process with alphabet A; it can arbitrarily mimic the behaviour of any other process with the same alphabet. Such a process is said to be *divergent*. It is used to model behaviour when things go wrong, it is a worst case scenario and we assume there is no escape from this erroneous behaviour. Often the alphabet will be clear from the context and we will simply write $\bot$.

*$\mathbb{F}\, X$ denotes the set of finite subsets of X while $\mathbb{P}\, X$ denotes the power set of X.

Process variable

The inclusion of process variables within the syntax allows a proper treatment of recursion. The term $x \in Var$ represents the process bound to variable x in the context of a particular choice of variable bindings. It is necessary to make explicit the choice of variable bindings before we can make any deductions about the process to which x is bound.

Non-deterministic choice

When two processes P and Q have a common alphabet A, we define the non-deterministic choice between these processes, $P \sqcap Q$, to be the process with alphabet A which non-deterministically behaves like P or like Q. As with CSP, this choice can be viewed as occurring internally within the system; the environment has no control over the outcome of the choice.

Non-deterministic choice satisfies the following axioms:

A–1 : $P \sqcap Q \equiv Q \sqcap P$

A–2 : $(P \sqcap Q) \sqcap R \equiv P \sqcap (Q \sqcap R)$

A–3 : $P \sqcap P \equiv P$

A–4 : $P \sqcap \bot \equiv \bot$

Axiom 4 reflects the observation that $\bot$ can mimic the behaviour of any process; in particular, we cannot distinguish between a choice in favour of P and $\bot$ arbitrarily mimicking the behaviour of P.

Set prefix

A choice set B is a subset of an alphabet A. Let P be a $\mathbb{P}(B)$-indexed family of processes, each with alphabet A. The process $[X \subseteq B \rightarrow P_X]$ can perform the events in any subset, C, of B at the first time step and then go on to behave like P_C. None of the events in B can initially be refused by the process, so the largest subset of events from B offered by the environment will he performed by the process. The process is initially unable to perform events not present in B. This process has a built in time-out behaviour in that, if the environment is not initially willing to offer any events in B, then the process will 'time-out' and then behave like $P_{\{\}}$.

Consider the process

$$[X \subseteq \{a, b\} \rightarrow (\bot \textbf{ if } X = \{\} \textbf{ else } P)].$$

Initially it is able to perform events a or b or both, but cannot perform any other events in its alphabet. If the environment is able to participate in at least one of a and b then the process will evolve to P at the next time step. On the other hand, if the environment is not in a position to offer either a or b to the process, then the process will not perform an event at the first time step and will evolve to chaos at the next time step.

Set prefixing provides the only form of environmental choice in our language. This choice differs from the external choice of both untimed and timed CSP [BHR84, RR86]. Unlike those models, SCSP does not support instantaneous resolution of external choice. A process cannot offer its environment the choice between two events without being able to offer both together. This is a direct consequence of our assumption that events observed simultaneously occur independently; the performance of one event at a particular time cannot preempt another event at the same time.

We have one axiom involving set prefix.

A–5 : If $C \subseteq B$,

$$[X \subseteq B \to P_X] \sqcap [Y \subseteq C \to Q_Y] \equiv [X \subseteq B \to R_X] \sqcap [Y \subseteq C \to Q_Y]$$

$$\text{where } R_{B'} \mathrel{\widehat{=}} \begin{cases} P_{B'} \sqcap Q_{B'} & \text{if } B' \subseteq C \\ P_{B'} & \text{if } B' \not\subseteq C \end{cases}$$

This axiom is explained by noting that an observer can establish which way the process resolved the non-deterministic choice after the first time step exactly when at least one of the events offered by the environment is in the set $B - C$.

Consider the process

$$[X \subseteq \{a, b\} \to P] \sqcap [Y \subseteq \{a\} \to Q]$$

If the environment offers a b initially, then we can tell after the first time step whether the process will behave like P or like Q by noticing whether the b occurred or not. If the environment does not offer a b initially, then there is no way of establishing which way the choice was resolved without further observation. We can therefore postpone the resolution of choice in those circumstances when the b is not initially offered. Hence the above process is equivalent to:

$$[X \subseteq \{a, b\} \to (P \text{ if } b \in X \text{ else } P \sqcap Q)] \sqcap [Y \subseteq \{a\} \to Q]$$

In the case where $B = C$ A–5 and the idempotence of non-deterministic choice allow us to deduce the distributivity law:

L–1 : $[X \subseteq B \to P_X] \sqcap [Y \subseteq B \to Q_Y] \equiv [Z \subseteq B \to (P_Z \sqcap Q_Z)]$

It is convenient to provide an alternative notation for set prefix. Let $I = \{1..n\}$ be a finite indexing set, let B_i $(i \in I)$ be distinct finite subsets of the alphabet A, and let P_i $(i \in I)$ and Q be processes with alphabet A. Then we write

$$[B_1 \rightarrow P_1 \,\Box\, B_2 \rightarrow P_2 \,\Box\, \ldots \,\Box\, B_n \rightarrow P_n \rhd Q] \mathrel{\hat{=}} [X \subseteq \bigcup B_i \rightarrow P_X]$$

where $P_{B'} \mathrel{\hat{=}} \begin{cases} P_i & \text{if } B' = B_i \text{ and } i \in I \\ Q & \text{otherwise} \end{cases}$

Parallel composition

The parallel composition, $P \parallel Q$, of two processes P and Q is the process which results from their concurrent execution. Assuming that the alphabets of P and Q are αP and αQ respectively, then the alphabet of $P \parallel Q$ is $\alpha P \cup \alpha Q$. While neither of the component processes are divergent, synchronisation must occur on common events. Synchronisation over common events means that such events can only occur when both P and Q are prepared to perform them; they are refused if one or both of the component processes refuses them. Events not in the common alphabet can occur or be refused according to the state of the corresponding component process. Once one or other of the component processes becomes divergent so does $P \parallel Q$.

Parallel composition satisfies the following axioms:

A–6 : $\quad \bot_A \parallel P \equiv \bot_{A \cup \alpha P}$

A–7 : $\quad P \parallel \bot_A \equiv \bot_{A \cup \alpha P}$

A–8 : $\quad (P \sqcap Q) \parallel R \equiv (P \parallel R) \sqcap (Q \parallel R)$

A–9 : $\quad P \parallel (Q \sqcap R) \equiv (P \parallel Q) \sqcap (P \parallel R)$

A–10 : $\quad [X \subseteq A' \rightarrow P_X] \parallel [Y \subseteq B' \rightarrow Q_Y] \equiv$
$[Z \subseteq ((A' \cap B') \cup (A' - B) \cup (B' - A)) \rightarrow (P_{Z \cap A'} \parallel Q_{Z \cap B'})]$
where $[X \subseteq A' \rightarrow P_X]$ and $[Y \subseteq B' \rightarrow Q_Y]$ have alphabets A and B.

Commutativity and associativity of parallel composition can be deduced within the proof system. The proof of L–2 is given as Theorem B.1 in Appendix B.

L–2 : $\quad P \parallel Q \equiv Q \parallel P$

L–3 : $\quad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$

Hiding

It is often useful to be able to change the level of abstraction of a problem by hiding events from the environment. For example, when building a model of a circuit we may develop subcomponents, take their parallel composition and finally hide all communication on internal wires between subcomponents. The set of events to be hidden, B, does not include the whole alphabet of the process P. A hidden event occurs as soon as the process is ready; the environment plays no part. As the passage of time cannot be hidden and events are assumed to occur no more than once at each tick of the global clock, 'infinite chatter' cannot arise over a finite time span.

Hiding satisfies the following axioms:

A–11 : $\quad \bot_A \setminus B \equiv \bot_{A-B}$

A–12 : $\quad (P \sqcap Q) \setminus A \equiv (P \setminus A) \sqcap (Q \setminus A)$

A–13 : $\quad [X \subseteq B \rightarrow P_X] \setminus A \equiv [Y \subseteq (B - A) \rightarrow (P_{Y \cup (B \cap A)} \setminus A)]$

and the following laws:

L–4 : $\quad (P \setminus A) \setminus B \equiv P \setminus (A \cup B)$

L–5 : $\quad (P \parallel Q) \setminus A' \equiv (P \setminus A') \parallel (Q \setminus A')$
if $A' \cap \alpha P \cap \alpha Q = \{\}$

Renaming

Renaming facilitates reuse of components. It is often the case that two processes in a system can be viewed as isomorphic in that their behaviour is the same up to some relabeling of events. Restricting ourselves to a bijective renaming function $S : \Sigma \rightarrow \Sigma$, we denote $P[S]$ to be a renaming of process P. If process P has alphabet A then process $P[S]$ has alphabet $A[S] \mathrel{\hat{=}} \{S(a) \mid a \in A\}$. $P[S]$ performs event $S(a)$ in exactly the circumstances that P would perform event a.

Renaming satisfies the following axioms:

A–14 : $\quad \bot_A [S] \equiv \bot_{A[S]}$

A–15 : $\quad (P \sqcap Q)[S] \equiv P[S] \sqcap Q[S]$

A–16 : $\quad [X \subseteq B \rightarrow P_X][S] \equiv [X \subseteq B[S] \rightarrow P_{X[S^{-1}][S]}]$

and the following laws:

L–6 : $\quad P[S][R] \equiv P[R \cdot S]$

L–7 : $\quad (P \parallel Q)[S] \equiv P[S] \parallel Q[S]$

L–8 : $\quad (P \setminus B)[S] \equiv P[S] \setminus B[S]$

2.1.2 Recursion

Until now we have only provided operators suitable for expressing finite processes. Any process which can be expressed using the operators developed so far will, after a finite time, behave like our basic process $\perp$.

$\mu x : A \bullet P$ represents the solution of the recursive definition of the process x defined as a particular (least) fixed point of the function $\lambda x \bullet P$. The mathematical details of this construction will be presented in a later section.

A–17 : $\quad \mu x : A \bullet P \equiv P[(\mu x : A \bullet P)/x]$

Here $P[(\mu x : A \bullet P)/x]$ denotes the process P with $\mu x : A \bullet P$ substitnted for every free occurrence of the variable x. (This syntactic substitution $[E/x]$ can always be distinguished from renaming $[S]$ by its context.) Recursion also satisfies alpha conversion:

L–9 : $\quad \mu x : A \bullet P \equiv \mu y : A \bullet P[y/x] \qquad$ where y is not free in P.

Uniqueness of fixed points

Following Brookes, Hoare and Roscoe, [BHR84], we define $P \downarrow n$ to be the process which behaves like P for the first n steps and then becomes chaotic, behaving like $\perp$. A function, F, from process terms to process terms is said to be *constructive* if the first $n+1$ steps of the behaviour of process $F(P)$, for an arbitrary process P, are only dependent on the first n steps of process P's behaviour; i.e.

$$F(P) \downarrow (n+1) = F(P \downarrow n) \downarrow (n+1) \qquad \text{for all } P.$$

Similarly F is said to be a *non-destructive* function if the first n steps of the behaviour of a process $F(P)$, are only dependent on the first n steps of P's behaviour; i.e.

$$F(P) \downarrow n = F(P \downarrow n) \downarrow n \qquad \text{for all } P.$$

These definitions can be extended to functions with more that one argument and a function can be described as being constructive or non-destructive in some or all of its arguments.

All the primitive operators developed here are non-destructive and $[X \subseteq B \rightarrow .]$ is constructive in all its arguments. It is a simple exercise to show that the composition of two non-destructive functions is non-destructive, while the composition of a constructive and non-destructive function is constructive. So any $\lambda x \bullet P$ with P a process term is non-destructive. Moreover if every occurrence of the free variable x in P is directly or indirectly guarded by a set prefix, then $\lambda x \bullet P$ is constructive in x. Formally:

Definition 2.1 We say that a process term P is guarded in x if one of the following conditions is met:

1. x does not occur free in P.
2. $P = y$ for some process variable $y \neq x$ and y is bound to a process which is guarded in x.
3. $P = [X \subseteq B \rightarrow P_X]$ for some subset, B, of the alphabet.
4. $P = Q \parallel R$, where Q and R are guarded in x.
5. $P = Q \sqcap R$, where Q and R are guarded in x.
6. $P = Q \setminus A$, where Q is guarded in x.
7. $P = Q[S]$ where Q is guarded in x.
8. $P = \mu\, y : A \bullet Q$, where Q is guarded in x.
9. $P = \langle x_i \mathrel{\hat{=}} Q_i \rangle_j$ with A, where Q_j is guarded in x. (Mutual recursion is defined below.)

$\Diamond$

Note that, unlike CSP, SCSP gives $P \setminus A$ guarded if P is guarded.

It is a trivial consequence of the above definition that P is guarded in x implies $\lambda x \bullet P$ is constructive in x.

It follows from the argument set out in [BHR84] that if P is guarded in x then $\mu x : A \bullet P$ is the unique fixed point of $\lambda x \bullet P$ in an appropriate partial order.

Mutual recursion

Let Λ be a finite totally ordered indexing set and let i, j range over Λ. If, for all $i \in \Lambda$, P_i is guarded in x_j for all $j \leq i$ then

$$\langle x_i \mathrel{\hat{=}} P_i \rangle_j \text{ with } A \qquad i \in \Lambda$$

represents the j^{th} component of the solution of the recursive definition of the vector of processes $\langle x_i \rangle$ defined as a particular (unique fixed point) solution to the equation $\langle x_i \mathrel{\hat{=}} P_i \rangle$. In a manner similar to the case for simple recursion we have the following axiom.

A–18 : $(\langle x_i \mathrel{\hat{=}} P_i \rangle_j \text{ with } A) \equiv P_j[(\langle x_i \mathrel{\hat{=}} P_i \rangle_k \text{ with } A)/x_k]$
where k ranges over the x_k free in P_j.

and alpha conversion:

L–10 : $(\langle x_i \mathrel{\hat{=}} P_i \rangle_j \text{ with } A) \equiv (\langle y_i \mathrel{\hat{=}} P_i[y_k/x_k] \rangle_j \text{ with } A)$
where no y_k is free in any P_i and k ranges over Λ.

2.1.3 Derived processes and operators

The following processes and operators can be constructed from those introduced in the previous sections. They are presented separately here since they are useful in their own right.

Wait

Wait is simply a special case of set prefixing. It may be that a process waits for a number of units of time, unable to engage in any actions, and then behaves like process P. We denote such a process by $wait(n) \rightarrow P$. We can define the wait prefix in terms of set prefixing with $\{\}$ as follows:

$$\begin{array}{rcl} wait(0) \rightarrow P & \widehat{=} & P \\ wait(n+1) \rightarrow P & \widehat{=} & [X \subseteq \{\} \rightarrow (wait(n) \rightarrow P)] \end{array}$$

L–11 : $wait(m) \rightarrow (wait(n) \rightarrow P) \equiv wait(n+m) \rightarrow P$

Proof: by induction on m.

Base case $m = 0$ is trivial by definition of *wait*.

Inductive step.

$$\begin{array}{ll} & wait(m+1) \rightarrow (wait(n) \rightarrow P) \\ \equiv & \{ \text{ defn. of wait } \} \\ & [X \subseteq \{\} \rightarrow (wait(m) \rightarrow (wait(n) \rightarrow P))] \\ \equiv & \{ \text{ inductive hypothesis } \} \\ & [X \subseteq \{\} \rightarrow (wait(n+m) \rightarrow P)] \\ \equiv & \{ \text{ defn. of wait } \} \\ & wait(n+m+1) \rightarrow P \end{array}$$

□

Stop

The process $STOP_A$ is not prepared to perform any event in its alphabet, A, at any time. The stability [RR86] of the process is not recorded in this model so this process covers both the case where the process is genuinely unwilling to participate in any event and the case where the process is only prepared to participate in internal events invisible to the environment. As far as the environment is concerned these are indistinguishable; in both cases the process is idling. So the process $STOP_A$ is the process which waits forever.

$$STOP_A \widehat{=} \mu\, x : A \bullet wait(1) \rightarrow x.$$

An obvious consequence of this definition is the following law:

L–12 : $wait(t) \rightarrow STOP_A \equiv STOP_A$.

Proof: by induction on t.

Base case $t = 0$, trivial by definition of $wait$.

Inductive step.

$$
\begin{array}{ll}
 & wait(t+1) \rightarrow STOP_A \\
\equiv & \{ \text{ by L–11 } \} \\
 & wait(1) \rightarrow (wait(t) \rightarrow STOP_A) \\
\equiv & \{ \text{ inductive hypothesis } \} \\
 & wait(1) \rightarrow STOP_A \\
\equiv & \{ \text{ defn. of } STOP \} \\
 & wait(1) \rightarrow (\mu\, x \bullet wait(1) \rightarrow x) \\
\equiv & \{ \text{ by A–17 } \} \\
 & \mu\, x \bullet wait(1) \rightarrow x \\
\equiv & \{ \text{ defn. of } STOP \} \\
 & STOP_A
\end{array}
$$

□

Run

The process RUN_A is always prepared to perform any set of events from its alphabet. It is given by:

$$RUN_A \mathrel{\widehat{=}} \mu\, x : A \bullet [X \subseteq A \rightarrow x]$$

As RUN_A never refuses to perform an event we have the following laws:

L–13 : $RUN_A \parallel RUN_B \equiv RUN_{A \cup B}$

Proof:

$$
\begin{array}{ll}
 & RUN_A \parallel RUN_B \\
\equiv & \{ \text{ defn. of } RUN \} \\
 & [X \subseteq A \rightarrow RUN_A] \parallel [X \subseteq B \rightarrow RUN_B] \\
\equiv & \{ \text{ by A–10 } \} \\
 & [Z \subseteq (A \cap B) \cup (A - B) \cup (B - A) \rightarrow RUN_A \parallel RUN_B] \\
\equiv & \{ \text{ set manipuation } \} \\
 & [Z \subseteq A \cup B \rightarrow RUN_A \parallel RUN_B] \\
\equiv & \{ \text{ by uniquness of solutions to guarded recursive equations } \} \\
 & RUN_{A \cup B}
\end{array}
$$

□

L–14 : $RUN_A \parallel P \equiv P \qquad \text{where} \qquad A \subseteq \alpha P$

Proof: Later, in Section 3.4, we show that every process can be characterised by a set of finite processes, each of which can be expressed using the process $\perp$ and the set prefix and non-deterministic choice constructs. By structural induction it is sufficient to establish the result in the following cases:

- $RUN_A \parallel \perp_B \equiv \perp_B$ if $A \subseteq B$, by A–6
- If $\alpha(P \sqcap Q) = B$ and $A \subseteq B$ then, $RUN_A \parallel (P \sqcap Q) \equiv P \sqcap Q$ assuming $RUN_A \parallel P \equiv P$ and $RUN_A \parallel Q \equiv Q$, follows from:

  $RUN_A \parallel (P \sqcap Q)$
  $\equiv$ {by A–8 }
  $(RUN_A \parallel P) \sqcap (RUN_A \parallel Q)$
  $\equiv$ { by inductive hypothesis }
  $P \sqcap Q$

- Assuming $\alpha([X \subseteq B' \to P_X]) = B$, $A \subseteq B$ and $\forall C \subseteq B' \bullet RUN_A \parallel P_C \equiv P_C$ then, $RUN_A \parallel [X \subseteq B' \to P_X] \equiv [X \subseteq B' \to P_X]$ follows from

  $RUN_A \parallel [X \subseteq B' \to P_X]$
  $\equiv$ {expanding definition of RUN }
  $[Y \subseteq A \to RUN_A] \parallel [X \subseteq B' \to P_X]$
  $\equiv$ { by A–10 }
  $[Z \subseteq (A \cap B') \cup (B' - A) \to P_{Z \cap B'} \parallel RUN_A]$
  $\equiv$ { rearranging terms }
  $[Z \subseteq B' \to P_Z \parallel RUN_A]$
  $\equiv$ { by inductive hypothesis }
  $[Z \subseteq B' \to P_Z]$

□

Event prefixing

If P is a process and a an event in the alphabet of P, then the process $a \rightsquigarrow P$ (P prefixed by event a) will wait indefinitely until an a is offered, at which point the a is performed and the process behaves like P. This is defined as follows:

$$a \rightsquigarrow P \mathrel{\hat{=}} \mu\, x \bullet [\{a\} \to P \rhd x]$$

This event guarded construct has a behaviour comparable with the prefix construct of both CSP and Timed CSP. For example we have the following laws:

L–15 : $(a \rightsquigarrow P) \sqcap (a \rightsquigarrow Q) \equiv a \rightsquigarrow (P \sqcap Q)$

L–16 : $(a \rightsquigarrow P) \parallel (a \rightsquigarrow Q) \equiv a \rightsquigarrow (P \parallel Q)$

L–17 :
$$(a \rightsquigarrow P) \parallel (b \rightsquigarrow Q) \equiv \begin{cases} STOP & \text{if } a, b \in \alpha P \cap \alpha Q \\ & \quad a \neq b \\ a \rightsquigarrow (P \parallel (b \rightsquigarrow Q)) & \text{if } a \notin \alpha P \cap \alpha Q \\ & \quad b \in \alpha P \cap \alpha Q \\ b \rightsquigarrow ((a \rightsquigarrow P) \parallel Q) & \text{if } a \in \alpha P \cap \alpha Q \\ & \quad b \notin \alpha P \cap \alpha Q \\ [\{a, b\} \rightarrow (P \parallel Q) & \text{if } a, b \notin \alpha P \cap \alpha Q \\ \quad \Box \{a\} \rightarrow P \parallel (b \rightsquigarrow Q) & \\ \quad \Box \{b\} \rightarrow (a \rightsquigarrow P) \parallel Q & \\ \quad \rhd (a \rightsquigarrow P) \parallel (b \rightsquigarrow Q)] & \end{cases}$$

Notice that our explicit 'true concurrency' approach results in the possibility of both a and b occurring simultaneously when the occurrence of these events is independent of interaction between component processes.

L–18 : $(a \rightsquigarrow P) \setminus \{a\} \equiv wait(1) \rightarrow (P \setminus \{a\})$

Here we see that, like Reed and Roscoe's Timed CSP [RR86], hiding events cannot hide time. Although we can hide the event a prefixing P, the time lapse, during which the hidden event is performed internally, remains visible.

2.2 Examples

In this section we present two examples which demonstrate the features of the language and the use of the algebraic laws.

2.2.1 A watchdog timer

A watchdog timer, as proposed by Hooman [Hoo90], monitors several processes in a system. Each of the processes which is being monitored is required to send an *ok* signal to the monitor at regular intervals. If the time between *ok* signals from any one process exceeds a given maximum then the timer signals failure.

Implementations of watchdog timers have been proposed in Timed CSP [Sch91] and PARTY [HSZFH92]. We shall demonstrate two ways of constructing such a timer in our algebra. Using the algebraic laws we then show these two implementations are equivalent.

For simplicity we shall consider a system in which there are two processes being monitored. Suppose also that the interval between consecutive *ok* signals from each process must not exceed 2 units.

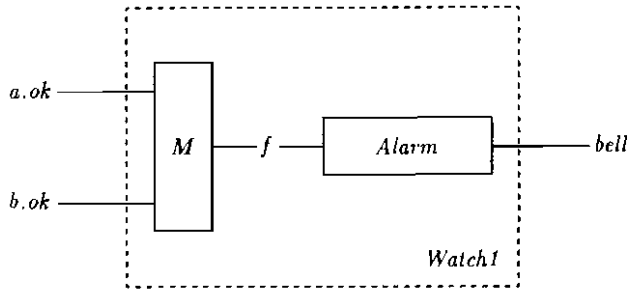


Figure 2.1: Watchdog timer: version 1

Version 1

In this version, Figure 2.1, the process M does all the work, monitoring the two processes on $a.ok$ and $b.ok$. If a signal is not received on either channel within the required time then failure is signalled to *Alarm*. Once the failure has been signalled the timer switches off and no longer checks the frequency of the signals from the processes it was monitoring.

$$M \mathrel{\widehat{=}} M_{\{\}} \qquad \alpha M = \{a.ok, b.ok, f\}$$

$$\begin{array}{rcl} \text{where } M_Y & \mathrel{\widehat{=}} & [X \subseteq \{a.ok, b.ok\} \rightarrow (M_{\{a.ok,b.ok\}-X} \textbf{ if } Y \subseteq X \textbf{ else } \mathit{Fail})] \\ \mathit{Fail} & \mathrel{\widehat{=}} & [X \subseteq \{a.ok, b.ok, f\} \rightarrow (\mathit{Off} \textbf{ if } f \in X \textbf{ else } \mathit{Fail})] \\ \mathit{Off} & \mathrel{\widehat{=}} & [X \subseteq \{a.ok, b.ok\} \rightarrow \mathit{Off}] \end{array}$$

M_Y will fail unless every event in Y occurs in the next interval. *Fail* will signal f as soon as the alarm is ready.

The alarm simply waits for a failure signal and then rings an alarm, signalled by the event *bell*. We are not interested in the behaviour of the system once the alarm has been raised; this is modelled by assuming the worst possible behaviour.

$$\begin{array}{rcll} \mathit{Alarm} & \mathrel{\widehat{=}} & f \rightsquigarrow \mathit{Ring} & \qquad \alpha \mathit{Alarm} = \{f, bell\} \\ \mathit{Ring} & \mathrel{\widehat{=}} & bell \rightsquigarrow \perp & \end{array}$$

The watchdog timer system is then given by composing the monitor process and the alarm in parallel and making the failure signal f internal.

$$\mathit{Watch1} \mathrel{\widehat{=}} (M \parallel \mathit{Alarm}) \setminus \{f\}$$

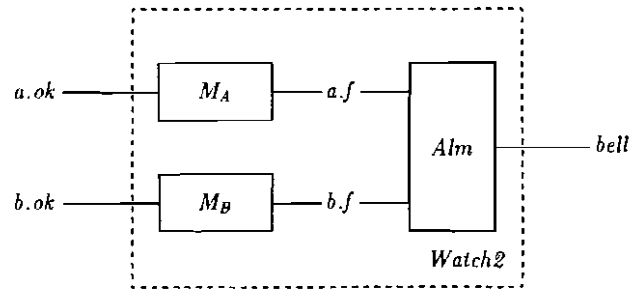


Figure 2.2: Watchdog timer: version 2

Version 2

The second approach, Figure 2.2, is to divide the task of monitoring the various processes among several components in the timer. Each component in the timer monitors a different process. The alarm is raised by the monitor as a whole if one of these components detects an error.

The general behaviour of the monitoring component is given by $M1$.

$$\begin{array}{rcll} M1 & \widehat{=} & [\{ok\} \rightarrow M1 \rhd M1'] & \alpha M1 = \{ok, f\} \\ M1' & \widehat{=} & [\{ok\} \rightarrow M1 \rhd F] & \\ F & \widehat{=} & [X \subseteq \{ok, f\} \rightarrow (O \textbf{ if } f \in X \textbf{ else } F)] & \\ O & \widehat{=} & ok \rightsquigarrow O & \end{array}$$

The monitoring component awaits an ok signal from the specified process and signals failure to the alarm if excess time has elapsed since the process last communicated satisfactorily with the monitor. The specific monitoring components in our system can be expressed as renamings of such a general monitor component.

$$M_A \widehat{=} M1[a.ok/ok, a.f/f]$$

$$M_B \widehat{=} M1[b.ok/ok, b.f/f]$$

The alarm waits for a failure signal on either of the channels $a.f$ or $b.f$. Once such a signal has been observed the alarm activates the *bell*. As before we are not concerned with the subsequent behaviour of the system and we assume that the alarm, and hence the whole system, behaves chaotically after its task has been performed.

$$\begin{array}{rcl} Alm & \widehat{=} & [X \subseteq \{a.f, b.f\} \rightarrow (Rng \textbf{ if } X \neq \{\} \textbf{ else } Alm)] \\ Rng & \widehat{=} & bell \rightsquigarrow \bot \end{array}$$

The watchdog timer is obtained by taking the parallel composition of these three components and hiding the failure signals $a.f$ and $b.f$

$$Watch2 \triangleq (M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\}$$

Comparing processes

We wish to establish that *Watch1* and *Watch2* are equivalent systems. To achieve this we shall, using the algebraic laws, reformulate the two processes eliminating both parallel composition and hiding constructs. Once in this simplified form it will be trivial to verify that the two systems are equivalent.

First we consider *Watch1*. By applying the algebraic laws and appealing to the uniqueness of guarded recursive equations we can establish that

$$\begin{array}{l} Watch1 \equiv W1_{\{\}} \\ \text{where} \quad W1_Y \triangleq [X \subseteq \{a.ok, b.ok\} \rightarrow (W1_{\{a.ok, b.ok\}-X} \text{ if } Y \subseteq X \text{ else } F1)] \\ \qquad\qquad F1 \triangleq [X \subseteq \{a.ok, b.ok\} \rightarrow R1] \\ \qquad\qquad R1 \triangleq [X \subseteq \{a.ok, b.ok, bell\} \rightarrow (\perp \text{ if } bell \in X \text{ else } R1)] \end{array}$$

It remains to consider *Watch2*. In this case we will go into a little more detail. To ease notation we shall write M'_A, F_A, O_A and M'_B, F_B, O_B for the obvious renamings of $M1'$, F and O. By expanding the expressions and applying the algebraic laws A-6, A-10, A-11 and A-13 we see that

$$\left\langle \begin{array}{l} (O_A \parallel M_B \parallel Rng) \setminus \{a.f, b.f\} \\ (O_A \parallel M'_B \parallel Rng) \setminus \{a.f, b.f\} \\ (O_A \parallel F_B \parallel Rng) \setminus \{a.f, b.f\} \end{array} \right\rangle$$

satisfies the guarded mutually recursive equation:

$$\left\langle \begin{array}{l} x \hat{=} [X \subseteq \{a.ok, b.ok, bell\} \rightarrow (\perp \text{ if } bell \in X \text{ else } (x \text{ if } b.ok \in X \text{ else } y))] \\ y \hat{=} [X \subseteq \{a.ok, b.ok, bell\} \rightarrow (\perp \text{ if } bell \in X \text{ else } (x \text{ if } b.ok \in X \text{ else } z))] \\ z \hat{=} [X \subseteq \{a.ok, b.ok, bell\} \rightarrow (\perp \text{ if } bell \in X \text{ else } z)] \end{array} \right\rangle$$

Moreover so too does $\langle R1, R1, R1 \rangle$. So by uniqueness of solution to guarded recursive equations

$$\left. \begin{array}{l} (O_A \parallel M_B \parallel Rng) \setminus \{a.f, b.f\} \\ (O_A \parallel M'_B \parallel Rng) \setminus \{a.f, b.f\} \\ (O_A \parallel F_B \parallel Rng) \setminus \{a.f, b.f\} \end{array} \right\} \equiv R1$$

By a similar argument we can show that

$$\left. \begin{array}{l} (M_A \parallel O_B \parallel Rng) \setminus \{a.f, b.f\} \\ (M'_A \parallel O_B \parallel Rng) \setminus \{a.f, b.f\} \\ (F_A \parallel O_B \parallel Rng) \setminus \{a.f, b.f\} \end{array} \right\} \equiv R1$$

and

$$(O_A \parallel O_B \parallel Rng) \setminus \{a.f, b.f\} \equiv R1$$

Now by expanding terms and noting the above ideutities we can show that

$$\left.\begin{array}{l}(M_A \parallel F_B \parallel Alm) \setminus \{a.f, b.f\} \\ (M'_A \parallel F_B \parallel Alm) \setminus \{a.f, b.f\} \\ (F_A \parallel F_B \parallel Alm) \setminus \{a.f, b.f\} \\ (F_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\} \\ (F_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\}\end{array}\right\} \equiv F1$$

Finally, by expanding the expressions and applying the algebraic laws A–10 and A–13, we can establish that

$$\begin{array}{l}(M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \quad \equiv [\{a.ok, b.ok\} \rightarrow (M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \qquad \Box \{a.ok\} \rightarrow (M_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\} \\ \qquad \Box \{b.ok\} \rightarrow (M'_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \qquad \rhd (M'_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\}]\end{array}$$

$$\begin{array}{l}(M_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\} \\ \quad \equiv [\{a.ok, b.ok\} \rightarrow (M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \qquad \Box \{b.ok\} \rightarrow (M'_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \rhd F1]\end{array}$$

$$\begin{array}{l}(M'_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \quad \equiv [\{a.ok, b.ok\} \rightarrow (M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \\ \qquad \Box \{a.ok\} \rightarrow (M_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\} \rhd F1]\end{array}$$

$$\begin{array}{l}(M'_A \parallel M'_B \parallel Alm) \setminus \{a.f, b.f\} \\ \quad \equiv [\{a.ok, b.ok\} \rightarrow (M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \rhd F1]\end{array}$$

By uniqueness of guarded recursive equations $(M_A \parallel M_B \parallel Alm) \setminus \{a.f, b.f\} \equiv W1$. Hence

$$Watch1 \equiv Watch2$$

as required, showing the systems equivalent.

2.2.2 A lift lobby

As a second example, we shall consider the behaviour of Hoare's lift lobby [Hoa86]. We shall see that the 'true concurrency' approach of SCSP gives us a slightly different insight into the working of the system from the 'interleaving concurrency' approach of CSP. Also the implicit discrete time framework of the algebra allows us to consider timing constraints on the system, constraints that could not even be expressed in CSP.

Components of the system

The lift lobby consists of a button, light and door. Each of these three components performs two actions; the two actions of a given component occur alternately. The button can be *pressed* or *released*, the light goes *on* and *off* and the door *opens* and *closes*. Initially the button is released, the light is off and the door is closed. So the three components can he described as follows.

$$\alpha BUTTON = \{press, release\}$$
$$BUTTON \mathrel{\widehat{=}} R \qquad R \mathrel{\widehat{=}} press \rightsquigarrow D \qquad D \mathrel{\widehat{=}} release \rightsquigarrow R$$

$$\alpha LIGHT = \{on, off\}$$
$$LIGHT \mathrel{\widehat{=}} F \qquad F \mathrel{\widehat{=}} on \rightsquigarrow N \qquad N \mathrel{\widehat{=}} off \rightsquigarrow F$$

$$\alpha DOOR = \{open, elose\}$$
$$DOOR \mathrel{\widehat{=}} C \qquad C \mathrel{\widehat{=}} open \rightsquigarrow O \qquad O \mathrel{\widehat{=}} close \rightsquigarrow C$$

When considering the system it is helpful to realize that, as specified, tbe button light and door each have two states. For example the button is either depressed (D) or released (R), depending on the last action performed by the button. In the depressed state the only event the button may participate in is a *release*, while in the released state *press* is the only event in which the button may participate. Similarly the light is either off (F) or on (N) and the door is either closed (C) or open (O). Making use of these observations, we are able to provide very simple algebraic encodings of the requirements.

System requirements

We shall assume several coustraints on the interaction between the components of the system. Using the algebra, we are able to establish the possible observations of events in the lift lobby. All the conditions we shall place on the system are safety requirements; these restrict the occurrence rather than the refusal of events. Consequently, by the nature of parallel composition, we can take the composition of processes, each of which represents an individual constraint, to obtain a process which satisfies all the constraints.

When developing processes which specify system requirements, we shall use a naming convention which reflects the state of the system at each point. The safety requirements of the system are given below.

- The light does not go off while the door is closed.

$$S1 \mathrel{\widehat{=}} C1 \qquad \alpha S1 = \{off, open, close\}$$
$$C1 \mathrel{\widehat{=}} open \rightsquigarrow O1$$
$$O1 \mathrel{\widehat{=}} [X \subseteq \{off, close\} \rightarrow (C1 \textbf{ if } close \in X \textbf{ else } O1)]$$

Notice that we allow the door to close and the light to go off simultaneously. We assume that the light can only go off if it observes that the door is open. It must take time to react to such an observation so it is reasonable for the light's reaction to coincide with the door closing. It is this type of assumption which gives us a different view of the system to that given in CSP.

- The light does not go on unless the button is depressed.

$$S2 \triangleq R2 \qquad \alpha S2 = \{press, release, on\}$$
$$R2 \triangleq press \rightsquigarrow D2$$
$$D2 \triangleq [X \subseteq \{on, release\} \rightarrow (R2 \textbf{ if } release \in X \textbf{ else } D2)]$$

- The light goes on only when the door is closed.

$$S3 \triangleq C3 \qquad \alpha S3 = \{on, open, close\}$$
$$C3 \triangleq [X \subseteq \{on, open\} \rightarrow (O3 \textbf{ if } open \in X \textbf{ else } C3]$$
$$O3 \triangleq close \rightsquigarrow C3$$

- The door cannot close while the button is depressed.

$$S4 \triangleq R4 \qquad \alpha S4 = \{press, release, close\}$$
$$R4 \triangleq [X \subseteq \{press, close\} \rightarrow (D4 \textbf{ if } press \in X \textbf{ else } R4)]$$
$$D4 \triangleq release \rightsquigarrow R4$$

- The door does not open unless the light is on or the button is depressed.

$$S5 \triangleq RF5 \qquad \alpha S5 = \{press, release, on, off, open\}$$
$$RF5 \triangleq [X \subseteq \{press, on\} \rightarrow (DN5 \textbf{ if } on \in X \textbf{ else } DF5) \textbf{ if } press \in X \textbf{ else } (RN5 \textbf{ if } press \in X \textbf{ else } RF5)]$$
$$DF5 \triangleq [X \subseteq \{release, on, open\} \rightarrow (RN5 \textbf{ if } on \in X \textbf{ else } RF5) \textbf{ if } release \in X \textbf{ else } (DN5 \textbf{ if } on \in X \textbf{ else } DF5)]$$
$$RN5 \triangleq [X \subseteq \{press, off, open\} \rightarrow (DF5 \textbf{ if } off \in X \textbf{ else } DN5) \textbf{ if } press \in X \textbf{ else } (RF5 \textbf{ if } off \in X \textbf{ else } RN5)]$$
$$DN5 \triangleq [X \subseteq \{release, off, open\} \rightarrow (RF5 \textbf{ if } off \in X \textbf{ else } RN5) \textbf{ if } release \in X \textbf{ else } (DF5 \textbf{ if } off \in X \textbf{ else } DN5)]$$

- The button is not released if the doors are closed and the light is off.

$$
\begin{aligned}
&S6 \triangleq RF6 \qquad \alpha S6 = \{release, on, off, open, close\} \\
&FC6 \triangleq [X \subseteq \{on, open\} \rightarrow (NO6 \textbf{ if } on \in X \textbf{ else } FO6) \\
&\qquad\qquad \textbf{if } open \in X \textbf{ else} \\
&\qquad\qquad (NC6 \textbf{ if } on \in X \textbf{ else } FC6)] \\
&NC6 \triangleq [X \subseteq \{off, open, release\} \rightarrow (FO6 \textbf{ if } off \in X \textbf{ else } NO6) \\
&\qquad\qquad \textbf{if } open \in X \textbf{ else} \\
&\qquad\qquad (FC6 \textbf{ if } off \in X \textbf{ else } NC6)] \\
&FO6 \triangleq [X \subseteq \{on, close, release\} \rightarrow (NC6 \textbf{ if } on \in X \textbf{ else } FC6) \\
&\qquad\qquad \textbf{if } close \in X \textbf{ else} \\
&\qquad\qquad (NO6 \textbf{ if } on \in X \textbf{ else } FO6)] \\
&NO6 \triangleq [X \subseteq \{off, close, release\} \rightarrow (FC6 \textbf{ if } off \in X \textbf{ else } NC6) \\
&\qquad\qquad \textbf{if } close \in X \textbf{ else} \\
&\qquad\qquad (FO6 \textbf{ if } off \in X \textbf{ else } NO6)]
\end{aligned}
$$

- The door does not close when the light is on.

$$
\begin{aligned}
&S7 \triangleq F7 \qquad \alpha S7 = \{press, release, close\} \\
&F7 \triangleq [X \subseteq \{on, close\} \rightarrow (N7 \textbf{ if } on \in X \textbf{ else } F7)] \\
&N7 \triangleq off \rightsquigarrow F7
\end{aligned}
$$

Behaviours of the Lift Lobby

As already suggested, we can take the composition of these seven constraints and the three components in the lift lobby to establish the allowed behaviours of the system. We are interested in the process given by:

$$LIFT \triangleq BUTTON \parallel LIGHT \parallel DOOR \parallel S1 \parallel S2 \parallel S3 \parallel S4 \parallel S5 \parallel S6 \parallel S7$$

By use of the algebraic laws we are able to eliminate the parallel composition operator from this expression, giving:

$$LIFT \equiv RFC$$

where

$$
\begin{aligned}
RFC &\triangleq press \rightsquigarrow DFC \\
DFC &\triangleq [\{on, open\} \rightarrow DNO \,\Box\, \{on\} \rightarrow DNC \,\Box\, \{open\} \rightarrow DFO \triangleright DFC] \\
DNO &\triangleq [\{release, off\} \rightarrow RFO \\
&\qquad \Box\, \{release\} \rightarrow RNO \,\Box\, \{off\} \rightarrow DFO \triangleright DNO] \\
DNC &\triangleq [\{release, open\} \rightarrow RNO \\
&\qquad \Box\, \{release\} \rightarrow RNC \,\Box\, \{open\} \rightarrow DNO \triangleright DNC] \\
DFO &\triangleq release \rightsquigarrow RFO \\
RFO &\triangleq [\{press, close\} \rightarrow DFC \\
&\qquad \Box\, \{press\} \rightarrow DFO \,\Box\, \{close\} \rightarrow RFC \triangleright RFO] \\
RNO &\triangleq [\{press, off\} \rightarrow DFO \,\Box\, \{press\} \rightarrow DNO \,\Box\, \{off\} \rightarrow RFO \triangleright RNO] \\
RNC &\triangleq [\{press, open\} \rightarrow DNO \\
&\qquad \Box\, \{press\} \rightarrow DNC \,\Box\, \{open\} \rightarrow RNO \triangleright RNC]
\end{aligned}
$$

Here the naming of processes is such that the first letter corresponds to the state of the button, the second to the state of the light and the final to the state of the door. (In the initial state the button is released, light off and door closed.) The transitions between the various states of the system are shown in Figure 2.3.

Timing constraints

Unlike CSP we can also consider timing constraints on our system. Suppose we now insist that the button cannot be released until two time units after it was last pressed. The doors cannot close while the button is depressed ($S4$), so in the situation where the button is pressed while the door is open, this new constraint will ensure an increased delay before the door may be shut. This could be seen as implementing a hold facility in our system. The new constraint is given by:

$$
\begin{aligned}
S8 &\triangleq R8 \qquad \alpha S8 = \{press, release\} \\
R8 &\triangleq press \rightsquigarrow (wait(1) \rightarrow D8) \\
D8 &\triangleq release \rightsquigarrow R8
\end{aligned}
$$

The effect of such a requirement on the system can easily be established by consideration of the process:

$$LIFT \parallel S8$$

We find that the behaviour of the new system is similar to that of the old with only the following terms in the mutual recursion being different to the processes in

the expansion of the process $LIFT$.

$$
\begin{aligned}
RF0 \triangleq\ & [\{press, close\} \rightarrow DFC \\
& \quad \Box \{press\} \rightarrow DFO' \Box \{close\} \rightarrow RFC \rhd RFO] \\
RN0 \triangleq\ & [\{press, off\} \rightarrow DFO' \\
& \quad \Box \{press\} \rightarrow DNO' \Box \{off\} \rightarrow RFO \rhd RNO] \\
RNC \triangleq\ & [\{press, open\} \rightarrow DNO' \\
& \quad \Box \{press\} \rightarrow DNC' \Box \{open\} \rightarrow RNO \rhd RNC]
\end{aligned}
$$

where

$$
\begin{aligned}
DFO' &\triangleq wait(1) \rightarrow DFO \\
DNO' &\triangleq [\{off\} \rightarrow DFO \rhd DNO] \\
DNC' &\triangleq [\{open\} \rightarrow DNO \rhd DNC]
\end{aligned}
$$

When the light is off and the door closed, then pressing the button initiates a call for the lift. We see that in this situation the introduction of extra unit delay between the event *press* and the subsequent *release* does not cause a delay in the overall system. It is only in situations where the system is already in the process of responding to a button press that the delay affects the possible behaviours.

2.3 Conclusion

In this chapter we have presented a language SCSP which

- expresses non-determinism, parallelism, hiding and recursion in a manner comparable with CSP;
- incorporates quantitative timing details via a set prefix operator, which takes a unit of time to evolve;
- captures the notion of true concurrency by a set prefix operator which represents the possibility of simultaneously occurring events;
- has sufficient algebraic laws to be able to eliminate parallel composition and hiding from expressions.

These features were demonstrated via examples, where we showed implementations of a watchdog timer to be equivalent and examined the allowed behaviours of a lift system visible from the lift lobby.

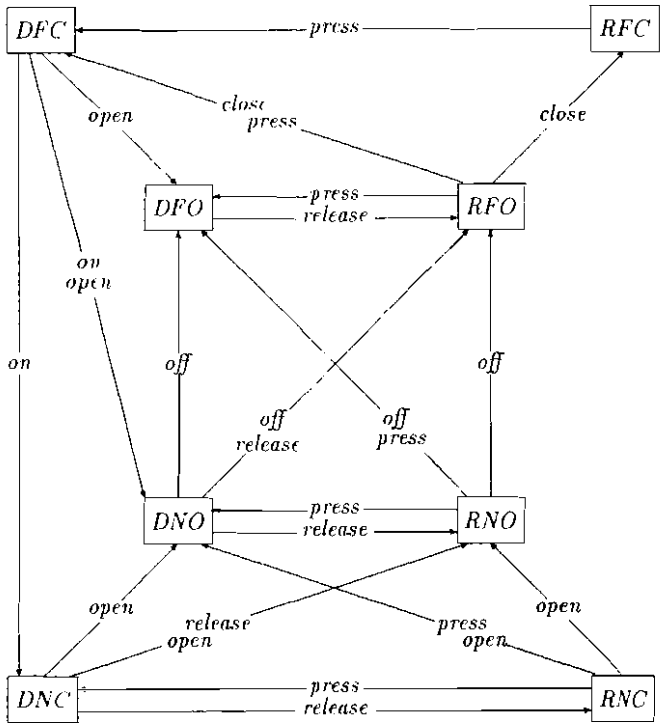


Figure 2.3: State transitions observable in the lift lobby
The transition arrows are labelled with the events that occur at that time step. The arrows corresponding to no events occurring at a time step are omitted; in all cases this does not result in a change of state.

Chapter 3

Semantics and Proof System for SCSP

In this chapter we present a denotational semantics for the language SCSP. The semantic model records the behaviours of processes in the form of failures-divergences information. This model forms a complete partial order under the non-determinism ordering presented in Section 3.1.3, which enables us to use a domain theoretic approach to establish a semantics for recursion.

Finally we develop a proof system for SCSP which is sound and complete with respect to the denotational semantics. The availability of such a proof system allows us to use an axiomatic approach when reasoning about process expressions.

3.1 Semantic Model

In this section we develop a semantic model which mathematically underpins the algebra presented in Chapter 2. The underlying model records failures-divergences information in a simple format. As we shall see the model consists of traces of sets; the structure of the traces naturally capturing the implicit timing aspect. By the provision of a semantic function from SCSP to the model and use of the structure inherent in the model we are able to make explicit the meaning of recursively-defined expressions in SCSP.

3.1.1 Notation

Here we introduce the key concepts of the model. We also provide some useful operators on traces.

Events and observation-sets

The model takes as its basis the notion that, at each time step, a process may engage in or refuse events offered by the environment. The *universal alphabet* Σ denotes the set of all possible events. From Σ we construct a disjoint set $\bar{\Sigma} \triangleq \{\bar{a} \mid a \in \Sigma\}$ of refusals. The occurrence of an event a is denoted by the event itself, whereas its refusal is denoted by $\bar{a} \in \bar{\Sigma}$.

A particular process may participate in a finite, non-zero number of events $A \in \mathbb{F}\Sigma - \{\}$; this set is the alphabet of the process, while the set $\bar{A}$, being the obvious subset of $\bar{\Sigma}$, is called the refusal alphabet. Clearly the refusal alphabet is disjoint from the alphabet itself, $\bar{A} \cap A = \{\}$.

We assume that the refusal of an event is as observable as its occurrence. To reflect this view we record as an *observation-set* the events which occurred or were refused at a given time step. For a process with alphabet A each observation-set is a subset of $\dot{A} \triangleq A \cup \bar{A}$. The observation-set provides a record of the process' behaviour in response to the environment making available a set of events at a given instant of time.

Traces

A trace is a finite sequence of observation-sets. So, given the alphabet A, the set of all traces is given by:

$$ST_A \triangleq (\mathbb{P}(\dot{A}))^*$$

At each tick of a global clock an observer may witness a number of occurrences and refusals from the set $\dot{A}$. By recording these sets of observations in a chronologically ordered sequence we obtain a trace of the process. Time is recorded implicitly – if there is nothing to observe at a given time, the empty set is recorded in the trace. Thus the n^{th} element of the trace is the set of observations seen at time n.

It is convenient to introduce operators on traces and observation-sets. The union of an observation-set, B, with a trace is the mapping of 'union B' to each element of the trace. Formally: *

$$\begin{aligned} \langle\rangle \cup B &= \langle\rangle \\ (\langle C \rangle^\frown s) \cup B &= \langle C \cup B \rangle^\frown (s \cup B) \end{aligned}$$

We define the intersection $(s \cap B)$ and subtraction $(s - B)$ similarly.

We also lift operators on sets to operators on traces; these return a trace whose n^{th} element is the result of the set operation on the n^{th} elements of the original

* $s_1 \frown s_2$ represents concatenation of traces s_1 and s_2.

traces. For example, the union of two traces is given by:

$$\begin{aligned} \langle\rangle \cup r &= r \\ s \cup \langle\rangle &= s \\ (\langle B\rangle^\frown s) \cup (\langle C\rangle^\frown r) &= \langle B \cup C\rangle^\frown (s \cup r) \end{aligned}$$

Saturation

Due to the maximal progress assumption made when defining **hiding**, the concept of a saturated trace is important.

Definition 3.1 A trace s is *saturated* with respect to a subset A' of the alphabet A if at each point in time the trace records the occurrence or refusal of each event in A'.

$$saturated_{A'}(s) \mathrel{\hat{=}} \forall a \in A', B \underline{\text{in}}\, s \bullet a \in B \vee \bar{a} \in B$$

where the relation $\underline{\text{in}}$ is defined as follows:

$$B \underline{\text{in}}\, s \mathrel{\hat{=}} \exists u, v \bullet u^\frown \langle B\rangle^\frown v = s.$$

This holds whenever the element B appears in the trace s. ◊

If the environment always offers all the elements from A' then a trace saturated with respect to the events in A' will be observed.

Feasibility

It makesno sense for an event to occur and be refused simultaneously. An observation-set B is feasible, $feasible(B)$, exactly when

$$feasible(B) \mathrel{\hat{=}} \forall a \in A \bullet \neg (a \in B \wedge \bar{a} \in B)$$

Processes

A process P is represented in our model by the pair (A, T), where $A = \alpha P$ is the alphabet of the process and T is the set of traces describing all possible behaviours of the process.

Not all subsets of ST_A will represent trace-sets of processes. Those that do, also satisfy certain closure conditions which are considered in the next section.

3.1.2 Closure conditions

In this section, we introduce closure conditions on a set T of traces with respect to alphabet A. Only those sets of traces which meet the closure conditions describe the observable behaviour of a process in our algebra.

i $\quad \langle\rangle \in T$

The empty trace is observable at time 0.

ii $\quad s^\frown r \in T \Rightarrow s \in T$

If a particular trace can be observed over a certain time span, then prefixes of this trace, corresponding to observations made for a shorter period of time, may also be observed.

iii $\quad s^\frown\langle B\rangle \in T \wedge B' \subseteq B \Rightarrow s^\frown\langle B'\rangle \in T$

If a set of observations is made at one time, then it would also be possible to observe any subset of these observations. We would see fewer events refused and fewer events occur; the environment offered the process fewer events. (If the environment does not offer an event then it can neither occur nor be refused.) This closure condition also says that events that occur simultaneously are not dependent on one another: events occur or are refused regardless of the occurrence or refusal of other events at that time.

iv $\quad s \in T \Rightarrow s^\frown\langle\{\}\rangle \in T$

Time marches on. If the environment does not offer the process any events the process will idle, which is recorded by extending the trace with an empty set.

v $\quad X \subseteq \bar{A} \wedge s^\frown\langle B\rangle^\frown r \in T \Rightarrow s^\frown\langle B - X\rangle^\frown r \in T$

The subsequent behaviour of a process is not affected if the environment does not offer an event that would be refused.

vi $\quad s^\frown\langle B\rangle^\frown r \in T$
$\wedge\, X = \{a \in A \mid a \notin B \wedge a \notin B\} \Rightarrow \quad s^\frown\langle B \cup X\rangle \in T$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee\, \exists a \in X \bullet s^\frown\langle B \cup \{\bar{a}\}\rangle^\frown r \in T$

A process will always respond to events offered by the environment, either by refusing or performing them. If it cannot perform all events it has not refused, then there must be one such event which the process can refuse without affecting its subsequent behaviour.

vii $\quad s^\frown\langle B\rangle \in T \wedge \neg\, \mathit{feasible}(B) \wedge r \in (\mathbb{P}(\bar{A}))^* \Rightarrow s^\frown r \in T$

Once a process is able to exhibit infeasible behaviour, it can do anything from that time on. In CSP, such a process is said to be divergent.

We shall denote by SM the set of all pairs (A, T) where A is a finite alphabet and T satisfies the closure conditions with respect to the alphabet A. This set is the underlying model for our algebra.

$$SM \triangleq \{(A, T') \mid A \in \mathbb{F}(\Sigma) - \{\} \wedge T' \subseteq ST_A \wedge T' \text{ satisfies conditions i-vii}\}$$

where Σ is the universal set of all events.

We shall let SM^A be the set of all processes with alphabet A.

$$SM^A \triangleq \{(A, T') \mid (A, T') \in SM\}$$

Further, we shall use SM_T to denote the set of all sets of traces for processes and SM_T^A to denote the subset of SM_T corresponding to processes with alphabet A.

$$SM_T \triangleq \{T' \mid \exists A \in \mathbb{F}(\Sigma) \bullet (A, T') \in SM\}$$

$$SM_T^A \triangleq \{T' \mid (A, T') \in SM^A\}$$

3.1.3 Non-determinism ordering

We define a non-determinism ordering on processes with the same alphabet. If (A, T_P) and (A, T_Q) represent two processes P and Q then we define the ordering $\sqsubseteq$ by:

$$(A, T_P) \sqsubseteq (A, T_Q) \triangleq T_Q \subseteq T_P$$

We say that Q is more deterministic than P, $P \sqsubseteq Q$, in the same sense as in the CSP failures-divergences model. [BR85]; that is Q is more predictable than P; any behaviour of Q is a behaviour of P.

That $\sqsubseteq$ is a complete partial order on SM^A follows from the following theorem.

Theorem 3.1 *$(SM_T^A, \supseteq)$ forms a complete partial order.*

Proof: We must show

a) SM_T^A has a least element under the ordering,

b) every directed $D \subseteq SM_T^A$ has a least upper bound in SM_T^A.

a) It is a trivial exercise to show that ST_A satisfies the closure conditions i-vii and clearly. $\forall T' \in SM_T^A$, $ST_A \supseteq T'$.

b) Suppose $D \subseteq SM_T^A$ is directed. Clearly $\bigcap D$ is the least upper bound of D in $\mathbb{P}(ST_A) \supseteq SM_T^A$. Thus $\bigcap D$ is the required least upper bound if $\bigcap D \in SM_T^A$. $\bigcap D$ trivially satisfies closure conditions i-v and vii. It remains to show $\bigcap D$ satisfies condition vi.

vi) Assume $s^\frown\langle B\rangle^\frown t \in \bigcap D$.

Now suppose $\bigcap D$ fails condition vi. Then setting $X = \{a \in A \mid a \notin B \wedge \bar{a} \notin B\}$ we have $s^\frown\langle B \cup X\rangle \notin \bigcap D$ and $\forall a \in X \bullet s^\frown\langle B \cup \{\bar{a}\}\rangle^\frown t \notin \bigcap D$. Hence $\exists D_0 \in D$ such that $s^\frown\langle B \cup X\rangle \notin D_0$. Also we can find a finite indexing set I to cover the elements of X, i.e. $X = \{a_i \mid i \in I\}$, so that for each $i \in I$, there exists a $D_i \in D$ such that $s^\frown\langle B \cup \{\bar{a}_i\}\rangle^\frown t \notin D_i$.

By the property of directed sets and as I is finite, we can find some $D' \in D$ such that $D_0 \supseteq D' \wedge \forall i \in I \bullet D_i \supseteq D'$.

So $s^\frown\langle B\rangle^\frown t \in D' \wedge s^\frown\langle B \cup X\rangle \notin D' \wedge \forall a \in X \bullet s^\frown\langle B \cup \{\bar{a}\}\rangle^\frown t \notin D'$, contradicting $D' \in SM_T^A$. □

The set of processes with finite alphabet A represented by our model also form a complete semi-lattice under the refinement ordering, $\sqsubseteq$. This is a direct consequence of the theorem:

Theorem 3.2 *$(SM_T^A, \supseteq)$ forms a complete semi-lattice.*

Proof: We already have from the previous theorem that $(SM_T^A, \supseteq)$ forms a complete partial order, so it remains to show that arbitrary subsets of SM_T^A have a greatest lower bound in SM_T^A.

By construction $\bigcup B$ is the greatest lower bound of the arbitrary subset $B \subseteq SM_T^A$, seen as a subset of the set $\mathbb{P}(ST_A) \supseteq SM_T^A$. Moreover $\bigcup B$ trivially satisfies the closure conditions i--vii, hence $\bigcup B \in SM_T^A$. □

3.2 Semantic Function

In this section we construct a semantic function which maps syntactic expressions of our language to processes in our model SM. Before we can consider the semantics of a process term we must provide a specific binding of process variables to processes in the standard way [Sto77, Hen88].

Variable bindings

Given a set of variables Var, we define a domain of bindings, $BIND$. This consists of all mappings from Var to the space of processes SM.

$$BIND \cong Var \to SM$$

Now we are able to define a semantic function

$$\mathcal{M} : SCSP \to BIND \to SM$$

$\mathcal{M}[\![P]\!]\rho$ denotes the meaning of process term P with variable binding ρ in terms of our model. This is evaluated by associating each free variable x with its value $\rho[\![x]\!]$ given by the binding ρ.

Syntactic substitution of free variables, as introduced in Section 2.1.2, results in a change in the variable binding in which a process is given its meaning. This occurs in the following sense:

$$\mathcal{M}[\![P[Q/x]]\!]\rho \triangleq \mathcal{M}[\![P]\!]\rho[(\mathcal{M}[\![Q]\!]\rho)/x]$$

where the binding $\rho[z/x]$ is defined as follows:

$$\rho[z/x][\![y]\!] \triangleq \begin{cases} z & \text{if } y = x \\ \rho[\![y]\!] & \text{otherwise} \end{cases}$$

We shall be predominately concerned with process terms whose meaning is the same with all possible variable bindings.

Definition 3.2 A process term is *closed* if it has no free variables. ◊

Lemma 3.3 *The meaning of a closed process term P is independent of the current binding. Formally, for P closed*

$$\forall \rho, \rho' : BIND \bullet \mathcal{M}[\![P]\!]\rho = \mathcal{M}[\![P]\!]\rho'$$

■

When reasoning about closed terms it is unnecessary to make the binding explicit.

The semantic function $\mathcal{M}$

Given a variable binding, $\mathcal{M}$ maps each process term to a pair representing the process alphabet and the set of traces of the process. We define α and $\mathcal{T}$ to be the natural projections onto the first and second components of this pair.

$$\mathcal{M}[\![P]\!]\rho = (\alpha[\![P]\!]\rho, \mathcal{T}[\![P]\!]\rho)$$

For a general process term both the alphabet and the set of traces of the process will depend upon the variable binding.

$$\alpha : SCSP \rightarrow BIND \rightarrow \mathbb{F}\,\Sigma.$$

$$\mathcal{T} : SCSP \rightarrow BIND \rightarrow SM_T.$$

Non-recursive processes

We define $\mathcal{M}$ over the non-recursive terms of SCSP by defining the two projections α and $\mathcal{T}$.

We take SCSP^0 to be the restriction of SCSP to non-recursive terms, that is terms with syntax:

$$P ::= \perp_A \mid x \mid P \sqcap P \mid [X \subseteq A \to P] \mid P \parallel P \mid P \setminus A \mid P[S]$$

Definition 3.3 The function α is defined as follows[†] over the syntax of SCSP^0.

$$\begin{aligned}
\alpha[\![\perp_A]\!]\rho &\triangleq A \\
\alpha[\![x]\!]\rho &\triangleq \pi_1 \rho[\![x]\!] \\
\alpha[\![P \sqcap Q]\!]\rho &\triangleq \alpha[\![P]\!]\rho \quad \text{if } \alpha[\![P]\!]\rho = \alpha[\![Q]\!]\rho \\
\alpha[\![[X \subseteq B \to P_X]]\!]\rho &\triangleq \alpha[\![P_B]\!]\rho \quad \text{if } \forall B' \subseteq B \bullet \alpha[\![P_{B'}]\!]\rho = \alpha[\![P_B]\!]\rho \\
\alpha[\![P \parallel Q]\!]\rho &\triangleq \alpha[\![P]\!]\rho \cup \alpha[\![Q]\!]\rho \\
\alpha[\![P \setminus B]\!]\rho &\triangleq \alpha[\![P]\!]\rho - B \quad \text{if } \alpha[\![P]\!]\rho \not\subseteq B \\
\alpha[\![P[S]]\!]\rho &\triangleq (\alpha[\![P]\!]\rho)[S]
\end{aligned}$$ ◊

Definition 3.4 The function $\mathcal{T}$ is defined as follows over the syntax of SCSP^0.

$$\begin{aligned}
\mathcal{T}[\![\perp_A]\!]\rho &\triangleq ST_A \\
\mathcal{T}[\![x]\!]\rho &\triangleq \pi_2 \rho[\![x]\!] \\
\mathcal{T}[\![P \sqcap Q]\!]\rho &\triangleq \mathcal{T}[\![P]\!]\rho \cup \mathcal{T}[\![Q]\!]\rho \\
\mathcal{T}[\![[X \subseteq B \to P_X]]\!]\rho &\triangleq \{\langle B' \cup C\rangle^\frown s \mid B' \subseteq B \wedge C \subseteq (\bar{A} - \bar{B}) \wedge s \in \mathcal{T}[\![P_{B'}]\!]\rho\} \\
&\quad \cup \{\langle\rangle\} \\
&\quad \text{where } A = \alpha[\![[X \subseteq B \to P_X]]\!]\rho
\end{aligned}$$

[†] π_n is the projection of an m-tuple onto its n^{th} component. So $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$

$$
\begin{aligned}
\mathcal{T}[\![P \parallel Q]\!]\rho \triangleq \{s \mid\ & \exists\, s_1, s_2 : (\mathbb{P}(\bar{A} \cap \bar{B}))^* \bullet s_1 \cap s_2 = \langle\{\}\rangle^{|s|} \\
& \wedge (s \cap \dot{A}) - s_1 \in \mathcal{T}[\![P]\!]\rho \wedge (s \cap \dot{B}) - s_2 \in \mathcal{T}[\![Q]\!]\rho\} \\
\cup\ & \\
\{s^\frown r \mid\ & \exists\, s_1, s_2 : (\mathbb{P}(\bar{A} \cap \bar{B}))^* \bullet s_1 \cap s_2 = \langle\{\}\rangle^{|s|} \\
& \wedge\ r \in ST_{A \cup B} \\
& \wedge\ ((((s \cap \dot{A}) - s_1)^\frown\langle \dot{A}\rangle \in \mathcal{T}[\![P]\!]\rho \\
& \qquad\qquad \wedge (s \cap \dot{B}) - s_2 \in \mathcal{T}[\![Q]\!]\rho) \\
& \quad \vee \\
& \quad (((s \cap \dot{B}) - s_2)^\frown\langle \dot{B}\rangle \in \mathcal{T}[\![Q]\!]\rho \\
& \qquad\qquad \wedge (s \cap \dot{A}) - s_1 \in \mathcal{T}[\![P]\!]\rho))\} \\
& \text{where } A = \alpha[\![P]\!]\rho \text{ and } B = \alpha[\![Q]\!]\rho
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{T}[\![P \setminus B]\!]\rho \triangleq \{s - \dot{B} \mid\ & s \in \mathcal{T}[\![P]\!]\rho \wedge \mathit{saturated}_{A \cap B}(s)\} \\
& \text{where } A = \alpha[\![P]\!]\rho
\end{aligned}
$$

$$
\mathcal{T}[\![P[S]]\!]\rho \triangleq \{s \mid s[S^{-1}] \in \mathcal{T}[\![P]\!]\rho\} \qquad \Diamond
$$

Notes

1. $\perp_A$ is modelled by the least element in the partial order $(SM^A, \sqsubseteq)$. This corresponds to $\perp_A$ being the least predictable process with alphabet A.

2. The non-deterministic choice operator is closely related to the ordering, $\sqsubseteq$, on the model. We can define the ordering in terms of this operator.

   $$\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!] \Leftrightarrow \mathcal{M}[\![P \sqcap Q]\!] = \mathcal{M}[\![P]\!]$$

   This relationship mirrors the fact that the ordering is a measure of non-determinism exhibited by processes.

3. The semantic function for parallel composition is by far the most complex and it is worth noting how its construction relates to the description of the operator.

   The first set (in the definition of $\mathcal{T}[\![P \parallel Q]\!]\rho$) gives the behaviours of $P \parallel Q$ obtained while both P and Q are not divergent. Each behaviour of $P \parallel Q$ is the result of the interaction of behaviours of P and Q. We recall that an event can only occur if all component processes with that event in their alphabet can perform the event, whereas an event is refused if either one of the component processes with that event in its alphabet refuses the event. So given a behaviour, s, of $P \parallel Q$ we should be able to find traces of refusals s_1 and s_2 in the common alphabet which represent those refusals from s solely due to Q and P respectively. Note that s_1 and s_2 are disjoint. Given this

choice of s_1 and s_2, $(s \cap \check{A}) - s_1$ must be a behaviour of P and $(S \cap \check{B}) - s_2$ must be a behaviour of Q.

The second set gives the behaviours of $P \parallel Q$ which result once one of the components has become divergent. The term $s^\frown r$ is composed of the behaviour up to divergence, s, which is governed as before, and arbitrary behaviour, r, following divergence.

4. In defining the hiding construct we assume that a hidden event occurs as soon as the process is ready. To capture this assumption in the semantic function we only consider behaviours of P which are saturated with respect to those events which are to be hidden in the alphabet of P. These behaviours correspond to the hidden events being offered by the environment for the duration of observation; thus they correspond to the hidden events occurring as soon as the process P is willing to perform them.

Theorem 3.4 *The terms of SCSP^0 are well defined with respect to the model.*

Proof: It is necessary and sufficient to establish that $\mathcal{M}[\![P]\!]\rho \in SM$ for all process expressions P in SCSP^0. This is achieved by structural induction over the syntax.

atomic terms It is clear by construction that $\perp_A$ is well defined with respect to the model. Moreover by definition of ρ, and since $\mathcal{M}[\![x]\!]\rho = \rho[\![x]\!]$ the process variables are also well defined.

operators We shall consider the two place operator $\|$; other operators follow in a similar manner. Assuming $\mathcal{M}[\![P]\!]\rho, \mathcal{M}[\![Q]\!]\rho \in SM$ we deduce that $\mathcal{M}[\![P \parallel Q]\!]\rho \in SM$. This requires us to check that $\mathcal{T}[\![P \parallel Q]\!]\rho$ satisfies the closure conditions for the alphabet $\alpha[\![P \parallel Q]\!]$. Conditions i-v and vii are easily verified; due to condition vi exhibiting a choice this is the hardest to verify. We present the proof that $\mathcal{T}[\![P \parallel Q]\!]\rho$ satisfies condition vi as Theorem A.1 in Appendix A.1. □

Recursive processes

Definition 3.5 We extend the definition of $\mathcal{M}$ to the full syntax of SCSP as

follows.

$$\begin{aligned}
\mathcal{M}[\![\mu\, x : A \bullet P]\!]\rho &\triangleq \underline{\mathrm{fix}}_A\, \lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x] \\
&\quad \text{where } y \text{ does not occur free in } P \text{ and } \underline{\mathrm{fix}}_A \\
&\quad \text{denotes the function's least fixed} \\
&\quad \text{point in } (SM^A, \sqsubseteq) \\
\mathcal{M}[\![\langle x_i \triangleq P_i\rangle_j \text{ with } A]\!]\rho &\triangleq (\underline{\mathrm{fix}}_{\underline{A}}\, \lambda\, \underline{y} \bullet \mathcal{M}[\![\underline{P}]\!]\rho[\underline{y}/\underline{x}])_j \\
&\quad \text{where for all } i,j \text{ the } y_i \text{ are not free in } P_j \text{ and} \\
&\quad \underline{\mathrm{fix}}_{\underline{A}} \text{ denotes the function's least fixed point} \\
&\quad \text{in the product c.p.o. } \textstyle\prod_i (SM^A, \sqsubseteq)
\end{aligned}$$

◊

In order to establish that $\mathcal{M}$ is well defined over $SCSP$ we must ensure that the least fixed points utilised in the above definition exist.

Lemma 3.5 *For P a recursion free process term, $\lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$ is continuous.*

Proof: We must establish that

$$\mathcal{M}[\![P]\!]\rho[\sqcup D/x] = \sqcup_{d \in D} \mathcal{M}[\![P]\!]\rho[d/x]$$

for D a directed set in $(SM^A, \sqsubseteq)$.

The continuity of atomic processes follows trivially in the case of $\perp_A$ and is a direct consequence of the definition of variable bindings in the case of process variables.

It is sufficient to check that each operator is continuous in each argument. The required result then follows from the continuity of finite compositions of continuous functions [Sto77].

The continuity of most of the operators follows from the distributivity of union through arbitrary intersection. The proofs for parallel composition and hiding are slightly more interesting. They are of a similar form and that for hiding is presented as Theorem A.2 in Appendix A.1. □

Lemma 3.6 *For y not free in P and $\lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$ continuous in $(SM^A, \sqsubseteq)$ for all $x \in Var$, then $\mathcal{M}[\![\mu\, x : A \bullet P]\!]$ is well defined and $\lambda\, y \bullet \mathcal{M}[\![\mu\, x : A \bullet P]\!]\rho[y/z]$ is continuous in $(SM^A, \sqsubseteq)$.*

Proof: $(SM^A, \sqsubseteq)$ is a complete semi-lattice and $\lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$ is, by assumption, continuous within the semi-lattice. So, appealing to the Knaster-Tarski Fixed Point Theorem [Tar55], a least fixed point exists. Hence $\underline{\mathrm{fix}}\, \lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$ is well defined.

Moreover, setting $H \triangleq \lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$, the least fixed point is given by the limit, $\bigsqcup_{n=0}^{\infty} H^n(\mathcal{M}[\![\ \perp_A\ ]\!])$.

As l.u.b. preserves continuity the required result holds. □

Lemma 3.7 *Let Λ be the totally ordered finite indexing set over which $(x_i \triangleq P_i)_j$ with A is defined. If, for each $i \in \Lambda$, $\lambda\, y \bullet \mathcal{M}[\![P_i]\!]\rho[y/x]$ is continuous in $(SM^A, \sqsubseteq)$ for all variables x and y is not free in P_i, then $\mathcal{M}[\![(x_i \triangleq P_i)_j \text{ with } A]\!]$ exists and $\lambda\, y \bullet \mathcal{M}[\![(x_i \triangleq P_i)_j]\!]\rho[y/z]$ is continuous in $(SM^A, \sqsubseteq)$.*

Proof: For each $i \in \Lambda$ we construct

$$f_i : \prod_{j \in \Lambda} SM^A \to SM^A$$

such that

$$f_i = \lambda\, \underline{z} \bullet \mathcal{M}[\![P_i]\!]\rho[\underline{z}/\underline{x}]$$

f_i is continuous since by hypothesis it is continuous in each of its finite number of arguments.

From this we define

$$\underset{\sim}{f} : \prod_{j \in \Lambda} SM^A \to \prod_{k \in \Lambda} SM^A$$

such that $\underset{\sim}{f}_i \triangleq f_i$. This function is continuous by construction. Now it is easy to verify that

$$\underset{\sim}{f} = \lambda\, \underline{y} \bullet \mathcal{M}[\![\underline{P}]\!]\rho[\underline{y}/\underline{x}]$$

So $\underset{\sim}{f}$ is the function for which we require a least fixed point. Following the reasoning of the previous lemma we have the required results. □

Theorem 3.8 *All process terms P of SCSP are well defined with respect to the model and $\lambda\, y \bullet \mathcal{M}[\![P]\!]\rho[y/x]$ is continuous.*

Proof: This follows from the previous lemmata. □

Corollary 3.9 *The operators of SCSP are monotonic with respect to the ordering $\sqsubseteq$.* ■

3.3 Expressivity of the Language

By defining a semantic function from SCSP to the model we have established a meaning in the model for every process expression. Conversely we now seek to establish that every process in the model, which exhibits infeasible behaviour after finite time, can be represented by a finite expression in the language.

Given an such a process (A, T) in the model, we shall construct a closed process expression $P(T)$ in the algebra which is mapped to (A, T) by the semantic function $\mathcal{M}$. In order to build our process expression it will be necessary to consider the process $(A, T/B)$ which is a process which behaves like (A, T) after the set B has been observed.

Definition 3.6 For $T \in SM_T$ and $\langle B \rangle \in T$ we define

$$T/B \triangleq \{s \mid \langle B \rangle^\frown s \in T\}.$$

$\Diamond$

Lemma 3.10 *If $(A, T) \in SM$ and $\langle B \rangle \in T$ then $(A, T/B) \in SM$.*

Proof: It is a trivial exercise to verify that T/B satisfies the closure conditions. $\square$

Theorem 3.11 *Let (A, T) be a process in the model which exhibits infeasible behaviour after a finite time m, that is $\forall s \in T \bullet |s| \geq m \Rightarrow s^\frown\langle \tilde{A} \rangle \in T$. Then (A, T) is denoted by a closed term in the language SCSP.*

Proof: We must construct a term P with alphabet A such that $\mathcal{T}[\![P]\!] = T$. Following the approach taken by Jeffrey [Jef91a], for (A, T) in the model we consider the following definition of a process $P(T)$

$$P(T) = \begin{cases} \perp_A & \text{if } \langle \tilde{A} \rangle \in T \\ \prod_{D \in \mathcal{D}} [X \subseteq D \cap A \rightarrow P(T/(X \cup (D \cap \tilde{A})))] & \text{otherwise} \end{cases}$$

where $\mathcal{D} = \{B \mid saturated_A(\langle B \rangle) \wedge \langle B \rangle \in T\}$ is finite.

Note that $\mathcal{D}$ represents the set of initial observations containing maximal (total) information about the events in the alphabet A. We shall prove that $\mathcal{T}[\![P(T)]\!] = T$ by induction on m, the time after which infeasible behaviour must result. We take as our inductive hypothesis:

$\forall (A, T) \in SM \bullet (\forall s \in T \bullet |s| \geq m \Rightarrow s^\frown\langle \tilde{A} \rangle \in T) \Rightarrow \mathcal{T}[\![P(T)]\!] = T.$

base case m = 0

$\langle \tilde{A} \rangle \in T$ thus $P(T) = \perp_A$ and by closure condition vii and the definition of $\mathcal{T}$, $T = ST_A = \mathcal{T}[\![\perp_A]\!]$

inductive step

case 1: $P(T) = \perp_A$

The result follow as for the base case since $\langle \tilde{A} \rangle \in T$

case 2: $P(T)$ takes the form of the non-deterministic choice between set-prefixed processes.

Notice that for $D \in \mathcal{D}$, by construction, we have that $\langle X \cup (D \cap \bar{A}) \rangle \in T$ for all $X \subseteq D \cap A$. So by the previous lemma $(A, T/(X \cup (D \cap \bar{A}))) \in SM$.

We shall use $P_{D,X}$ to denote $P(T/(X \cup (D \cap \bar{A})))$ for all $X \subseteq D \cap A$.

Assuming $(\forall s \in T \bullet |s| \geq m + 1 \Rightarrow s^\frown\langle \tilde{A} \rangle \in T)$ we must show $T = \mathcal{T}[\![P(T)]\!]$.

Firstly note by considering the definition of $T/(X \cup (D \cap \bar{A}))$ we can show

$|s| \geq m \wedge s \in T/(X \cup (D \cap \bar{A})) \Rightarrow s^\frown\langle \tilde{A} \rangle \in T/(X \cup (D \cap \bar{A}))$

Hence by induction $\mathcal{T}[\![P_{D,X}]\!] = T/(X \cup (D \cap \bar{A}))$.

Now we can demonstrate that $T = \mathcal{T}[\![P(T)]\!]$. Trivially $\langle\rangle \in T \wedge \langle\rangle \in \mathcal{T}[\![P(T)]\!]$. It remains to show $\langle B \rangle^\frown s \in T \Leftrightarrow \langle B \rangle^\frown s \in \mathcal{T}[\![P(T)]\!]$.

We shall use $B^E \mathrel{\hat{=}} B \cap A$ and $B^R \mathrel{\hat{=}} B \cap \bar{A}$ to denote the events and refusals in B.

$\Rightarrow$: Suppose $\langle B \rangle^\frown s \in T$ then setting $Y \mathrel{\hat{=}} \{a \in A \mid a \notin B \wedge \bar{a} \notin B\}$ we can establish from closure condition vi and induction over $|Y|$ that

$$\begin{aligned}
&\exists X^E \subseteq Y, X^R \subseteq \bar{Y} \bullet \mathit{saturated}_A(\langle X^E \cup X^R \cup B \rangle) \wedge \langle X^E \cup X^R \cup B \rangle \in T \\
&\qquad\qquad \wedge \langle X^R \cup B \rangle^\frown s \in T \\
\Rightarrow\ & \{ \text{ by definition of } \mathcal{D} \ \} \\
&\exists D \in \mathcal{D} \bullet B \subseteq D \wedge \langle (D \cap \bar{A}) \cup B^E \rangle^\frown s \in T \\
\Rightarrow\ & \{ \text{ construction of } T/((D \cap \bar{A}) \cup B^E) \ \} \\
&\exists D \in \mathcal{D} \bullet B \subseteq D \wedge s \in T/((D \cap \bar{A}) \cup B^E) \\
\Rightarrow\ & \{ \text{ by the inductive hypothesis as noted above } \} \\
&\exists D \in \mathcal{D} \bullet B \subseteq D \wedge s \in \mathcal{T}[\![P_{D,B^E}]\!] \\
\Rightarrow\ & \{ \text{ by set manipulation and as } D \text{ is saturated } \} \\
&\exists D \in \mathcal{D} \bullet B^R \subseteq \bar{A} - \overline{(D \cap A)} \wedge B^E \subseteq (D \cap A) \wedge s \in \mathcal{T}[\![P_{D,B^E}]\!] \\
\Rightarrow\ & \{ \text{ by definition of set-prefix } \} \\
&\exists D \in \mathcal{D} \bullet \langle B^E \cup B^R \rangle^\frown s \in \mathcal{T}[\![(X \subseteq D \cap A \rightarrow P_{D,X})]\!] \\
\Rightarrow\ & \{ \text{ by definition of non-deterministic choice } \} \\
&\langle B^E \cup B^R \rangle^\frown s \in \mathcal{T}[\![\textstyle\bigsqcap_{D \in \mathcal{D}} (X \subseteq D \cap A \rightarrow P_{D,X})]\!] \\
\Rightarrow\ & \{ \text{ by definition of } T(P) \ \} \\
&\langle B \rangle^\frown s \in \mathcal{T}[\![P(T)]\!]
\end{aligned}$$

The converse follows similarly giving the reqnired result. □

3.4 A Sound and Complete Algebra

In this section we introduce a proof system for our language which is sound and complete for equivaleuce of processes in the semantic model introduced in Section 3.1. In a style similar to Brookes [Bro83] we consider the suhlangnage SCSP^{l} first. This sublanguage is restricted to the non-recursive closed terms of SCSP. The logical language will consist of assertions of the form $P \sqsubseteq Q$ and $P \equiv Q$. We give a set of axioms and inference rules for proving assertions, and show that the system is both sound and complete.

We extend the result to the full language SCSP, allowing recursive terms.

3.4.1 The sublanguage SCSP^{l}

The syntax of SCSP^{l} is given by:

$$P ::= \bot_A \mid P \sqcap P \mid [X \subseteq A \to P] \mid P \parallel P \mid P \setminus A \mid P[S]$$

This language is a sublanguage of SCSP^{0} resulting from the elimination of free variables. We refer the reader to Section 3.2 for the definition of the semantic function $\mathcal{M}$ over terms generated by the above syntax.

The logical language of our proof system is built from SCSP^{l} terms and two binary relation symbols $\sqsubseteq$ and $\equiv$ (which can be defined over the fnll language SCSP). Formulae in the language take the form $P \sqsubseteq Q$ or $P \equiv Q$. We take $P \sqsubseteq Q$ to mean

$$\forall \rho \in BIND \bullet \mathcal{M}[\![P]\!]\rho \sqsubseteq \mathcal{M}[\![Q]\!]\rho$$

and $P \equiv Q$ to mean

$$\forall \rho \in BIND \bullet \mathcal{M}[\![P]\!]\rho = \mathcal{M}[\![Q]\!]\rho.$$

The axioms of the system are given in Appendix B.1. These include many of the equivalences already stated. A further axiom relates the non-deterministic choice operator to the relation $\sqsubseteq$. Inference rules are included which establish $\sqsubseteq$ to be a partial order with $\equiv$ the associated equivalence. Finally there are inference rules asserting the monotonicity of the operators.

Several simple properties of $\sqsubseteq$ and $\equiv$ are not explicitly given in the axiom system as they are deducible from the axioms. These results include

O–5 $\quad \vdash P \equiv P$ $\qquad$ **O–6** $\quad \vdash \bot \sqsubseteq P$

O–7 $\quad \dfrac{P \sqcap Q \equiv P}{P \sqsubseteq Q}$ $\qquad$ **O–8** $\quad \dfrac{P \sqsubseteq Q}{P \sqcap Q \equiv P}$

O–7 and O–8 demonstrate the link between non-deterministic choice and the ordering. O–6 states that $\perp$ is the least element in the ordering. We shall present the proofs for O–5 and O–8.

O–5:

$$\dfrac{\dfrac{\dfrac{\vdash\ P \sqcap P \equiv P}{P \sqsubseteq P \sqcap P \sqsubseteq P}\ \text{O–3}}{P \sqsubseteq P}\ \text{O–4}}{P \equiv P}\ \text{O–2}$$

O–8:

$$\dfrac{\dfrac{P \sqsubseteq Q \qquad \dfrac{\vdash\ P \equiv P}{P \sqsubseteq P}\ \text{O–3}}{P \sqcap P \sqsubseteq P \sqcap Q}\ \text{M–1} \qquad \dfrac{\vdash\ P \sqcap P \equiv P}{P \sqsubseteq P \sqcap P}\ \text{O–3}}{P \sqsubseteq P \sqcap Q}\ \text{O–4} \qquad \dfrac{\vdash\ P \sqcap Q \sqsubseteq P}{P \sqcap Q \equiv P}\ \text{O–2}$$

□

Soundness

In order to establish soundness of the proof system we must ensure that every provable assertion is true. It is necessary to verify the truth of each of the axioms of our system. These checks are trivial and unenlightening, so are omitted here. The inference rules stating that $\sqsubseteq$ is a partial order follow from the structure of $(SM, \sqsubseteq)$. As all the operators were shown to be monotone the rules M–1 and M–2 follow.

If we write $\vdash\ P \sqsubseteq Q$ to assert that $P \sqsubseteq Q$ is provable then the following theorem states that the proof system is sound.

Theorem 3.12 (Soundness) *For all terms P, Q in $SCSP^1$*

$$(\vdash\ P \sqsubseteq Q) \Rightarrow \mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$$

■

Completeness

In order to establish completeness of a system we must show that whenever an assertion is true, it is provable. We must show that whenever $\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$ then the formula $P \sqsubseteq Q$ is provable. We shall define a class of normal forms

and show that every term is provably equivalent to a unique normal form. Finally we establish that the system consisting of the class of normal form processes is complete.

Normal form

A normal form is a term in the language with a specific structure. $\perp_A$ is a normal form. All other normal forms are the non-deterministic choice between a finite number of set-prefixed processes, where each of the set-prefixed processes is in normal form. The choice sets in the set-prefix constructs are unique. This and a further condition insisting that the set prefixed processes are as non-determiuistic as possible ensure that the normal forms are unique.

Definition 3.7 We say a process, P, with alphabet A, is in *normal form* if it is $\perp_A$ or takes the form

$$P = \prod_{B \in \mathcal{B}} [X \subseteq B \rightarrow P_{B,X}]$$

where

- $\mathcal{B}$ is a non-empty, finite subset of $\mathbb{P}\, A$
- $\forall B, B' \in \mathcal{B} \bullet X \subseteq B \subseteq B' \Rightarrow P_{B',X} \sqsubseteq P_{B,X}$
- For all $B \in \mathcal{B}$ and $X \subseteq B$, $P_{B,X}$ is in normal form.

$\Diamond$

By the second condition we ensure that the set-prefixed processes are as non-deterministic as possible. This reflects A–5 and the ability to postpone the resolution of choice; such postponement can occur exactly when the observation of a given set of events must be the result of the environment's initial offer of events being insufficient to resolve the choice instantly.

Rather than prove directly that every process is provably equivalent to a unique normal form, we shall first consider the class of processes in pre-normal form. The pre-normal form is similar to the normal form; the structure of the processes is the same, although with fewer restrictions on the construction, uniqueness cannot be guaranteed. We shall demonstrate that each process in pre-normal form is provably equivalent to a process in normal form. It is then sufficient to show that each general process is provably equivalent to a process in pre-normal form.

Definition 3.8 We say a process, P, with alphabet A, is in *pre-normal form* if:

$$P = \begin{cases} \perp_A \\ \prod_{i \in I} [X \subseteq B_i \rightarrow P_{i,X}] \end{cases}$$

where

- I is a non-empty, finite indexing set and $\{B_i \mid i \in I\} \subseteq \mathbb{P}\, A$.
- Each $P_{i,X}$ is in pre-normal form.

$\Diamond$

Notice that we no longer insist on the uniqueness of the choice-sets B_i.

Lemma 3.13 *Every pre-normal form in $SCSP^I$ is provably equivalent to a process in normal form.*

Proof: We define the depth of a process in pre-normal form as follows:

$$\begin{aligned} d(\bot) &= 0 \\ d(\textstyle\prod_{i\in I} P_i) &= \max_{i\in I} d(P_i) \\ d([X \subseteq B \rightarrow P_X]) &= \max_{X\subseteq B} d(P_X) + 1 \end{aligned}$$

We shall prove the required result by induction on the depth of processes.

Base case: $d(P) = 0$

Trivial since the only possible pre-normal form with zero depth is $\bot$ which is in normal form.

Inductive step: $d(P) = n+1$

P is in pre-normal form

$$P = \prod_{i\in I} [X \subseteq B_i \rightarrow P_{i,X}],$$

by applications of A–5, A–4 and A–3 we have

$$\vdash \quad P \equiv \prod_{i\in I} [X \subseteq B_i \rightarrow P'_{i,X}]$$

where $P'_{i,X} = \begin{cases} \bot & \text{if } P_{j,X} = \bot \text{ for any } j \in \{k \mid B_k \subseteq B_i\} \\ \prod_{B_j \subseteq B_i} P_{j,X} & \text{otherwise} \end{cases}$

By the construction of the $P'_{i,X}$ they are all in pre-normal form. By the idempotence of non-deterministic choice (A–3) we can remove all duplicate set-prefixed terms. So we can assume that the B_i are unique. Now

$$\begin{aligned} d(P'_{i,X}) &\leq \max_{B_j\subseteq B_i} d(P_{j,X}) \leq \max_{i\in I} \max_{X\subseteq B_i} d(P_{i,X}) \\ &< \max_{i\in I}(\max_{X\subseteq B_i} d(P_{i,X}) + 1) = d(P) \end{aligned}$$

so we have that $d(P'_{i,X}) < d(P)$. So since $d(P'_{i,X}) < d(P)$ we can find, by induction, $Q_{i,X}$ in normal form such that $Q_{i,X} \equiv P'_{i,X}$ is provable. Then clearly

$$\vdash \quad P \equiv \prod_{i \in I} [X \subseteq B_i \rightarrow Q_{i,X}]$$

with the right hand side being in normal form. □

Lemma 3.14 *Every process expression P, in SCSP^I is provably equivalent to a process in pre-normal form.*

Proof: We define a rank function l on finite processes as follows:

$$\begin{aligned} l(\bot) &= 0 \\ l(P_1 \sqcap P_2) &= l(P_1) + l(P_2) + 1 \\ l(P_1 \parallel P_2) &= l(P_1).l(P_2) + 1 \\ l([X \subseteq B \rightarrow P_X]) &= \max_{X \subseteq B} l(P_X) + 1 \\ l(P \setminus A) &= 2.l(P) + 1 \\ l(P[S]) &= 3.l(P) + 1 \end{aligned}$$

We shall prove by induction on the rank of P, $l(P)$, that all processes in SCSP^I are provably equivalent to a process in pre-normal form with rank no greater than that of P. So we take as our inductive hypothesis:

$$\begin{aligned} l(P) = n \Rightarrow \exists\, Q \bullet\ & Q \text{ is in pre-normal form} \\ & \wedge\ l(Q) \leq l(P) \\ & \wedge \vdash \quad Q \equiv P. \end{aligned}$$

Base case: $n = 0$

We must have $P = \bot$, so P is already in pre-normal form and we are done.

Inductive step: $l(P) = n+1$

P must take the form of an operator on component processes. We shall only present the proof for parallel composition here, proofs for the other operators being similar.

$P = P_1 \parallel P_2$

By definition of the rank function, $l(P) = l(P_1).l(P_2) + 1$.

Either $l(P) = 1$ in which case one of P_1 or P_2 is $\bot$, so by A–6 or A–7 we have that $\vdash \quad P \equiv \bot$ and we are done.

Alternatively $l(P) > 1$, in which case neither of P_1 nor P_2 is $\bot$ and we must have $l(P_i) < l(P)$ for $i \in \{1, 2\}$.

So by the inductive hypothesis we can find Q_i in pre-normal form such that $l(Q_i) \leq l(P_i)$ and $\vdash \ P_i \equiv Q_i$.

By monotonicity $\vdash \ P \equiv Q_1 \parallel Q_2$.

Now either one of $Q_i = \perp$, in which case by A–6 or A–7, $\vdash \ P \equiv \perp$ and we are done. Or both Q_i take the form of a non-deterministic choice between set-prefixed processes.

$$Q_1 = \prod_{i \in I} Q_i^1 \text{ where } Q_i^1 = [Y \subseteq B_i^1 \rightarrow Q_{iY}^1]$$

$$Q_2 = \prod_{i \in J} Q_i^2 \text{ where } Q_i^2 = [Y \subseteq B_i^2 \rightarrow Q_{iY}^2]$$

Consider the case where $|I|$ or $|J|$ is greater than 1 and without loss of generality assume $|J| > 1$. Then by A–8

$\vdash \ Q_1 \parallel Q_2 \equiv \prod_{i \in I}(Q_i^1 \parallel Q_2)$

We can deduce that $l(Q_i^1 \parallel Q_2) < l(Q_1 \parallel Q_2) \leq l(P)$. So, by the inductive hypothesis we can find Q_i^0 in pre-normal form such that $\vdash \ Q_i^1 \parallel Q_2 \equiv Q_i^0$ and $l(Q_i^0) \leq l(Q_i^1 \parallel Q_2)$.

Thus $\vdash \ P \equiv \prod_{i \in I} Q_i^0$. Either one of the $Q_i^0 = \perp$, in which case as $\perp$ is a zero of non-deterministic choice, $\vdash \ P \equiv \perp$ and we are done. Otherwise $\prod_{i \in I} Q_i^0$ is in pre-normal form and it remains to establish that the rank of this process is no greater than the rank of P.

Consider the case when $l(Q_2) > 1$.

$$\begin{array}{lll}
l(\prod_{i \in I} Q_i^0) = & \sum_{i \in I} l(Q_i^0) + |I| - 1 & \{ \text{ by definition of } l \ \} \\
\leq & \sum_{i \in I} l(Q_i^1 \parallel Q_2) + |I| - 1 & \{ \ l(Q_i^0) \leq l(Q_i^1 \parallel Q_2) \ \} \\
= & \sum_{i \in I} (l(Q_i^1).l(Q_2) + 1) + |I| - 1 & \{ \text{ by definition of } l \ \} \\
\leq & l(Q_2).(\sum_{i \in I} l(Q_i^1) + |I| - 1) + 1 & \{ \text{ assuming } l(Q_2) \geq 2 \ \} \\
= & l(Q_2).l(Q_1) + 1 & \{ \text{ by definition of } l \text{ and } Q_1 \ \} \\
\leq & l(P_1).l(P_2) + 1 & \{ \ l(Q_i) \leq l(P_i) \ \} \\
= & l(P) & \{ \text{ by definition of } l \text{ and } P \ \}
\end{array}$$

In the case where $l(Q_2) = 1$ we must consider the form of Q_i^0 more closely. As Q_2 is in pre-normal form the only form this process can take is $[X \subseteq B^2 \rightarrow \perp]$. In which case, by A–10 and A–7,

$\vdash \ Q_i^1 \parallel Q_2 \equiv Q_i^0$

where $Q_i^0 = [Y \subseteq C_i \rightarrow \perp]$ and $C_i = (B_i^1 \cap B^2) \cup (B_i^1 - A_2) \cup (B_i - A_1)$, A_1 and A_2 being the alphabets of Q_1 and Q_2 respectively.

Clearly Q_i^θ is in pre-normal form and $l(Q_i^\theta) = 1$. Thus by simple analysis of terms we can show $l(\prod_{i \in I} Q_i^\theta) \leq l(P)$.

Hence setting $Q = \prod_{i \in I} Q_i^\theta$ we have the required result.

Finally we must consider the case where $|I| = |J| = 1$. So

$$Q_1 = [Y \subseteq B_1 \rightarrow Q_Y^1] \qquad Q_2 = [Y \subseteq B_2 \rightarrow Q_Y^2]$$

By the axiom for parallel composition of set-prefixed processes (A-10) we have

$\vdash \quad Q_1 \parallel Q_2 \equiv [Y \subseteq C \rightarrow (Q^1_{Y \cap B_1} \parallel Q^2_{Y \cap B_2})]$

where $C = (B_1 \cap B_2) \cup (B_1 - A_2) \cup (B_2 - A_1)$ with A_1 and A_2 the alphabets of Q_1 and Q_2 respectively.

Now we can deduce that

$l(Q^1_{Y \cap B_1} \parallel Q^2_{Y \cap B_2}) < l(P)$. Thus by the inductive hypothesis, we can find Q_Y^θ in pre-normal form such that

$\vdash \quad Q_Y^\theta \equiv Q^1_{Y \cap B_1} \parallel Q^2_{Y \cap B_2}$ and $l(Q_Y^\theta) \leq l(Q^1_{Y \cap B_1} \parallel Q^2_{Y \cap B_2})$.

Setting $Q = [Y \subseteq C \rightarrow Q_Y^\theta]$, clearly Q is in pre-normal form and $\vdash \quad P \equiv Q$. Analysis of expressions gives $l(Q) \leq l(P)$, so we are done. □

Corollary 3.15 *Every process expression, P, in $SCSP^l$ is provably equivalent to a process in normal form.*

Proof: This follows from the previons two lemmata. □

Finally we show that the class of normal forms is complete.

Lemma 3.16 *For P and Q, with alphabet A, in normal form*

$$\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!] \Rightarrow (\vdash \quad P \sqsubseteq Q).$$

Proof: By structural induction on P.

base case: $P = \perp$

Then the result follows by O-6 (page 43).

inductive step: P takes the form of a nondeterministic choice.

Now $\perp$ is the only process with an initial infeasible event, so $\langle \tilde{A} \rangle \notin \mathcal{T}[\![P]\!]$ hence $\langle \tilde{A} \rangle \notin \mathcal{T}[\![Q]\!]$ and Q must also take the form of a non-deterministic choice.

$$P = \prod_{B \in \mathcal{B}} [X \subseteq B \rightarrow P_{B,X}]$$

where $\forall B, B' \in \mathcal{B} \bullet B \subseteq B' \Rightarrow (\forall X \subseteq B \bullet \mathcal{M}[\![P_{B',X}]\!] \sqsubseteq \mathcal{M}[\![P_{B,X}]\!])$

and

$$Q = \prod_{C \in \mathcal{C}} [X \subseteq C \rightarrow Q_{C,X}]$$

where $\forall C, C' \in \mathcal{C} \bullet C \subseteq C' \Rightarrow (\forall X \subseteq C \bullet \mathcal{M}[\![Q_{C',X}]\!] \sqsubseteq \mathcal{M}[\![Q_{C,X}]\!])$

Now by the definition of the semantic function,

$\forall C \in \mathcal{C} \bullet (C \cup (\bar{A} - \bar{C})) \in \mathcal{T}[\![Q]\!]$
$\Rightarrow$ { as $\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$ by hypothesis }
$\forall C \in \mathcal{C} \bullet (C \cup (\bar{A} - \bar{C})) \in \mathcal{T}[\![P]\!]$
$\Rightarrow$ { by definition of P and semantic function on non-deterministic choice }
$\forall C \in \mathcal{C} \bullet \exists B \in \mathcal{B} \bullet (C \cup (\bar{A} - \bar{C})) \in \mathcal{T}[\![[X \subseteq B \rightarrow P_{B,X}]]\!]$
$\Rightarrow$ { by definition of semantic function on set-prefix }
$\forall C \in \mathcal{C} \bullet \exists B \in \mathcal{B} \bullet C \subseteq B \wedge (\bar{A} - \bar{C}) \subseteq (\bar{A} - \bar{B})$
$\Rightarrow$ { by set manipulation }
$\mathcal{C} \subseteq \mathcal{B}$

Now by axiom O-1

$$\vdash \; P \sqsubseteq \prod_{C \in \mathcal{C}} [X \subseteq C \rightarrow P_{C,X}]$$

It is sufficient to show that, for all $C \in \mathcal{C}$, $\mathcal{M}[\![P_{C,X}]\!] \sqsubseteq \mathcal{M}[\![Q_{C,X}]\!]$. The result then follows from structural induction and monotonicity of the operators.

$s \in \mathcal{T}[\![Q_{C,X}]\!]$
$\Rightarrow$ { definition of set-prefix and non-deterministic choice }
$(X \cup (\bar{A} - \bar{C}))^\frown s \in \mathcal{T}[\![Q]\!]$
$\Rightarrow$ { as $\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$ by hypothesis }
$(X \cup (\bar{A} - \bar{C}))^\frown s \in \mathcal{T}[\![P]\!]$
$\Rightarrow$ { definition of non-deterministic choice and set-prefix }
$\exists B \in \mathcal{B} \bullet X \subseteq B \wedge (\bar{A} - \bar{C}) \subseteq (\bar{A} - \bar{B}) \wedge s \in \mathcal{T}[\![P_{B,X}]\!]$
$\Rightarrow$ { by set manipulation }
$\exists B \in \mathcal{B} \bullet X \subseteq B \subseteq C \wedge s \in \mathcal{T}[\![P_{B,X}]\!]$
$\Rightarrow$ { since P is in normal form }
$\exists B \in \mathcal{B} \bullet s \in \mathcal{T}[\![P_{B,X}]\!] \wedge \mathcal{T}[\![P_{B,X}]\!] \subseteq \mathcal{T}[\![P_{C,X}]\!]$
$\Rightarrow$ { by definition of subset }
$s \in \mathcal{T}[\![P_{C,X}]\!]$

Giving the required result. □

From the previous results it is trivial to deduce completeness.

Theorem 3.17 (Completeness) *For all terms P, Q in $SCSP^l$*

$$\mathcal{M}[\![P]\!] \subseteq \mathcal{M}[\![Q]\!] \Rightarrow (\vdash\ P \sqsubseteq Q)$$

■

3.4.2 An extended proof system

In this section we extend our proof system to cover the full language of closed terms in *SCSP*. Like Brookes [Bro83], we characterise each process by its set of finite syntactic approximations. This enables us to reason about an infinite process, that is one containing recursive constructs, by considering its finite approximations.

Definition 3.9 The relation $\prec$ is the smallest relation on terms satisfying:

$$\begin{array}{rcl}
\bot & \prec & P \\
P & \prec & P \\
P[(\mu x \bullet P)/x] & \prec & \mu x \bullet P \\
P_j[\langle x_i \mathrel{\hat{=}} P_i\rangle_k / x_k] & \prec & \langle x_i \mathrel{\hat{=}} P_i\rangle_j \\
P \prec Q \prec R & \Rightarrow & P \prec R \\
P_1 \prec Q_1, P_2 \prec Q_2 & \Rightarrow & (P_1 \sqcap P_2) \prec (Q_1 \sqcap Q_2) \\
\forall X \subseteq B \bullet P_X \prec Q_X & \Rightarrow & [X \subseteq B \rightarrow P_X] \prec [X \subseteq B \rightarrow Q_X] \\
P_1 \prec Q_1, P_2 \prec Q_2 & \Rightarrow & (P_1 \parallel P_2) \prec (Q_1 \parallel Q_2) \\
P \prec Q & \Rightarrow & (P \setminus A) \prec (Q \setminus A) \\
P \prec Q & \Rightarrow & P[S] \prec Q[S]
\end{array}$$

If $P \prec Q$ then we say that P is a *syntactic approximation* of Q. ◊

It can be shown by simple structural induction that if P is a syntactic approximation of Q then Q is more deterministic than P.

$$P \prec Q \Rightarrow \mathcal{M}[\![P]\!] \subseteq \mathcal{M}[\![Q]\!]$$

Given a closed process P, we construct the set of its finite syntactic approximations $FIN(P)$. In this context we say a process is finite precisely when it is a term in the language SCSP^l. So the formal definition of $FIN(P)$ is given by:

Definition 3.10 $\quad FIN(P) \mathrel{\hat{=}} \{Q \in \text{SCSP}^l \mid Q \prec P\}$ ◊

$FIN(P)$ forms a directed set under $\prec$ and consequently the semantic image of the set forms a directed set under $\subseteq$, with $\mathcal{M}[\![P]\!]$ an upper bound. It can be established that every finite process, Q, which is less deterministic than P is less deterministic than some $P' \in FIN(P)$. Intuitively this follows since every finite process must behave like chaos after a finite time. Suppose Q degenerates to chaos after n time units. By choosing any finite syntactic approximation P' of P which has the same behaviour as P until time n we are guaranteed $Q \sqsubseteq P'$. This gives us the result.

Lemma 3.18 *If $Q \in SCSP^1$ and $\mathcal{M}[\![Q]\!] \sqsubseteq \mathcal{M}[\![P]\!]$, then there exists $P' \in FIN(P)$ such that $\mathcal{M}[\![Q]\!] \sqsubseteq \mathcal{M}[\![P']\!]$.* ■

The semantic model consists solely of finite traces. We have already seen the semantics of a recursive process constructed as the limit of the semantics of a chain of finite approximations to that process. More generally the semantics of any process, P, can be described as the least upper bound of the directed set of the semantics of certain finite syntactic approximations. By the above lemma the least upper bound of this set must lie below that of the semantic image of $FIN(P)$. Thus we have

Theorem 3.19 $\mathcal{T}[\![P]\!] = \bigcap_{Q \in FIN(P)} \mathcal{T}[\![Q]\!]$ ■

Extended proof system

We extend the proof system of Section 3.4.1 with the following:

A–17 $\quad \vdash P[(\mu x \bullet P)/x] \equiv \mu x \bullet P$

A–18 $\quad \vdash P_j[\langle x_i \triangleq P_i \rangle_k / x_k] \equiv \langle x_i \triangleq P_i \rangle_j$

R–1 $\quad \dfrac{\forall Q \in FIN(P) \bullet Q \sqsubseteq R}{P \sqsubseteq R}$

The inference rule captures the fact that P is the least fixed point of its set of finite syntactic approximants. It should be observed that the inference rule is an infinitary rule as $FIN(P)$ may be an infinite set. We would not expect to be able to construct a decidable proof system for a language which is Turing machine equivalent.

Soundness and completeness

The least fixed point construction of the semantics of recursive constructs guarantees the soundness of axioms A–17 and A–18. While the inference rule is sound by Theorem 3.19.

Theorem 3.20 (Soundness) *For all closed terms P and Q in SCSP*

$$(\vdash P \sqsubseteq Q) \Rightarrow \mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$$

■

Completeness is established by considering the characterisation of a process by its syntactic approximation.

Theorem 3.21 (Completeness) *For all closed terms P and Q in SCSP*

$$\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!] \Rightarrow (\vdash\ P \sqsubseteq Q)$$

Proof: Suppose $\mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$.

Let $P' \in FIN(P)$. Then $\mathcal{M}[\![P']\!] \sqsubseteq \mathcal{M}[\![P]\!] \sqsubseteq \mathcal{M}[\![Q]\!]$.

So by Lemma 3.18 we can find $Q' \in FIN(Q)$ such that $\mathcal{M}[\![P']\!] \sqsubseteq \mathcal{M}[\![Q']\!]$.

By Theorem 3.17 we have $\vdash\ P' \sqsubseteq Q'$.

Now $Q' \sqsubseteq Q$ is provable for every $Q' \in FIN(Q)$, so $\vdash\ P' \sqsubseteq Q$.

Hence by the inference rule R–1 the result follows. □

3.5 Conclusion

The semantic model for SCSP presented in this chapter has provided a mathematical underpinning of the language in Chapter 2. The image of a process expression in the language, under the semantic function $\mathcal{M}$, gives a denotational interpretation of the process in the model. The model captures the behaviours of processes by recording failures-divergences information comparable to that used in [BR85] for CSP; by incorporating refusals into the trace of events and introducing the concept of infeasible behaviour it was possible to provide an elegant mechanism for recording divergences within the traces. The traces record time implicitly; simultaneously occurring events are recorded in a single set, the position of this set in the trace represents the time at which it was observed. Moreover the model has sufficient mathematical structure for domain theoretic results to be applicable, giving a formal underpinning of recursion.

Finally, a sound and complete proof system for SCSP has been developed. In verifying that the proof system was complete, a normal form for closed terms in the language was considered. The proof system enables relationships between processes, deducible from the semantic model, to be established directly, using axioms, within the algebra.

Chapter 4

Communication and Protocols

So far any interaction between processes has been by their cooperation over the performance of common events. In this chapter we consider how we can model the concept of communication of data via channels within a system.

In the second part of this chapter we utilise the notation developed for communication in order to specify a token ring protocol in SCSP.

4.1 Communication

A channel is seen as a medium for the communication of data between processes. The data carried by a channel may take a finite number of values. We stipulate that the channels are uni-directional so a process will use a given channel for input or output exclusively for all time. Also, only one process in a system will use a given channel for output.

Like CSP, we consider $c.v$, the value v communicated on channel c, as being an atomic event. Within our model atomic events should be independent in the sense that the occurrence of one should not be able to affect the ability of another to be performed at the same time. We must be aware that $c.v$ and $c.w$ are distinct atomic events and as such it is possible to describe processes which allow these events to be performed simultaneously without chaotic consequences. Clearly such an occurrence would have little meaning in the context in which these events were intended. The notation we will provide simply ensures a disciplined use of the events, so that processes, which purport to model communication along channels, conform to the expected behaviour.

Output channels

If channel c is an output channel for a process, then whenever the process is willing to output data we would expect the process to select a particular data value, v, say.

The process should only make available this chosen data value for communication to the environment. Moreover any attempt to send several data values along a channel at a given time would, in general, have catastrophic consequences.

We use the notation $c!v$ to indicate that the occurrence of the event $c.v$ should be interpreted as the process outputting data along channel c. So

$$[X \subseteq \{c!v\} \rightarrow (P \textbf{ if } X = \{\} \textbf{ else } Q)]$$

will represent a process which may initially output value v on channel c and then go on to behave like Q or like P depending on whether the output occurred.

Input channels

If channel c is an input channel for a process then whenever the process is willing to accept an item of data on c it should be able to accept any of the possible data values. The choice as to which value is actually received will be made by the environment. However, the process should not be able to accept more than one data value from the channel at any one time as this cannot be achieved without interference on the channel.

Letting x be a free variable, we use the notation $c?x$ within the choice set of a set prefix construct to indicate the availability of all the events from $\{c.v \mid v \text{ an allowed data value on } c\}$ in the choice set. This should be seen within the context of the process being able to receive any data value along channel c. We shall assume that any attempt by the environment to perform more than one of the available data communications on a given channel will result in chaotic behaviour. We therefore find it unnecessary to state the behaviour of the process in such circumstances. For example,

$$[X \subseteq \{c?x\} \rightarrow (P \textbf{ if } X = \{\} \textbf{ else } Q(x))]$$

represents a process which may initially receive input on channel c; it then goes on to behave like P or $Q(x)$ depending on whether an input action occurred. In the case where a single data value is received, x takes this value, determining the behaviour of $Q(x)$.

Composition of processes with channels

We have already stipulated that only one process in a system will use a given channel for output. We put no restrictions on the number of processes in a system using a given channel for input. Thus we allow for simple specification of the concept of channels being forked supplying many processes with input. We shall now consider the effect of composing processes which have channels.

Suppose c is an output channel of P and an input channel of Q. Within the composed processes $P \parallel Q$ data is transmitted from P to Q via channel c. Other

processes within the system can read the data from P. To allow this, the channel c is visible as an output channel of the process $P \parallel Q$.

If c is an input channel of both P and Q, then the channel c is an input channel of the composed process $P \parallel Q$. Data transmitted to the composed process is forked internally to the component processes.

If c is a channel of P bnt not of Q, then this channel is visible as a channel of $P \parallel Q$ of the same type (input or output) as for process P.

4.1.1 Syntax for communication

Suppose, within a system, events which arise in the context of data communication are only referred to using the notation for input and ontput. Then we can provide a syntax within our algebra which allows us to abstract away from the individual events which make up the commnnication. We are able to view the problem as one of channels parameterised by the data they are carrying.

Alphabets

In SCSP every process has an alphabet. When considering communication it is convenient to distinguish the communication events from other events in the alphabet. We shall associate with each process, P, its input channels $in(P)$, its output channels $out(P)$ and non-communication events $ev(P)$. We define the set of channels of P to be $chan(P) = in(P) \cup out(P)$. Each channel, c, has an associated data set $\delta(c)$, a finite set giving permissible data values on channel c. The data set should be seen to be an attribute of a given channel in a system. Every process with a particular channel c will see the same associated data set for that channel. The alphabet of P is then given by

$$\alpha P = ev(P) \cup \{c.v \mid c \in chan(P) \wedge v \in \delta(c)\}$$

We also stipulate that the input and output channels of a process are disjoint,

$$in(P) \cap out(P) = \{\}$$

and that non-communication events do not coincide with communicatiou events

$$ev(P) \cap \{c.v \mid c \in chan(P) \wedge v \in \delta(c)\} = \{\}.$$

We strengthen our requirements on the alphabets of component processes in process expressions.

- Whereas previously component processes were reqnired to have the same alphabets, we now also demand that the input channels and output channels of the various components coincide.

- Parallel composition is restricted to the case where there are no output channels common to the component processes, and non-communication events of one process do not coincide with communication events of the other. The process $P \parallel Q$ has an alphabet composed of the following:

$$\begin{aligned} ev(P \parallel Q) &\triangleq ev(P) \cup ev(Q) \\ out(P \parallel Q) &\triangleq out(P) \cup out(Q) \\ in(P \parallel Q) &\triangleq (in(P) \cup in(Q)) - out(P \parallel Q) \end{aligned}$$

Processes

Communication events arise within processes expressions in the same situations as non-communication events. We provide special notation for referencing communication events to ensure they only appear in a meaningful context. If we restrict ourselves to the notation provided here for communication along channels, we can provide clarity in our modelling of such communication and ensure that the process constructs adhere to the requirements of such systems. We consider each of the constructs in which events appear explicitly.

Set prefix

Any reference to communication events in the choice set of the set prefix construct is only made via communication terms.

Definition 4.1 A *communication term* is either an *input term* or an *output term*. Output terms take the form $c!e$, where $c \in out(P)$ and expression e denotes a value in $\delta(c)$. Input terms take the form $c?x$, where $c \in in(P)$ and x is a free variable ◊

For every channel of the process there can be at most one communication term in the choice set. As before the behaviour of a process after a prefix choice will depend on the terms, X, selected from the choice set. The process is parameterised by $\dot{X}$ where any output terms, $c!v$, and input terms $d?x$, in X are replaced by $c.v$ and $d.x$ respectively in $\dot{X}$. Formally

$$\dot{X} = \{\dot{a} \mid a \in X\}, \qquad \text{where} \quad \dot{a} = \begin{cases} c.v & \text{if } a = c!v \\ c.x & \text{if } a = c?x \\ a & \text{otherwise} \end{cases}$$

So the resultant process is parameterised by the free variables corresponding to input data.

We give the set prefix construct containing communication terms the following meaning in basic SCSP.

$$[X \subseteq B \rightarrow P_X] \mathrel{\widehat{=}} [Y \subseteq B' \rightarrow Q_Y]$$

For every output term $c!v$ in B the event $c.v$ is made available in B'. For every input term $d?x$ in B all the events in the set $\{d.v \mid v \in \delta(d)\}$ are made available in B'. Formally,

$$B' = \bigcup_{a \in B} g(a), \quad \text{where } g(a) = \begin{cases} \{c.v\} & \text{if } a = c!v \\ \{c.v \mid v \in \delta(c)\} & \text{if } a = c?x \\ \{a\} & \text{otherwise.} \end{cases}$$

When an input term $c?x$ arises in the choice set B we need only provide processes corresponding to no more than one of the $c.v$ being chosen, the assnmption being that in all other cases the process behaves chaotically. Thus

$$Q_Y = \begin{cases} \bot & \text{if } \exists\, c?x \in B \bullet c.v, c.w \in Y \wedge v \neq w \\ P_Y & \text{otherwise} \end{cases}$$

Hiding

In addition to non-communication events we allow hiding of output channels. The implication is that all data communication events associated with a hidden channel are hidden. The set of hidden events may now include output channel names annotated with the ! symbol. So

$$P \setminus \{c!\}$$

behaves like process P with all the communication events on channel c hidden.

In general, the process $P \setminus B$ has output channels $out(P) - \{c \mid c! \in B\}$, the same input channels as P and non-communication events $ev(P) - B$. We give the hiding construct containing output channels the following meaning in basic SCSP.

$$P \setminus B \mathrel{\widehat{=}} P \setminus B'$$

For every output channel, $c!$, in B all the events in the set $\{c.v \mid v \in \delta(c)\}$ are present in B'. Formally:

$$B' = \bigcup_{a \in B} g(a), \quad \text{where } g(a) = \begin{cases} \{c.v \mid v \in \delta(c)\} & \text{if } a = c! \\ \{a\} & \text{otherwise.} \end{cases}$$

Renaming

Renaming must respect the type of an event and should preserve all the restrictions mentioned above. In order to ensure this, renaming of communication terms is divided into renaming of the channels and renaming of the data set $\delta(c)$ associated with the channel.

4.1.2 Laws for communication

The laws provided here can be established by expressing the processes in basic SCSP, applying laws of SCSP, then returning to the notation for communication. With these new laws available it is unnecessary to concern ourselves with the underlying SCSP treatment of communication. Instead we will be able to view data communication along channels at a level of abstraction more appropriate to the system being described.

Almost all the laws of SCSP can be applied directly to processes which use the communication notation. The laws which cannot be applied in their current form are those where the structure of either the choice set in a set prefix construct, or the set of terms hidden in communication abstraction is important. We shall consider generalised forms for the laws in question.

The axiom for parallel composition of set prefixed processes becomes:

A−10′: $[X \subseteq A' \to P_X] \parallel [Y \subseteq B' \to Q_Y] = [Z \subseteq C \to (P_{f(\dot{Z},P)} \parallel Q_{f(\dot{Z},Q)})]$

Terms in C are either communication terms or non-communication events. Communication terms arise in C if one component process is prepared to communicate with the environment on a channel which is not shared by the components, or if both components are willing to communicate along a common channel. If the common channel is an input channel for both components then cooperation will result in an input to the composite process. If the common channel is an input channel for one component and an output channel for the other then data flow can occur between the components. The data can be seen to be output by the composed process. The occurrence of non-communication events in C is governed in the same way as before. Formally:

$$\begin{aligned} C = \; & \{c!v \mid c \in (out(P) \cup out(Q)) - (chan(P) \cap chan(Q)) \wedge c!v \in A' \cup B'\} \\ & \cup \{c?x_c \mid c \in (in(P) \cup in(Q)) - (chan(P) \cap chan(Q)) \wedge \exists\, x \bullet c?x \in A' \cup B'\} \\ & \cup \{c?x_c \mid c \in in(P) \cap in(Q) \wedge \exists\, x, y \bullet c?x \in A' \wedge c?y \in B'\} \\ & \cup \{c!v \mid c \in in(P) \cap out(Q) \wedge \exists\, x \bullet c?x \in A' \wedge c!v \in B'\} \\ & \cup \{c!v \mid c \in in(Q) \cap out(P) \wedge \exists\, x \bullet c?x \in B' \wedge c!v \in A'\} \\ & \cup \{a \mid a \in ev(P) \cup ev(Q) \wedge a \in ((A' \cap B') \cup (A' - ev(Q)) \cup (B' - ev(P)))\} \end{aligned}$$

$f(\dot{Z}.P)$ gives the terms from $\dot{Z}$ seen in the context of process P. We can only

see events in the alphabet of P and communication on the channels of P.

$$f(\dot{Z}, P) = \{c.v \in \dot{Z} \mid c \in chan(P)\} \cup (\dot{Z} \cap ev(P))$$

The axiom for distributing hiding through set prefix becomes:

A–13′: $[X \subseteq B \rightarrow P_{\dot{X}}] \setminus A = [Y \subseteq C \rightarrow (P_{\dot{Y} \cup (\dot{B} - \dot{C})} \setminus A)]$

where $C = B - (A \cup \{c!v \in B \mid c! \in A\})$. This ensures that all communication terms on hidden channels are hidden.

Finally the law for distributing hiding through parallel composition is further restricted:

L–5′: $(P \parallel Q) \setminus A = (P \setminus A) \parallel (Q \setminus A)$

where $A \cap ev(P) \cap ev(Q) = \{\}$ and $\{c \mid c! \in A\} \cap chan(P) \cap chan(Q) = \{\}$. So hiding only distributes through parallel composition when neither common events nor common channels are hidden.

4.2 Token Ring

Using the communication notation developed in this chapter, we demonstrate the use of SCSP in specifying a protocol for local area networks. The protocol chosen is a token ring protocol based on the IEEE 802.5 Standard [IEE85] as described in [Tan89].

In a network using a token ring protocol every station in the network is connected to a *ring interface*, Figure 4.1. The ring itself is constructed by connecting the ring interfaces by point-to-point links to form a complete circle, allowing unidirectional data flow around the ring. When a bit arrives at a ring interface it is copied into a 1-bit buffer, inspected and possibly modified, then written back out to the ring.

Whenever all the stations are idle, having nothing to transmit, a special bit pattern, called the *token*, circulates around the ring. There is only one such token on the ring. When a station wants to transmit data it must capture the token. It may then enter its data on to the ring and, once transmission is complete, replace the token. As there is only one token, this procedure eliminates any possibility of collisions.

The token ring has a minimum capacity implicit in its design — when all stations are idle the complete token must reside on the ring. The capacity of the ring is dependent on the number of stations in the network, each providing a 1-bit delay, and signal propagation delay along the wires.

Each ring interface has two modes of operation, *listen* or *transmit*. In listen mode data is simply copied from the input to the output. If the data is addressed

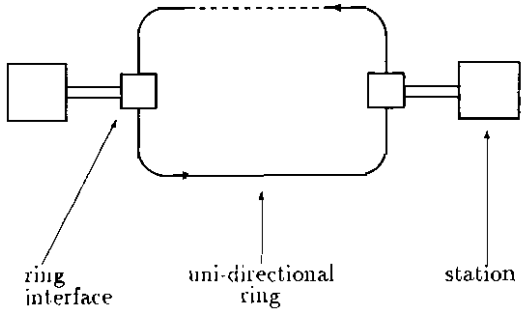


Figure 4.1: A Token Ring

to the listening station, then it is also copied to the station as it passes through the interface. A ring interface can only enter transmit mode if it is in possession of the token. In this mode the interface breaks the connection between input and output and enters its own data onto the ring. Once the data is returned to the sending interface it is drained from the ring. The sending station may then discard the returned data or check it against the original. Transmission completed, the token is regenerated. Finally, once all the data has returned to the sender the ring interface returns to listen mode, completing the ring.

As it is not necessary to hold the complete frame of data on the ring at any one instant, there is no physical limit on the size of data packets which may be transmitted. Acknowledgement of receipt of data can be achieved simply by including a bit in the frame format which is inverted by the receiving station and checked on return to the sending station.

When traffic is light the token spends most of its time going around the ring. When traffic is heavy the stations are given the opportunity to transmit in turn around the ring; as the token is relinquished by one station it is captured by the next station round the ring which wishes to transmit. Setting an upper bound on token holding time we can ensure that the protocol is fair. Also network efficiency, with respect to utilising the ring, can approach 100 percent under heavy loads.

Each ring has a *monitor* station which is responsible for ring maintenance. Its responsibilities include ensuring the token is not lost, keeping the ring free of garbage and dealing with breaks in the ring. Other features of the 802.5 token ring include a priority system for ring access and the ability to broadcast data to several stations.

4.2.1 Specification in SCSP

The specification developed here only covers the basic features of the token ring. By making the following assumptions we restrict ourselves to specifying a simple ring interface in this study.

- We shall ignore signal propagation delay along links between interfaces. The assumption being that the ring is started with sufficient stations to provide the ring capacity required to hold the token.
- We assume that the sending station discards the returned data without checking it.
- Acknowledgement of data is not considered here. We are only concerned with the flow of data rather than its exact value, we shall assume that data reached its destination if it returned to the sender.
- We shall not specify a monitor station: we shall only concern ourselves with the behaviour of the ring when all the stations are operating correctly. Ring maintenance only becomes necessary when the stations go down, loosing the token or breaking the ring as they do so.
- Special features of the 802.5 token ring such as a priority system for ring access are not considered in this study.
- The sending station is restricted to sending just one data frame on each occasion it captures the token. In the 802.5 token ring several data frames may be sent but, to ensure successful draining of the ring, no more than one header is allowed to reside on the ring at any one instant and the token is not replaced until the last header has been drained from the ring. This way once transmission is complete it is sufficient to drain all data up to and including the first 'end-of-frame' field before resuming listen mode.

Finally we shall use a very simplified format for both the token and the data frame. The token in the 802.5 token ring consists of 3 octets and can be distinguished from other data on the ring by its first two octets. This means that a ring interface must keep recorded the last few values it has seen on the ring so that it can recognise the token when necessary. Similarly the data frame contains several fixed length fields in its head and tail, the details of which can be found in [Tan89].

We shall assume that the following distinguished bits are available.

TK -- token
SF — start-of-frame
EF — end-of-frame

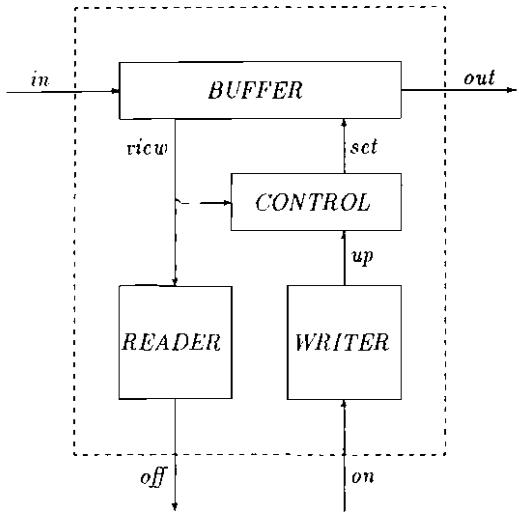


Figure 4.2: Structure of a ring interface

For our purposes the token is a single distinguished bit while the data frame has format:

$$SF \mid DA \mid \text{Data} \mid EF$$

SF and *EF* are 1-bit markers while *DA* is a fixed length field containing the destination address. By assuming the existence of these distinguished bits we can ignore the problem of internal buffering and concentrate on the mechanism of the protocol.

4.2.2 Ring interface

We divide the ring interface into four components as shown in Figure 4.2. These components are:

BUFFER This is a 1-place buffer the contents of which can be viewed and altered by the *CONTROL* process.

CONTROL This process switches between letting data through the buffer unchanged and setting the value in the buffer, depending on whether the interface is in listen or transmit mode.

WRITER This process provides a buffer of data waiting to be transmitted.

READER This process observes data entering the buffer and forwards data with matching address to its station.

The *CONTROL* and *WRITER* processes are responsible for transmission of data while the *READER* acts as a receiver for the station.

BUFFER

BUFFER acts as a 1-place buffer on the ring. Between the data entering the buffer from the ring and exiting the buffer to the ring there is an opportunity for the data to be viewed and altered. In order to allow the controller to modify the buffer, the buffer cycles its behaviour over a 3 phase clock cycle.

In the first phase the data currently held by the buffer is written to the ring and a new data value is read from the ring. In the second phase the controller and the reader can if necessary view the data which has been read into the buffer. In the final phase the controller may overwrite the contents of the buffer.

$$BUFF_y \;\widehat{=}\; [\{in?x, out!y\} \rightarrow BUFF'_x \rhd \bot]$$

$$BUFF'_y \;\widehat{=}\; [X \subseteq \{view!y\} \rightarrow BUFF''_y]$$

$$BUFF''_y \;\widehat{=}\; [\{set?x\} \rightarrow BUFF_x \rhd BUFF_y]$$

The design requires all the interfaces to be synchronised to allow the data transfer around the ring to occur without corruption. The ability to specify synchronised data transfer around the ring means that the ring capacity in our model is equal to the number of buffers on the ring.

CONTROL

When the ring interface is in listen mode the controller allows the hits to pass through the buffer unchanged. The controller also registers whether its station has outstanding data to send. If the station sends data to the interface for transmission then it is the task of the controller to capture the token so the interface can enter transmission mode.

$$\begin{aligned} LISTEN \;\widehat{=}\; & [\{up?z, view?x\} \rightarrow (TRANS_{SF} \textbf{ if } x = TK \textbf{ else } REQ) \\ & \Box \{up?z\} \rightarrow REQ \rhd LISTEN] \end{aligned}$$

$$REQ \;\widehat{=}\; view?x \rightsquigarrow (TRANS_{SF} \textbf{ if } x = TK \textbf{ else } REQ)$$

Once the ring interface has captured the token it enters transmission mode and the *CONTROL* process transmits all the data provided by the *WRITER* process.

$$
\begin{aligned}
TRANS_y &\triangleq [\{set!y\} \rightarrow TRANS' \rhd \bot] \\
TRANS' &\triangleq [\{up?x\} \rightarrow TRANS''_x \rhd DRAIN''_{TK}] \\
TRANS''_y &\triangleq wait(1) \rightarrow TRANS_y
\end{aligned}
$$

Transmission works on a 3 phase cycle in order to keep synchronised with the buffer. The first phase of the transmitter must coincide with the final phase of the buffer; in this step the data in the buffer is reset. Failure to reset data in the buffer before it is transferred would result in data corruption. In the second phase, the next value to be transmitted is collected; if no such value is available transmission is assumed to be complete. The token must be replaced and the data frame drained from the ring before the controller returns to listen mode. It is not necessary to view incoming data from the ring during transmission as we have chosen to discard the returned data without checking it. Consequently the third phase, which could have involved inspection of the buffer, involves a wait to maintain synchronisation.

Once transmission is complete the data frame must be drained from the ring. This is achieved by transmitting null characters (denoted $*$) until the end of frame marker has returned to the sender.

$$
\begin{aligned}
DRAIN_y &\triangleq [\{set!y\} \rightarrow DRAIN' \rhd \bot] \\
DRAIN' &\triangleq wait(1) \rightarrow DRAIN''_* \\
DRAIN''_y &\triangleq [\{view?z\} \rightarrow (END \textbf{ if } z = EF \textbf{ else } DRAIN_y) \rhd \bot]
\end{aligned}
$$

While the ring interface is draining the data from the ring, it is imperative that each bit of incoming data is viewed so as not to miss the end of frame delimiter. Failure to see this delimiter would result in either the token or messages belonging to other stations erroneously being drained from the ring.

Once the end of frame marker has returned we must delete this from the buffer and immediately return to listen mode.

$$END \triangleq [\{set!*\} \rightarrow LISTEN \rhd \bot]$$

WRITER

Data arrives at the ring interface from the station on channel *on*. The *WRITER* is responsible for formatting the data into a data frame. As the token holding time is directly proportional to the length of the data frame the writer can be responsible for ensuring that the token is returned within the token holding time. This is

achieved by splitting data across several frames if it is too long to be transmitted within the time limit.

$$
\begin{aligned}
WRITE &\triangleq WRITE_{\langle\rangle} \\
WRITE_{\langle\rangle} &\triangleq on?d \rightsquigarrow WRITE_{fr(d)} \\
WRITE_{\langle y:s\rangle:frs} &\triangleq up!y \rightsquigarrow WRITE_{s:frs} \\
WRITE_{\langle\rangle:frs} &\triangleq wait(3) \rightarrow WRITE_{frs}
\end{aligned}
$$

where $fr(d)$ is a list of frames, $(s : frs)$ denotes frame s followed by frames frs, and $(y : s)$ denotes bit y followed by bits s.

Once a complete frame has been sent, the *WRITER* blocks sending for sufficient time to guarantee the controller will stop transmitting and return the token to the ring.

READER

This process acts as a receiver, viewing data as it passes the buffer. If it sees a start frame delimiter then it checks the destination address of the frame. If the address matches the address of the station then the reader stores all subsequent data until the end of frame delimiter and passes the data to the station. If the data does not carry the correct address then it is ignored by the reader.

$$
\begin{aligned}
READ &\triangleq view?y \rightsquigarrow (CHECK_{\langle\rangle} \textbf{ if } y = SF \textbf{ else } READ) \\
CHECK_s &\triangleq view?y \rightsquigarrow (\ KEEP_{\langle\rangle} \textbf{ if } s^\frown\langle y\rangle = address \textbf{ else} \\
&\qquad\qquad (CHECK_{s^\frown\langle y\rangle} \textbf{ if } s^\frown\langle y\rangle < address \textbf{ else } READ)) \\
KEEP_s &\triangleq view?y \rightsquigarrow (SEND_s \textbf{ if } y = EF \textbf{ else } KEEP_{s^\frown\langle y\rangle}) \\
SEND_s &\triangleq [\{view?y, off!s\} \rightarrow (CHECK_{\langle\rangle} \textbf{ if } y = SF \textbf{ else } READ) \\
&\qquad \Box\ \{off!s\} \rightarrow READ \rhd \bot]
\end{aligned}
$$

We assume that the station is always able to accept data. This may require some intermediate buffering.

Notice also, this process is always willing to view data. Thus it cannot block the buffer process.

4.2.3 A complete ring

When the system is started we assume that one of the buffers carries the token and the remaining buffers on the ring contain null bits. We assume that every interface is in listen mode with no data pending transmission. So every interface is given by:

$$INTER_x \triangleq (BUFF_x \parallel LISTEN \parallel WRITE \parallel READ) \setminus \{view!, up!, set!\}$$

where $x \in \{TK, *\}$

Assuming there are $n+1$ stations and the interface labeled 0 contains the token then the system is initially described by the process:

$$SYSTEM \triangleq (\ \|_{i \in [1..n]}\ i.INTER_*) \parallel 0.INTER_{TK}$$

where

$$i.INTER_x \triangleq INTER_x[i.ring/in, (i \oplus 1).ring/out, i.off/off, i.on/on]$$

$\oplus$ being addition modulo $n+1$.

We could hide the ring mechanism by hiding the channels $\{i.ring \mid 0 \leq i \leq n\}$. Although this hides the token, enabling it to travel around the ring internally, such abstraction does not resnlt in infinite chatter and chaos. Even when an event is hidden we know it can occur at most once in every time unit. This contrasts to CSP, where much care must be taken to avoid infinite chatter when hiding events which could occur arbitrarily often, such as the token passing round a ring when all interfaces are idle.

4.2.4 Investigating the interface

In order to investigate the behaviour of a ring interface we shall consider the composition of the components *BUFFER* , *CONTROL* , and *WRITER* . We shall restrict our interest to the transmission properties of the interface, so we shall exclude the *READER* component from our investigations. We simply note that the *READER* cannot affect the behaviour of the channel *view* and this is the only opportunity for communication between the *READER* process and the remaining processes within the interface.

We wish to establish the bebavionrs of the process

$$INT_y \triangleq (BUFF_y \parallel LISTEN \parallel WRITE) \setminus \{view!, set!, up!\}$$

To achieve this we use the algebraic laws to reduce the above expression to a form which does not involve the parallel composition or hiding operators.

$$\begin{array}{ll} & INT_y \\ \equiv & \{ \text{ by definition of } INT_y \ \} \\ & (BUFF_y \parallel LISTEN \parallel WRITE) \setminus \{view!, set!, up!\} \\ \equiv & \{ \text{ by L–3 } \} \\ & ((BUFF_y \parallel LISTEN) \parallel WRITE) \setminus \{view!, set!, up!\} \end{array}$$

Now

$$
\begin{array}{ll}
 & (BUFF_y \parallel LISTEN) \\
\equiv & \{ \text{ expanding definition of } BUFF_y \text{ and } LISTEN \ \} \\
 & [\{in?x, out!y\} \rightarrow BUFF'_x \rhd \bot] \\
 & \parallel [\{up?z, view?x\} \rightarrow (TRANS_{SF} \textbf{ if } x = TK \textbf{ else } REQ) \\
 & \quad \Box \{up?z\} \rightarrow REQ \rhd LISTEN] \\
\equiv & \{ \text{ by A-10}' \ \} \\
 & [\{in?x, out!y, up?z\} \rightarrow BUFF'_x \parallel REQ \\
 & \Box \{in?x, out!y\} \rightarrow BUFF'_x \parallel LISTEN \rhd \bot]
\end{array}
$$

So

$$
\begin{array}{ll}
 & ((BUFF_y \parallel LISTEN) \parallel WRITE) \setminus \{view!, set!, up!\} \\
\equiv & \{ \text{ expanding } WRITE \text{ and from above } \} \\
 & ([\{in?x, out!y, up?z\} \rightarrow BUFF'_x \parallel REQ \\
 & \quad \Box \{in?x, out!y\} \rightarrow BUFF'_x \parallel LISTEN \rhd \bot] \\
 & \parallel [\{on?d\} \rightarrow WRITE_{fr(d)} \rhd WRITE]) \setminus \{view!, set!, up!\} \\
\equiv & \{ \text{ by A-10}' \text{ and A-13}' \ \} \\
 & [\{in?x, out!y, on?d\} \rightarrow \\
 & \qquad (BUFF'_x \parallel LISTEN \parallel WRITE_{fr(d)}) \setminus \{view!, set!, up!\} \\
 & \Box \{in?x, out!y\} \rightarrow (BUFF'_x \parallel LISTEN \parallel WRITE) \setminus \{view!, set!, up!\} \\
 & \rhd \bot]
\end{array}
$$

By continuing to eliminate parallel composition and hiding in the above manner we can demonstrate that

$$INT_y \equiv I(y, L, \langle\rangle)$$

where $I(y, L, \langle\rangle)$ is given by the mutual recursion in Figure 4.3. The parameters of I can be given the following interpretation in the system:

- The first parameter is the current value stored in the buffer.
- The second parameter is a value taken from the set $\{L, T, D\}$ and indicates the mode of the interface,

  L — Listen mode

  T — Transmit mode, data still being sent

  D — Transmit mode, returned data being drained.

- The final parameter is a list of lists of bits, representing data pending transmission, stored in frames.

$$
\begin{aligned}
I(y, L, \langle\rangle) &\triangleq [\{in?x, out!y.\ on?d\} \rightarrow I'(x, L, fr(d)) \\
&\quad \Box \{in?x, out!y\} \rightarrow I'(x, L, \langle\rangle) \\
&\quad \rhd \bot] \\
\\
I'(y, L, \langle\rangle) &\triangleq [\{on?d\} \rightarrow (wait(1) \rightarrow I(y, L, fr(d))) \\
&\quad \rhd [\{on?d\} \rightarrow I(y, L, fr(d)) \\
&\quad\quad \rhd I(y, L, \langle\rangle)]\ ] \\
\\
I(y, L, s{:}frs) &\triangleq [\{in?x, out!y\} \rightarrow I'(x, L, s{:}frs) \\
&\quad \rhd \bot] \\
\\
I'(y, L, (z{:}s){:}frs) &\triangleq wait(2) \rightarrow (I(SF, T, s{:}frs) \textbf{ if } y = TK \textbf{ else} \\
&\quad\quad\quad\quad I(y, L, (z{:}s){:}frs)) \\
\\
I(y, T, (z{:}s){:}frs) &\triangleq [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow I(z, T, s{:}frs)) \\
&\quad \rhd \bot] \\
\\
I(y, T, \langle\rangle{:}(s{:}frs)) &\triangleq [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow I(TK, D, s{:}frs)) \\
&\quad \rhd \bot] \\
\\
I(y, T, \langle\langle\rangle\rangle) &\triangleq [\{in?x, out!y\} \rightarrow I'(TK, T, \langle\rangle) \\
&\quad \rhd \bot] \\
\\
I'(y, T, \langle\rangle) &\triangleq [\{on?d\} \rightarrow (wait(1) \rightarrow I(y, D, fr(d))) \\
&\quad \rhd [\{on?d\} \rightarrow I(y, D, fr(d)) \\
&\quad\quad \rhd I(y, D, \langle\rangle)]\ ] \\
\\
I(y, D, s{:}frs) &\triangleq [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow I''(x, s{:}frs)) \\
&\quad \rhd \bot] \\
\\
I(y, D, \langle\rangle) &\triangleq [\{in?x, out!y, on?d\} \rightarrow (wait(2) \rightarrow I''(x, fr(d))) \\
&\quad \Box \{in?x, out!y\} \rightarrow I'(x, D, \langle\rangle) \\
&\quad \rhd \bot] \\
\\
I'(y, D, \langle\rangle) &\triangleq [\{on?d\} \rightarrow (wait(1) \rightarrow I''(y, fr(d))) \\
&\quad \rhd [\{on?d\} \rightarrow I''(y, fr(d)) \\
&\quad\quad \rhd I''(y, \langle\rangle)]\ ] \\
\\
I''(x, s) &\triangleq I(*, L, s) \textbf{ if } x = EF \textbf{ else } I(*, D, s)
\end{aligned}
$$

Figure 4.3: Specification of the ring interface, $INT_y \equiv I(y, L, \langle\rangle)$

Interface with arbitrary data supply

When we consider properties of the ring interface we want to be sure that these hold regardless of the data transmitted. We shall thus place the interface in an environment in which the data supply is specified as weakly as possible. We model an arbitrary data supply by the process:

$$DATA \triangleq (\prod_{d \in \delta(on)} (on!d \rightsquigarrow DATA)) \sqcap (wait(1) \rightarrow DATA)$$

The process waits a random (possibly infinite) length of time before offering the first data item to the interface. Once this is accepted further data items may be offered after random delays. Assuming that once data is made available for transmission it will remain available until accepted by the interface, then any actual data supply D will be more deterministic than the process $DATA$, $DATA \sqsubseteq D$.

We shall now consider the behaviour of the interface when supplied with data in the above arbitrary manner, by considering the process

$$(INT_y \parallel DATA) \setminus \{on!\}$$

As before we use the algebraic laws to eliminate parallel composition and hiding from this expression. We define

$$ID(y, X, s) \triangleq (I(y, X, s) \parallel DATA) \setminus \{on!\}$$

Clearly by the definition of INT_y

$$(INT_y \parallel DATA) \setminus \{on!\} \equiv ID(y, L, \langle\rangle)$$

We can derive the mutual recursion shown in Figure 4.4. The first steps of this derivation are presented in Appendix C.1.

Timing properties of the interface

We shall demonstrate that, regardless of the data sent to the interface for transmission, the time lapse between an interface receiving the token and outputting the token to the next ring interface does not exceed $3(m+1)$ units. Here m is the maximum allowed length of a data frame.

We recall there is only one token on the ring. While data is being transmitted by an interface the token must be held by that interface, so no token can arrive at an interface while it is transmitting data. The last bit of the data frame is an end-of-frame marker. once this has been transmitted the token is returned to the ring. Once the end-of-frame marker has returned to the sending interface it immediately returns to listen mode. The token cannot arrive at the draining interface before

$$
\begin{aligned}
ID(y, L, \langle\rangle) &\equiv [\{in?x, out!y\} \rightarrow ((\textstyle\bigsqcap_{d \in \delta(on)} ID'(x, L, fr(d))) \\
&\qquad\qquad \sqcap ID'(x, L, \langle\rangle)) \\
&\quad \rhd \bot] \\
ID'(y, L, \langle\rangle) &\equiv wait(2) \rightarrow ((\textstyle\bigsqcap_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle)) \\
ID(y, L, s{:}frs) &\equiv [\{in?x, out!y\} \rightarrow ID'(x, L, s{:}frs) \\
&\quad \rhd \bot] \\
ID'(y, L, (z{:}s){:}frs) &\equiv wait(2) \rightarrow (ID(SF, T, s{:}frs) \textbf{ if } y = TK \textbf{ else} \\
&\qquad\qquad ID(y, L, (z{:}s){:}frs)) \\
ID(y, T, (z{:}s){:}frs) &\equiv [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow ID(z, T, s{:}frs)) \\
&\quad \rhd \bot] \\
ID(y, T, \langle\rangle{:}(s{:}frs)) &\equiv [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow ID(TK, D, s{:}frs)) \\
&\quad \rhd \bot] \\
ID(y, T, \langle\langle\rangle\rangle) &\equiv [\{in?x, out!y\} \rightarrow ID'(TK, T, \langle\rangle) \\
&\quad \rhd \bot] \\
ID'(y, T, \langle\rangle) &\equiv wait(2) \rightarrow ((\textstyle\bigsqcap_{d \in \delta(on)} ID(y, D, fr(d))) \sqcap ID(y, D, \langle\rangle)) \\
ID(y, D, s{:}frs) &\equiv [\{in?x, out!y\} \rightarrow (wait(2) \rightarrow ID''(x, s{:}frs)) \\
&\quad \rhd \bot] \\
ID(y, D, \langle\rangle) &\equiv [\{in?x, out!y\} \rightarrow ID'(x, D, \langle\rangle) \\
&\quad \rhd \bot] \\
ID'(y, D, \langle\rangle) &\equiv wait(2) \rightarrow ((\textstyle\bigsqcap_{d \in \delta(on)} ID''(y, fr(d))) \sqcap ID''(y, \langle\rangle)) \\
ID''(x, s) &\equiv ID(*, L, s) \textbf{ if } x = EF \textbf{ else } ID(*, D, s)
\end{aligned}
$$

Figure 4.4: Specification of the ring interface supplied with random data

the end-of-frame marker. So the token can only arrive at an interface if it is in listen mode.

We consider the behaviour of an interface in listen mode, with a random supply of data, when a token is input to the interface on channel *in*. We are interested in the time it takes for the token to be output to the next interface on channel *out*.

We shall describe the provision of a token by

$$\begin{aligned} TOK &\triangleq in!TK \rightsquigarrow NUL \\ NUL &\triangleq in!* \rightsquigarrow NUL \end{aligned}$$

This process provides the interface with a token followed by arbitrary data, represented by the $*$ symbol. We shall assume that data flow around the ring is not blocked, so all output on channel *out* is allowed by the environment.

We are interested in the time elapsed before the event $out!TK$ can be performed by the process

$$(ID(y, L, s) \parallel TOK) \setminus \{in!\}$$

where $s \in \{\langle\rangle\} \cup \{fr(d) \mid d \in \delta(on)\}$

By application of the laws of SCSP we can show:

$$\begin{aligned} &(ID(y, L, \langle\rangle) \parallel TOK) \setminus \{in!\} \\ &\quad \equiv [\{out!y\} \rightarrow (wait(2) \rightarrow ((\textstyle\prod_{d \in \delta(on)} (ID(SF, T, fr'(d)) \parallel NUL) \setminus \{in!\}) \\ &\qquad\qquad\qquad\qquad \sqcap [\{out!TK\} \rightarrow P \rhd \bot]) \\ &\qquad \rhd \bot] \end{aligned}$$

and

$$\begin{aligned} &(ID(y, L, fr(d)) \parallel TOK) \setminus \{in!\} \\ &\quad \equiv [\{out!y\} \rightarrow (wait(2) \rightarrow (ID(SF, T, fr'(d)) \parallel NUL) \setminus \{in!\}) \\ &\qquad \rhd \bot] \end{aligned}$$

where $fr'(d) = s{:}frs$, given $fr(d) = (SF{:}s){:}frs$.

We see that after 3 units either the token is returned or the interface enters transmit mode. If the interface does not enter transmit mode then the token is passed through the buffer unchanged.

It remains to consider the time taken for an interface in transmit mode to return a token. $fr'(d)$ is a list of frames with the start-of-frame marker removed from the first frame. So the first frame has length less than m. We shall establish, by induction on $|s|$, that the process

$$(ID(y, T, s{:}frs) \parallel NUL) \setminus \{on!\}$$

is willing to perform the event $out!TK$ after $3(|s|+1)$ units. This result is sufficient to guarantee our requirement regarding the token holding time.

Base case $|s| = 0$. Using the laws of SCSP we can demonstrate that:

$$\begin{aligned}&(ID(y, T, \langle\rangle : frs) \parallel NUL) \setminus \{on!\}\\ &\quad \equiv [\{out!y\} \rightarrow (wait(2) \rightarrow [\{out!TK\} \rightarrow P \rhd \bot])\\ &\quad\quad \rhd \bot]\end{aligned}$$

So $(ID(y, T, \langle\rangle : frs) \parallel NUL) \setminus \{on!\}$ is willing to perform the event $out!TK$ after 3 units.

Inductive step $|z : s| = n + 1$. Again using the laws of SCSP we can show that:

$$\begin{aligned}&(ID(y, T, (z : s) : frs) \parallel NUL) \setminus \{on!\}\\ &\quad \equiv [\{out!y\} \rightarrow (wait(2) \rightarrow (ID(y, T, s : frs) \parallel NUL) \setminus \{on!\})\\ &\quad\quad \rhd \bot]\end{aligned}$$

So, assuming the environment is always willing to receive data on channel out, $(ID(y, T, (z : s) : frs) \parallel NUL) \setminus \{on!\}$ behaves like process

$$(ID(y, T, s : frs) \parallel NUL) \setminus \{on!\}$$

after 3 units and by induction this process is willing to perform the event $out!TK$ after 3(n+1) units. So

$$(ID(y, T, (z : s) : frs) \parallel NUL) \setminus \{on!\}$$

is willing to perform the event $out!TK$ after $3(|z : s| + 1)$ units.

Hence we have established that the time lapse between the token arriving at the interface and its being relinquished by the interface does not exceed $3(m + 1)$ units, where m is the maximum allowed frame length.

4.3 Conclusion

By providing a special notation, communication of data via channels between components of a system can be captured succinctly by SCSP processes in a manner familiar to CSP [Hoa85]. Although the underlying treatment of communication is fairly complex in SCSP, modification of some of the algebraic laws of SCSP has made it possible to manipulate algebraically expressions, which use the communication notation, without referring to the underlying SCSP representation of communication.

Using the communication notation, we have been able to specify a simple token ring in SCSP in terms of several simple components. We have demonstrated that the algebra is sufficiently powerful for us to establish behavioural properties of the ring interface by simple algebraic manipulation. Moreover, as SCSP incorporates

an element of timing information, we have been able to establish the token holding time of the ring interface. The timed framework of SCSP makes it possible to hide the mechanism of the protocol by hiding the channels which constitute the ring. If the ring is idle, the (then hidden) token could be passed around the ring indefinitely; however, this abstraction does not result in infinite chatter, as it would in an untimed model such as CSP, since the token takes time (which is not hidden) to pass around the ring.

Chapter 5

Synchronous Receptive Process Theory

A discrete time algebra is particularly appropriate for modelling clocked circuits; the time component exactly captures the clock's behaviour. However, components within a circuit are always willing to receive input, while output is never blocked. In SCSP it is possible to model such systems by making sure every event corresponding to an input is available for all time and assuming that the system becomes chaotic if an output is blocked by the environment.

Example We consider the SCSP specification of a NAND gate with unit response time. The output of a NAND gate is only low if both inputs were high at the previous time step. We assume the NAND gate has input wires 'a' and 'b' and output wire 'c' as shown in Figure 5.1. We model the gate by recording voltage-levels on wires. Event a occurring corresponds to the voltage-level on the wire labelled 'a' being high, otherwise the voltage-level is assumed to be low. We assume that both inputs are initially high. This gives us the following specification:

$$NAND = [X \subseteq \{a, b\} \rightarrow (NAND \textbf{ if } X = \{a, b\} \textbf{ else } NAND')]$$
$$NAND' = [X \subseteq \{a, b, c\} \rightarrow (NAND \textbf{ if } X = \{a, b, c\} \textbf{ else } (NAND' \textbf{ if } c \in X \textbf{ else } \bot))]$$

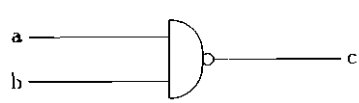


Figure 5.1: A NAND Gate

The gate is always willing to accept a high voltage-level on the input wires, while the voltage-level on the output wire can only vary in accordance with the inputs at the previous time step.

This small example has resulted in a process expression which is encumbered by the assumptions concerning the nature of communication. By encoding such assumptions into our model we are able to develop a synchronous version of Receptive Process Theory [Jos92]. This language can be viewed as a sublanguage of SCSP and its links with SCSP will be presented in a later chapter.

In this chapter we present the language of the synchronous receptive process theory, SRPT, and its associated denotational model. As with SCSP, a given event can occur at most once at each time step. In contrast to SCSP, the language of SRPT distinguishes between input events and output events.

Later in this chapter we shall see that, by making a semantic distinction between input and output events, we do not need to record refusal information in the model for SRPT. Behaviours are recorded as traces of sets, the sets consisting simply of events. We also obtain a straightforward encoding of divergence, however, by doing this we find ourselves considering a partial order on processes different to the usual non-determinism ordering.

5.1 The Language

SRPT is intended to model the interaction of an input-output system with its environment. A system is always able to accept any input from the environment and the environment may not block any output from a system. The term 'receptive', previously used by Josephs [Jos92] and Dill [Dil89], is used to capture these conditions on the input and output of a system. As in SCSP all communication is instantaneous.

As in SCSP, we presuppose a universal alphabet of events Σ. In SRPT we associate two sets of events $I, O \subseteq \Sigma$ with each process. These are referred to as the input and output alphabets of a process. We require both I and O to be finite, at least one of I and O to be non-empty, $I \cup O \neq \{\}$, and the sets to be disjoint, $I \cap O = \{\}$. We also presuppose a set of process variables Var. As before, these variables facilitate the definition of recursion.

The abstract syntax of the receptive language is similar to that of SCSP, the only noticeable variation being the slightly different form of the prefix construct. Take P to range over process terms. $I, O \in \mathbb{F}\,\Sigma$, $x \in Var$ and S to range over bijective renaming functions $S : \Sigma \to \Sigma$. Then, with certain restrictions on the alphabets of

the processes. the following grammar defines the syntax of the language SRPT:

$P ::=$	$\bot_{I,O}$	Chaos
	$\mid x$	process variable
	$\mid P \sqcap P$	non-deterministic choice
	$\mid [!O?X \rightarrow P_X]$	output prefix
	$\mid P \parallel P$	parallel composition
	$\mid P \setminus O$	hiding
	$\mid P[S]$	renaming
	$\mid \mu x : I, O \bullet P$	recursion

We now present the informal interpretation of each of these terms, highlighting the differences between this language and SCSP. We also consider the restrictions imposed upon the alphabets of the process terms.

5.1.1 Primitive processes and operators

Throughout this section we shall use ιP and oP to denote the input and ontpnt alphabets of process P, while αP will denote the combined alphabets of P, $\alpha P = \iota P \cup oP$.

Chaos

The process $\bot_{I,O}$ is the most undesirable process with input alphabet I and output alphabet O; it can give no information about its behavionr. This process is nsed to model behaviour when things go wrong, no useful information is available about the system, the process is divergent. No recovery is available from a process in this erroneous state and in this respect chaos is identifiable witb the process $\bot$ in SCSP.

Where the alphahets can be deduced from the context we will simply write $\bot$.

Process variable

$x \in Var$ represents the process bound to variable x in the context of given variable bindings. As in SCSP, we cannot make any deductions about the process to which x is bound until the choice of variable bindings is made explicit.

Non-deterministic choice

If two processes P and Q have common input and output alphabets, I and O respectively, then the non-deterministic choice of these two processes $P \sqcap Q$ is

defined to be the process witb iuput alphabet I and output alphabet O which non-deterministically behaves like P or Q. The choice occurs internally within the system; the environment has no control over the outcome of the choice. Non-deterministic choice is a demonic choice; a process which may internally choose to behave erroneously, is itself erroneous. This is reflected in a-4 below:

a–1 : $P \sqcap Q \equiv Q \sqcap P$

a–2 : $(P \sqcap Q) \sqcap R \equiv P \sqcap (Q \sqcap R)$

a–3 : $P \sqcap P \equiv P$

a–4 : $P \sqcap \bot \equiv \bot$

Output prefix

Let P be a $\mathbb{P}(I)$-indexed family of processes, each with input alphabet I and output alphabet O. A prefix set B is a subset of the output alphabet O. The process $[!B?X \rightarrow P_X]$ performs the events in B and any subset C of events from I at the first time step. The process then goes on to behave like P_C. If P is a process with empty input alphabet we shall simply write $[!B \rightarrow P]$.

This differs from the set prefix construct of *SCSP* in that it reflects the receptive behaviour of processes. The environment must allow the process to perform all the output events in B. This prefix construct does not provide a choice as to the output performed. The environment only has a choice as to how much input it provides the process. The process must allow any possible combination of input and its subsequent behaviour is influenced by the input provided by the environment.

Consider, for example, the process

$$[!\{a\}?X \rightarrow (\bot \textbf{ if } X = \{\} \textbf{ else } P)]$$

Initially this process will perform the output event a and a set of events in its input alphabet. No other output can occur. If the environment provides the process with input, then the process will evolve to P at the next time step. If no input is received, then the process evolves to chaos at the next time step.

We have one axiom involving output prefix:

a–5 : $[!B?X \rightarrow P_X] \sqcap [!B?Y \rightarrow Q_Y] \equiv [!B?Z \rightarrow (P_Z \sqcap Q_Z)]$

This is weaker that the corresponding axiom in SCSP. Consider the process

$$[!B?X \rightarrow P_X] \sqcap [!C?Y \rightarrow Q_Y]$$

All the events in the prefix set must occur at the first time step. So whenever $B \neq C$, an observer will be able to establish which way the process resolved the choice at the first time step, simply by considering the output which occurred. The only situation in which the resolution of choice can be postponed is when both prefix sets are equal, which justifies the distributivity result, a-5.

Parallel composition

The parallel composition of two processes, $P \parallel Q$, is the process which results from the interaction between two concurrently executing processes. Parallel composition is defined for processes with disjoint output alphabets: $P \parallel Q$ is defined if $oP \cap oQ = \{\}$. The input and output alphabets of $P \parallel Q$ are $(\iota P \cup \iota Q) - (oP \cup oQ)$ and $(oP \cup oQ)$. Synchronisation must occur on common events, in the sense that such common events can only occur when both component processes are prepared to perform them. The occurrence of other events in the composition is governed by the behaviour of the component process which contributed these events. If one process becomes chaotic and can provide no further information about its behaviour, then the composition also becomes uninformative. Communication between component processes, resulting from output from one component being received as input to the other component, is seen as output of the composed process. This allows us to model forks in wires easily.

Parallel composition satisfies the following axioms:

a–6 : $\bot_{I,O} \parallel P \equiv \bot_{I',O'}$ where $O' = (oP \cup O)$ and $I' = (\iota P \cup I) - O'$

a–7 : $P \parallel \bot_{I,O} \equiv \bot_{I',O'}$ where $O' = (oP \cup O)$ and $I' = (\iota P \cup I) - O'$

a–8 : $(P \sqcap Q) \parallel R \equiv (P \parallel R) \sqcap (Q \parallel R)$

a–9 : $P \parallel (Q \sqcap R) \equiv (P \parallel Q) \sqcap (P \parallel R)$

a–10 : $[!B?X \rightarrow P_X] \parallel [!C?Y \rightarrow Q_Y] \equiv$
$[!(B \cup C)?Z \rightarrow P_{(Z \cup C) \cap \iota P_{\{\}}} \parallel Q_{(Z \cup B) \cap \iota Q_{\{\}}}]$

and the following laws:

l–1 : $P \parallel Q \equiv Q \parallel P$

l–2 : $(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$

Hiding

As in SCSP we should be able to change the level of abstraction of a problem by hiding events from the environment. Since input events form the external control of a process, it does not make sense to be able to hide input events. The only events in a process' alphabet which may be hidden are output events; this corresponds to ignoring the information provided by the process. A hidden event will occur unseen whenever the process desires to perform it. We note that, unlike SCSP, hiding does not *introduce* maximal progress implications. Rather, all output in a receptive system already occurs as soon as possible.

If B is a set of events disjoint from the input alphabet of P such that $\alpha P - B \neq \{\}$, then $P \setminus B$ is the process which behaves as P, with all the events in B occurring unseen by the environment. Notice, we are always able to hide events which are the result of communication between processes in parallel composition, since such communication is visible as output.

Hiding satisfies the following axioms:

a–11 : $\quad \bot_{I,O} \setminus B \equiv \bot_{I,O-B}$

a–12 : $\quad (P \sqcap Q) \setminus A \equiv (P \setminus A) \sqcap (Q \setminus A)$

a–13 : $\quad [!B?X \rightarrow P_X] \setminus A \equiv [!(B - A)?X \rightarrow (P_X \setminus A)]$

and the following laws:

l–3 : $\quad (P \setminus A) \setminus B \equiv P \setminus (A \cup B)$

l–4 : $\quad (P \parallel Q) \setminus A' \equiv (P \setminus A') \parallel (Q \setminus A')$
if $A' \cap \alpha P \cap \alpha Q = \{\}$

Renaming

Given a bijective renaming function $S : \Sigma \rightarrow \Sigma$, we use $P[S]$ to denote a renaming of process P. Process $P[S]$ has input alphabet $(\iota P)[S]$ and output alphabet $(o P)[S]$, where for a subset B of Σ we define $B[S] \mathrel{\hat{=}} \{S(e) \mid e \in B\}$. Like SCSP, $P[S]$ performs event $S(a)$ in exactly the circumstances that P would perform event a.

Renaming satisfies the following axioms:

a–14 : $\quad \bot_{I,O}[S] \equiv \bot_{I[S],O[S]}$

a–15 : $\quad (P \sqcap Q)[S] \equiv P[S] \sqcap Q[S]$

a–16 : $\quad [!B?X \rightarrow P_X][S] \equiv [!B[S]?Y \rightarrow (P_{Y[S^{-1}]})[S]]$

and the following laws:

l–5 : $\quad P[S][R] \equiv P[R \cdot S]$

l–6 : $\quad (P \parallel Q)[S] \equiv P[S] \parallel Q[S]$

l–7 : $\quad (P \setminus B)[S] \equiv P[S] \setminus B[S]$

5.1.2 Recursion

By the use of recursion we are able to extend our language to describe infinite processes. $\mu x : I, O \bullet P$ represents the solution of the recursive definition of the process x defined as a particular (least) fixed point of the function $\lambda x \bullet P$. The mathematical details of this construction will be presented later.

a–17 : $\quad \mu x : I, O \bullet P \equiv P[(\mu x : I, O \bullet P)/x]$

here $P[(\mu x : I, O \bullet P)/x]$ denotes the process P with $\mu x : I, O \bullet P$ substituted for every free occurrence of the variable x.

Recursion also satisfies alpha reduction:

l–8 : $\quad \mu x : I, O \bullet P \equiv \mu y : I, O \bullet P[y/x] \qquad$ where y is not a variable in P.

By the same argument as for SCSP, we can show that there is a unique fixed point of the function $\lambda x \bullet P$ whenever every occurrence of x in P is directly or indirectly guarded by an output prefix.

5.1.3 Derived processes and operators

Many of the derived processes and operators of SCSP cannot be expressed in SRPT. Processes in SRPT can always accept input; they cannot wait for time to pass without making input events available. The only process which can be viewed as a unit of parallel composition, in the sense that *RUN* could be in SCSP, must have an empty output alphabet. This is a consequence of the alphabet restrictions on parallel composition. Due to the requirements on input events, the only non-divergent process with input alphabet I and empty output alphabet is $STOP_{I,\{\}}$, which is presented below.

Stop

The process $STOP_{I,O}$ never outputs and never becomes chaotic; it represents a deadlocked process. Like all other processes of SRPT, $STOP_{I,O}$ can always accept input; in this respect it resembles the deadlocked process in the theory of Asynchronous processes presented in [JHH89].

$$STOP_{I,O} \mathrel{\widehat{=}} \mu x : I, O \bullet [!\{\}?X \rightarrow x].$$

We have the following law:

l–9 : $\quad STOP_{I_1,O_1} \parallel STOP_{I_2,O_2} \equiv STOP_{I,O} \qquad$ where $O = O_1 \cup O_2$
$I = (I_1 \cup I_2) - O$

Proof:

$STOP_{I_1,O_1} \parallel STOP_{I_2,O_2}$

$\equiv$ { defn of $STOP$ and by a–17 }

$[!\{\}?X \rightarrow STOP_{I_1,O_1}] \parallel [!\{\}?Y \rightarrow STOP_{I_2,O_2}]$

$\equiv$ { by a–10 }

$[!\{\}?Z \rightarrow (STOP_{I_1,O_1} \parallel STOP_{I_2,O_2})]$

$\equiv$ { by uniqueness of solutions to guarded recursive equations }

$STOP_{I,O}$ □

If the output alphabet is empty, then $STOP_{I,\{\}}$ is prepared to perform any of the events in its alphabet. We obtain the following law:

l–10 : $STOP_{I,\{\}} \parallel P \equiv P$ if $I \subseteq \alpha P$

5.2 Example: Basic digital logic circuits

In this section we draw on the field of digital circuit design to provide some small examples of the use of SRPT. Throughout these examples we shall model components by recording voltage-levels on labelled input and output wires. A component is represented by a process with input events corresponding to input wires and output events corresponding to output wires. Event a occurring corresponds to the voltage-level on the wire 'a' being high; otherwise the voltage-level is assumed to be low. We shall assume that initially all the wires in the system are low. Throughout these examples we shall assume there is a delay associated with the gates modelled, in that time must elapse between the provision of input and the observation of the desired output corresponding to this input. In reality there is no significant delay associated with the simple gates described here.

5.2.1 Gates

AND gate The output of an AND gate with unit delay is only high when both the inputs were high at the previous time step.

$$AND \triangleq [!\{\}?X \rightarrow (AND' \textbf{ if } X = \{a,b\} \textbf{ else } AND)]$$
$$AND' \triangleq [!\{c\}?X \rightarrow (AND' \textbf{ if } X = \{a,b\} \textbf{ else } AND)]$$

OR gate The output of an OR gate is only low when both inputs were low at the previous time step.

$$OR \triangleq [!\{\}?X \rightarrow (OR \textbf{ if } X = \{\} \textbf{ else } OR')]$$
$$OR' \triangleq [!\{c\}?X \rightarrow (OR \textbf{ if } X = \{\} \textbf{ else } OR')]$$

EXOR gate The output of an EXOR (exclusive or) gate is only high when exactly one of the inputs was high at the previous time step.

$$EXOR \triangleq [!\{\}?X \rightarrow (EXOR' \textbf{ if } |X| = 1 \textbf{ else } EXOR)]$$
$$EXOR' \triangleq [!\{c\}?X \rightarrow (EXOR' \textbf{ if } |X| = 1 \textbf{ else } EXOR)]$$

5.2.2 Half-adder

A circuit for calculating the sum s and carry c of two bits a and b can be constructed from two gates as shown in Figure 5.3. So the process describing the half-adder can be defined as

$$HA \triangleq AND \parallel EXOR[s/c]$$

We shall expand the definition, eliminating parallel composition and renaming from the expressions. This way we are able to demonstrate that the circuit has the desired behaviour.

Firstly we define:

$$HA_1 \triangleq AND \parallel EXOR'[s/c]$$

and $$HA_2 \triangleq AND' \parallel EXOR[s/c]$$

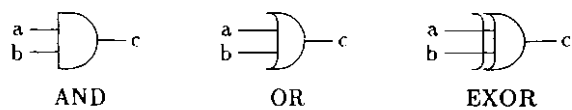


Figure 5.2: Three gates, all with input wires a, b and output wire c.

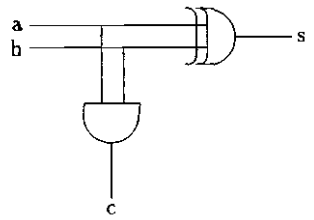


Figure 5.3: A half-adder

Now we have

$$
\begin{array}{ll}
 & HA \\
\equiv & \{ \text{ by definition } \} \\
 & [!\{\}?X \rightarrow (AND' \textbf{ if } X = \{a,b\} \textbf{ else } AND)] \\
 & \quad \| [!\{\}?X \rightarrow (EXOR' \textbf{ if } |X| = 1 \textbf{ else } EXOR)][s/c] \\
\equiv & \{ \text{ by a-16 } \} \\
 & [!\{\}?X \rightarrow (AND' \textbf{ if } X = \{a,b\} \textbf{ else } AND)] \\
 & \quad \| [!\{\}?X \rightarrow (EXOR'[s/c] \textbf{ if } |X| = 1 \textbf{ else } EXOR[s/c])] \\
\equiv & \{ \text{ by a-10 } \} \\
 & [!\{\}?X \rightarrow (AND' \parallel EXOR[s/c] \textbf{ if } X = \{a,b\} \textbf{ else} \\
 & \qquad (AND \parallel EXOR[s/c] \textbf{ if } X = \{\} \textbf{ else } AND \parallel EXOR'[s/c]))] \\
\equiv & \{ \text{ by the above definitions } \} \\
 & [!\{\}?X \rightarrow (HA_2 \textbf{ if } X = \{a,b\} \textbf{ else} \\
 & \qquad (HA \textbf{ if } X = \{\} \textbf{ else } HA_1))]
\end{array}
$$

Continuing in this manner we can show:

$$HA \equiv [!\{\}?X \rightarrow HA(X)]$$

where

$$HA(X) \hat{=} HA_2 \textbf{ if } X = \{a,b\} \textbf{ else } (HA \textbf{ if } X = \{\} \textbf{ else } HA_1)$$

$$HA_1 \equiv [!\{s\}?X \rightarrow HA(X)]$$
$$HA_2 \equiv [!\{c\}?X \rightarrow HA(X)]$$

This has the behaviour of a half-adder with unit delay.

Up to now we have only considered combinatorial circuits with unit delay. In such circuits output is a function of the previous input alone. Thus these combinatorial circuits can be represented by processes with general form:

$$P_Y \equiv [!f(Y)?X \rightarrow P_X]$$

where $X, Y \subseteq \iota P$ and $f : \iota P \rightarrow oP$.
In the next section we shall consider circuits which have state.

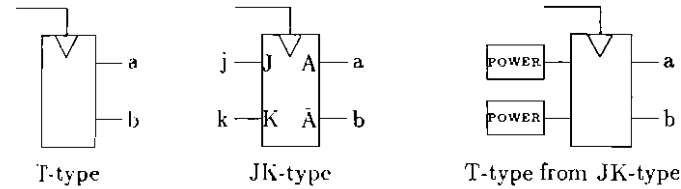


Figure 5.4: Flipflops

5.2.3 Clocked flipflops

Flipflops are one of the basic components of computer memory. Flipflops are clocked components which are typically triggered by the rising edge of the clock cycle. The output of a flipflop is determined by its own state and the state of its input wires on the rising edge of the clock cycle. It determines the state of its output wires before the following rising edge.

By assuming that the 'tick' of the clock in our language corresponds to the rising edge of the clock cycle in the circuit we can provide a representation of flipflops by recording the value on their input and output wires at these 'ticks'. We shall consider the T-flipflop and the JK-flipflop and show how the former can he derived from the latter, a well-known result.

T-type

The output of a T-type flipflop toggles between a and b with every clock pulse.

$$\iota T = \{\} \qquad oT = \{a, b\}$$
$$T \triangleq [!\{a\} \rightarrow T']$$
$$T' \triangleq [!\{b\} \rightarrow T]$$

JK-type

Output A is set high when J=1 and K=0 and reset low when J=0 and K=1. When both J and K are high the output toggles and when both J and K are low the output remains unchanged.

$$\iota JK = \{j, k\} \qquad oJK = \{a, b\}$$
$$JK \triangleq [!\{a\}?X \rightarrow (JK' \textbf{ if } k \in X \textbf{ else } JK)]$$
$$JK' \triangleq [!\{b\}?X \rightarrow (JK \textbf{ if } j \in X \textbf{ else } JK')]$$

Deriving a T-type from a JK-type

Clearly by holding both J and K high we can construct a T-type from a JK-type flipflop. So all we need to do is attach both inputs to power. Power consists of a single outpnt which is always high.

$$POWER \triangleq [!\{a\} \rightarrow POWER] \qquad \iota POWER = \{\} \quad o POWER = \{a\}$$

So consider

$$(JK \parallel POWER[j/a] \parallel POWER[k/a]) \setminus \{j,k\}$$

First consider

$$\begin{array}{ll} & POWER[j/a] \parallel POWER[k/a] \\ \equiv & \{ \text{ expanding definition of } POWER \ \} \\ & [!\{a\} \rightarrow POWER[j/a]] \| [!\{a\} \rightarrow POWER][k/a] \\ \equiv & \{ \text{ by a-16 } \} \\ & [!\{j\} \rightarrow (POWER[j/a])] \| [!\{k\} \rightarrow (POWER[k/a])] \\ \equiv & \{ \text{ by a-10 } \} \\ & [!\{j,k\} \rightarrow ((POWER[j/a]) \| (POWER[k/a]))] \end{array}$$

So by uniqueness of guarded recursive equations

$$POWER[j/a] \parallel POWER[k/a] \equiv P$$

where $P \triangleq [!\{j,k\} \rightarrow P]$

Thus

$$\begin{array}{ll} & (JK \parallel POWER[j/a] \parallel POWER[k/a]) \setminus \{j,k\} \\ \equiv & \{ \text{ substituting } P \ \} \\ & (JK \parallel P) \setminus \{j,k\} \\ \equiv & \{ \text{ expanding processes } \} \\ & [!\{a\}?X \rightarrow (JK' \textbf{ if } k \in X \textbf{ else } JK)] \| [!\{j,k\} \rightarrow P] \setminus \{j,k\} \\ \equiv & \{ \text{ by a-10 } \} \\ & [!\{a,j,k\} \rightarrow (JK' \parallel P)] \setminus \{j,k\} \\ \equiv & \{ \text{ by a-13 } \} \\ & [!\{a\} \rightarrow (JK' \parallel P) \setminus \{j,k\}] \\ \equiv & \{ \text{ continuing expansion } \} \\ & [!\{a\} \rightarrow [!\{b\} \rightarrow (JK \parallel P) \setminus \{j,k\}]] \end{array}$$

So by the uniqueness of solutions to guarded recursive equations:

$$(JK \parallel POWER[j/a] \parallel POWER[k/a]) \setminus \{j,k\} \equiv T.$$

We shall return to the modelling of clocked circuits in Chapter 7.

5.3 Semantic Model

In this section we present a denotational semantics for the language SPRT. The semantic model makes a distinction between input and output events. This and the receptive nature of the language results in it being unnecessary to record refusal information. A process cannot refuse to perform input events. Output events are not blocked, so if an output event does not occur the process must be refusing to perform it. This results in a model which is far simpler than that presented for SCSP in Chapter 3. The new model forms a complete partial order under an information ordering presented in Section 5.3.3.

5.3.1 Notation

Here we introduce the key concepts of the model.

Events

An event is either an input to a process from the environment, or an output from a process. We denote the *universal alphabet* Σ to be the set of all possible events.

A particular process may participate in a finite number of input and output events $I, O \subseteq \Sigma$; I is the input alphabet of the process, while O is its output alphabet. The input and output alphabets are necessarily disjoint, $I \cap O = \{\}$, and at least one of these is non-empty, $I \cup O \neq \{\}$.

Within our model we shall record the set of events performed at a given time step. We shall refer to such a set as an *occurrence-set*, it is a subset of $I \cup O$. The receptive nature of our model means that any input event which was not observed, was not offered by the environment, while any output event which was not observed, was not made available by the process.

Traces

A trace is a finite sequence of occurrence-sets. Given input and output alphabets, I and O, the set of all traces is given by:

$$RT_{I,O} \triangleq (\mathbb{P}(I \cup O))^*$$

At each tick of a global clock an observer may witness a number of events from the set $I \cup O$. By recording these sets of occurrences in a chronologically ordered sequence we obtain a trace of the process. As in the model for SCSP, time is recorded implicitly, times when nothing occurred being marked by the empty set in the trace.

As traces take the form of sequences of sets we shall continue to use the operators developed for traces in Section 3.1.1.

Maximal behaviours

If a process is divergent it can give no useful information about its behaviour. We shall record nothing about the behaviour of a process once it has become divergent. If a process diverges after exhibiting behaviour s then no extension of this behaviour is a behaviour of the process. No further record of the passage of time is made. This is comparable to the use of time stops by Moller and Tofts [MT90] to model an undesirable state.

A behaviour which precedes a divergent state is maximal in that no extension of this behaviour is recorded in the model.

Definition 5.1 The set of *maximal behaviours* of a process with trace set T is given by:

$$\widehat{T} \triangleq \{s \in T \mid \neg\ \exists r \in T \bullet r > s\}$$

◊

For any set of traces T, $\widehat{T}$ is a *maximal set*.

Processes

A process P is represented in our model by the triple (I, O, T) where I is the input alphabet, O is the output alphabet and $T = T(P)$ is the set of all traces describing possible behaviours of the process. Only subsets of $RT_{I,O}$ satisfying the closure conditions to be given in Section 5.3.2 will represent trace sets of processes.

Restriction

Definition 5.2 We can take the *restriction* of a trace set, T, by a (maximal) set of traces, S, to obtain a trace set $T \downarrow S$,

$$T \downarrow S \triangleq \{s \in T \mid \neg\ (\exists r \in S \bullet r < s)\}$$

◊

$T \downarrow S$ consists of the behaviours of T which are not extensions of any behaviour in S. A process with trace set $T \downarrow S$ diverges more often than one with trace set T. If $s \in S$ is a trace in T, then either s is a maximal behaviour in $T \downarrow S$ and immediately precedes divergence, or there is a trace $s' < s$ which is a maximal behaviour in $T \downarrow S$ so s is not a behaviour of $T \downarrow S$. If S is a maximal set then only the former of the above cases applies.

The following are direct consequences of the definition of maximal behaviours and restriction.

Lemma 5.1 *If T, S, R and M are trace sets and M is a maximal set then:*

1. $T \downarrow \widehat{T} = T$
2. $T \downarrow R \subseteq T$
3. $T \subseteq S \Rightarrow T \downarrow R \subseteq S \downarrow R$
4. $T \subseteq S \downarrow \widehat{T} \Leftrightarrow T \subseteq S$
5. $M \subseteq S \wedge T = S \downarrow M \Rightarrow M \subseteq \widehat{T}$
6. $(T \downarrow R) \downarrow S = (T \downarrow S) \downarrow R$
7. *restriction by S is idempotent.*
8. *restriction by S distributes through union and intersection.*

■

5.3.2 Closure conditions

In this section we introduce closure conditions on a set T of traces which must be satisfied for T to represent the trace set of a process with input and output alphabets I and O. We notice that the first two conditions correspond to conditions i and ii of SCSP. The elimination of refusal information from the model results in a reduction in the number of closure conditions required to just three.

I $\langle\rangle \in T$

The empty trace is observable at time 0.

II $s^\frown r \in T \Rightarrow s \in T$

Prefix closure; if a particular traces can be observed over a certain time span, then prefixes of this trace, corresponding to observation made for a shorter time, may also be observed.

III $s^\frown(X) \in T \wedge Y \subseteq I \Rightarrow s^\frown((X \cap O) \cup Y) \in T$

At each time step the process can accept any input. This reflects the assumption that the process is always receptive to any input. We also notice that the output performed at a given time step is not influenced by input received at that time.

We shall let RM be the set of all triples (I, O, T) where I and O are finite disjoint input and output alphabets and T satisfies the closure conditions with respect to I and O. This set is the underlying model for our receptive language.

$$RM \triangleq \{(I, O, T') \mid I, O \in \mathbb{F}(\Sigma) \wedge I \cap O = \{\} \wedge I \cup O \neq \{\} \\ \wedge\ T' \subseteq RT_{I,O} \wedge T \text{ satisfies conditions I–III}\}$$

Where Σ is the universal set of all events.

We use $RM^{I,O}$ to denote the set of all processes with input alphabet I and outpnt alphabet O.

$$RM^{I,O} \triangleq \{(I, O, T') \mid (I, O, T') \in RM\}$$

Furthermore we let RM_T be the set of all sets of traces for processes and $RM_T^{I,O}$ be the subset of RM_T corresponding to processes with input and output alphabets I and O.

$$RM_T \triangleq \{T' \mid \exists I, O \in \mathbb{F}\,\Sigma \bullet (I, O, T') \in RM\}$$
$$RM_T^{I,O} \triangleq \{T' \mid (I, O, T') \in RM^{I,O}\}$$

We notice that $RM_T^{I,O}$ is closed under restriction by any subset of $RT_{I,O}$.

5.3.3 Information ordering

The natural ordering on the model for SCSP was a non-determinism ordering. Due to the novel representation of divergence in the model for SRPT, a non-determinism ordering, as presented in Section 3.1.3, is no longer the most natural ordering to work with. We define a new *information ordering* on processes with the same alphabets.

If (I, O, T_P) and (I, O, T_Q) represent two processes P and Q, then we define the ordering $\leqslant$ by:

$$(I, O, T_P) \leqslant (I, O, T_Q) \triangleq T_Q \downarrow \widehat{T_P} = T_P$$

This relation does not give an ordering between any non-divergent processes. The information ordering has the same philosophy as Roscoe's definedness ordering [Ros88a]. Q is more reliable than P, $P \leqslant Q$; any behaviour of P is a behaviour of Q; moreover any behaviour of Q is either a behaviour of P or an extension of a maximal behaviour of P.

Lemma 5.2 *If (I, O, T_P) and (I, O, T_Q) represent two processes P and Q then*

$$P \leqslant Q \Leftrightarrow T_P \subseteq T_Q \\ \wedge\ (r \in T_Q \wedge r \notin T_P \Rightarrow \exists s \in \widehat{T_P} \bullet s \leqslant r).$$

■

Since the ordering is only on processes with the same alphabets we can consider it as an ordering on $RM_T^{I,O}$.

Lemma 5.3 *The least element of $RM_T^{I,O}$ under the information ordering is $\{\langle\rangle\}$.* ■

$\{\langle\rangle\}$ is the trace set of the process which diverges immediately, so never gives any useful information.

Lemma 5.4 *Every $\leqslant$-directed set, $D \subseteq RM_T^{I,O}$, has a least upper bound in $RM_T^{I,O}$ and this least upper bound is $\bigcup D$.*

Proof: We claim that $\bigcup D \in RM_T^{I,O}$ and $\sqcup_{\leqslant} D = \bigcup D$.

The former is immediate, it remains to show that $\sqcup_{\leqslant} D = \bigcup D$.

Firstly we show that $\bigcup D$ is an upper bound, that is $\forall P \in D \bullet \bigcup D \downarrow \hat{P} = P$.

For $P \in D$ we have $s \in \bigcup D \downarrow \hat{P} \Rightarrow (\exists P' \in D \bullet s \in P') \wedge \neg (\exists r \in \hat{P} \bullet r < s)$.

Choose $P_0 \in D$ with $s \in P_0$. As D is directed, choose Q with $P \leqslant Q$ and $P_0 \leqslant Q$.

$$Q \downarrow \widehat{P_0} = P_0 \Rightarrow s \in Q, \quad \{ \text{ since } s \in P_0 \ \}$$
$$Q \downarrow \hat{P} = P \Rightarrow s \in P, \quad \left\{ \text{ since } s \in Q \text{ and as } \neg (\exists r \in \hat{P} \bullet r < s) \ \right\}$$

Hence $\bigcup D \downarrow \hat{P} \subseteq P$. Moreover $P \subseteq \bigcup D$, giving $P \subseteq \bigcup D \downarrow \hat{P}$ by Lemma 5.1, as required. It remains to show that $\bigcup D$ is the least upper bound of D. Take Q an upper bound of D, so $\forall P \in D \bullet P \leqslant Q$. We must show $Q \downarrow \widehat{\bigcup D} = \bigcup D$.

Clearly $\bigcup D \subseteq Q \downarrow \widehat{\bigcup D}$. We show that $Q \downarrow \widehat{\bigcup D} \subseteq \bigcup D$ by contradiction. Suppose $s \in Q \downarrow \widehat{\bigcup D}$ and $s \notin \bigcup D$. Then $s \in Q$ and $\forall P \in D \bullet s \notin P$, giving $(\forall P \in D \bullet \exists r \in \hat{P} \bullet r < s)$, as Q is an upper bound of D. Now consider the set

$$T \triangleq \{r \mid \exists P \in D \bullet r \in \hat{P} \wedge r < s\}$$

This is clearly non-empty and finite as s is of finite length, so we can take the maximum of this set r'. We claim that $r' \in \widehat{\bigcup D}$. We know that $r' \in \bigcup D$, if it is not maximal then we can find $r_0 \in \bigcup D$ such that $r_0 > r'$. We can choose $P \in D$ such that $r_0 \in P$, but we know that there exists an $r_1 \in T$ with $r_1 \in \hat{P}$. So $r_0 > r' \geq r_1$ contradicting the definition of $\hat{P}$. Hence $r' \in \widehat{\bigcup D}$ and $r' < s$. □

Theorem 5.5 *$(RM^{I,O}, \leqslant)$ forms a complete partial order.*

Proof: Follows from Lemma 5.3 and Lemma 5.4. □

Lemma 5.6 *Every non-empty subset of $RM_T^{I,O}$ has a $\leqslant$-greatest lower bound.*

Proof: Suppose S is a non-empty subset of $RM_T^{(I,O)}$. We define

$$K \triangleq \{s \in \bigcap S \mid \exists P, Q \in S, X \subseteq I \cup O \bullet s^\frown\langle X\rangle \in P - Q\}$$

and

$$K' \triangleq \{s \in K \mid \neg\ \exists r \in K \bullet r < s\}.$$

We claim that $S' \triangleq (\bigcap S) \downarrow K'$ is the greatest lower bound of S in $RM_T^{(I,O)}$.

S' satisfies conditions I-III, thus $S' \in RM_T^{(I,O)}$. We must show that $S' = \sqcap_{\leqslant} S$

Firstly we show that S' is a lower bound of S, that is $\forall P \in S \bullet S' = P \downarrow \widehat{S'}$

$S' \subseteq P \downarrow \widehat{S'}$ since $S' \subseteq P$. We shall show that $P \downarrow \widehat{S'} \subseteq S'$ by contradiction. Suppose $s \in P \downarrow \widehat{S'}$ and $s \notin S'$

case $s \in \bigcap S$: As $s \notin S'$ there exists $r \in K'$ such that $r < s$. Now by construction $K' \subseteq \widehat{S'}$ so $\exists r \in \widehat{S'} \bullet r < s$. Thus $s \notin P \downarrow \widehat{S'}$ contradicting our original assumption.

case $s \notin \bigcap S$: We can find a prefix $r^\frown\langle X\rangle \leq s$ with $r \in \bigcap S$ and $r^\frown\langle X\rangle \notin \bigcap S$. So we can choose $Q \in S$ such that $r^\frown\langle X\rangle \notin Q$. Hence $r \in K$ and $\exists r' \in K' \bullet r' \leq r < s$. Since $K' \subseteq \widehat{S'}$ we have that $\exists r' \in \widehat{S'} \bullet r' < s$. Thus $s \notin P \downarrow \widehat{S'}$. Hence result by contradiction.

It remains to show that S' is the greatest lower bound of S.

Take Q a lower bound of S, so $\forall P \in S \bullet P \downarrow \widehat{Q} = Q$. We must show $S' \downarrow \widehat{Q} = Q$.

$S' \downarrow \widehat{Q} \subseteq Q$, by the construction of S'. We show that $Q \subseteq S' \downarrow \widehat{Q}$ by contradiction.

Suppose $s \in Q$ and $s \notin S' \downarrow \widehat{Q}$. Then $s \notin S'$ and $s \in Q \Rightarrow \forall P \in S \bullet s \in P \Rightarrow s \in \bigcap S$. So $\exists r \in K' \bullet r < s$. We can choose $P_0, P_1 \in S$ and $X \subseteq (I \cup O)$ such that $r^\frown\langle X\rangle \in P_0 - P_1$. Since $s \in Q$, no prefix of s is maximal in Q, so $r^\frown\langle X\rangle \in P_0 \downarrow \widehat{Q}$, thus $r^\frown\langle X\rangle \in Q$. However, $r^\frown\langle X\rangle \in Q \Rightarrow r^\frown\langle X\rangle \in \bigcap S \Rightarrow r^\frown\langle X\rangle \in P_1$. Contradicting the choice of P_1. □

Theorem 5.7 *$(RM^{I,O}, \leqslant)$ forms a complete semi-lattice.*

Proof: This follows from Theorem 5.5 and Lemma 5.6. □

Lemma 5.8 *If $D \subseteq RM_T^{I,O}$ is a $\leqslant$-directed set, then*

$$\widehat{\bigcup D} = \{s \in \bigcup_{P \in D} \hat{P} \mid \forall P \in D \bullet (\exists s' \in \hat{P} \bullet s' \leq s)\}$$

■

5.4 Semantic Function

In this section we construct a semantic function which maps syntactic expressions of our language to processes in our model RM. It is necessary to consider each process term with a specific binding of process variables to processes.

Variable bindings

Given a set of variables Var, we define a domain of bindings, $BIND_R$, this consists of all mappings from Var to the space of processes RM.

$$BIND_R \triangleq Var \to RM.$$

Now we are able to define a semantic function:

$$\mathcal{M}_R : SRPT \to BIND_R \to RM$$

$\mathcal{M}_R[\![P]\!]\sigma$ denotes the meaning of process term P with variable binding σ, in terms of our model. This is evaluated by associating each free variable x with its value $\sigma[\![x]\!]$ in binding σ.

Semantic substitution of free variables occurs in the same sense as in Section 3.2. As before, when reasoning about closed process terms, that is those with no free variables, it is unnecessary to make the variable bindings explicit.

The semantic function $\mathcal{M}_R$

Given a variable binding, $\mathcal{M}_R$ maps each process term to a triple representing the process' input alphabet, output alphabet and the set of traces of the process. We define ι, o, and $\mathcal{T}_R$ to be the natural projections onto the first, second and last component of this triple.

$$\mathcal{M}_R[\![P]\!]\sigma = (\iota[\![P]\!]\sigma, o[\![P]\!]\sigma, \mathcal{T}_R[\![P]\!]\sigma)$$

For a general process both alphabets and the set of traces of the process will depend upon the variable binding.

$$\iota, o : SPRT \to BIND_R \to \mathbb{F}\,\Sigma$$
$$\mathcal{T}_R : SPRT \to BIND_R \to RM_T$$

Non-recursive processes

We define $\mathcal{M}_R$ over the non-recursive terms of SRPT by defining the projections ι, o and $\mathcal{T}_R$. We take SRPT^g to be the restriction of SRPT to the non-recursive terms, that is the terms with syntax:

$$P ::= \bot_{\iota,o} \mid x \mid P \sqcap P \mid [!B?X \to P_X] \mid P \parallel P \mid P \setminus A \mid P[S]$$

Definition 5.3 The functions ι and o are defined as follows over the syntax of SRPT^{θ}.

$$\iota[\![\ \bot_{I,O}\ ]\!]\sigma \mathrel{\hat{=}} I$$
$$o[\![\ \bot_{I,O}\ ]\!]\sigma \mathrel{\hat{=}} O$$

$$\iota[\![x]\!]\sigma \mathrel{\hat{=}} \pi_1\sigma[\![x]\!]$$
$$o[\![x]\!]\sigma \mathrel{\hat{=}} \pi_2\sigma[\![x]\!]$$

$$\iota[\![P[S]]\!]\sigma \mathrel{\hat{=}} (\iota[\![P]\!]\sigma)[S]$$
$$o[\![P[S]]\!]\sigma \mathrel{\hat{=}} (o[\![P]\!]\sigma)[S]$$

if $\iota[\![P]\!]\sigma = \iota[\![Q]\!]\sigma$ and $o[\![P]\!]\sigma = o[\![Q]\!]\sigma$, then

$$\iota[\![P \sqcap Q]\!]\sigma \mathrel{\hat{=}} \iota[\![P]\!]\sigma$$
$$o[\![P \sqcap Q]\!]\sigma \mathrel{\hat{=}} o[\![P]\!]\sigma$$

if $B \subseteq o[\![P_{\{\}}]\!]\sigma$ and $\forall\, C \subsetneq \iota[\![P_{\{\}}]\!] \bullet \iota[\![P_C]\!]\sigma = \iota[\![P_{\{\}}]\!]\sigma \wedge o[\![P_C]\!]\sigma = o[\![P_{\{\}}]\!]\sigma$, then

$$\iota[\![[!B?X \to P_X]]\!]\sigma \mathrel{\hat{=}} \iota[\![P_{\{\}}]\!]\sigma$$
$$o[\![[!B?X \to P_X]]\!]\sigma \mathrel{\hat{=}} o[\![P_{\{\}}]\!]\sigma$$

if $\iota[\![P]\!]\sigma \cap \iota[\![Q]\!]\sigma = \{\}$, then

$$\iota[\![P \parallel Q]\!]\sigma \mathrel{\hat{=}} (\iota[\![P]\!]\sigma \cup \iota[\![Q]\!]\sigma) - (o[\![P]\!]\sigma \cup o[\![Q]\!]\sigma)$$
$$o[\![P \parallel Q]\!]\sigma \mathrel{\hat{=}} o[\![P]\!]\sigma \cup o[\![Q]\!]\sigma$$

if $B \cap \iota[\![P]\!]\sigma = \{\}$ and $\iota[\![P]\!]\sigma \cup o[\![P]\!]\sigma - B \neq \{\}$, then

$$\iota[\![P \setminus B]\!]\sigma \mathrel{\hat{=}} \iota[\![P]\!]\sigma$$
$$o[\![P \setminus B]\!]\sigma \mathrel{\hat{=}} o[\![P]\!]\sigma - B$$

◊

Definition 5.4 The function $\mathcal{T}_{\mathcal{R}}$ is defined as follows * over the syntax of SRPT^{θ}.

$$\mathcal{T}_{\mathcal{R}}[\![\ \bot_{I,O}\ ]\!]\sigma \mathrel{\hat{=}} \{\langle\rangle\}$$

$$\mathcal{T}_{\mathcal{R}}[\![x]\!]\sigma \mathrel{\hat{=}} \pi_S\sigma[\![x]\!]$$

$$\mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \mathrel{\hat{=}} (\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma) \cup (\mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma)$$

$$\mathcal{T}_{\mathcal{R}}[\![[!B?X \to P_X]]\!]\sigma \mathrel{\hat{=}} \{\langle B \cup Y\rangle^\frown s \mid Y \subseteq I \wedge s \in \mathcal{T}_{\mathcal{R}}[\![P_Y]\!]\sigma\} \cup \{\langle\rangle\}$$
$$\text{where } I = \iota[\![[!B?X \to P_X]]\!]\sigma$$

$$\mathcal{T}_{\mathcal{R}}[\![P \parallel Q]\!]\sigma \mathrel{\hat{=}} \{s \mid s \cap A \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \wedge s \cap B \in \mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma\}$$
$$\text{where } A = \iota[\![P]\!]\sigma \cup o[\![P]\!]\sigma \quad B = \iota[\![Q]\!]\sigma \cup o[\![Q]\!]\sigma$$

$$\mathcal{T}_{\mathcal{R}}[\![P \setminus B]\!]\sigma \mathrel{\hat{=}} \{s - B \mid s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma\} \downarrow \{r - B \mid r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma\}$$

$$\mathcal{T}_{\mathcal{R}}[\![P[S]]\!]\sigma \mathrel{\hat{=}} \{s \mid s[S^{-1}] \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma\}$$

◊

*We write $\hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma$ for the set of maximal behaviours of $\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma$

Notes

1. As for SCSP, $\perp_{I,O}$ is modelled by the least element in the partial order, in the case of SRPT this partial order is $(RM^{I,O}, \leqslant)$. This corresponds to $\perp_{I,O}$ being the least informative process with alphabets I and O.

2. Non-deterministic choice cannot be defined as easily as in SCSP. This is because we no longer model divergent processes as arbitrarily non-deterministic processes. The restriction in the expression here ensures that undefined behaviour ensues if both components can behave in a manner given by trace s, and s is a maximal behaviour of one of the component processes.

   The information ordering is weaker that a non-determinism ordering, if we define the non-determinism ordering in terms of the non-deterministic choice operator as for SCSP:

   $$\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!] \mathrel{\hat{=}} \mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!] = \mathcal{M}_{\mathcal{R}}[\![P]\!]$$

   then $\quad \mathcal{M}_{\mathcal{R}}[\![P]\!] \leqslant \mathcal{M}_{\mathcal{R}}[\![Q]\!] \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]$

   Proof: We can assume that the alphabets of P and Q are the same.

   $\mathcal{M}_{\mathcal{R}}[\![P]\!] \leqslant \mathcal{M}_{\mathcal{R}}[\![Q]\!]$
   $\Rightarrow$ { by definition of the ordering }
   $\mathcal{T}_{\mathcal{R}}[\![Q]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] = \mathcal{T}_{\mathcal{R}}[\![P]\!]$
   $\Rightarrow$ { trivially }
   $\mathcal{T}_{\mathcal{R}}[\![Q]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \cup \mathcal{T}_{\mathcal{R}}[\![P]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] = \mathcal{T}_{\mathcal{R}}[\![P]\!] \cup \mathcal{T}_{\mathcal{R}}[\![P]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]$
   $\Rightarrow$ { recalling definition of $\mathcal{T}_{\mathcal{R}}$ and by properties of restriction }
   $\mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!] = \mathcal{T}_{\mathcal{R}}[\![P]\!]$
   $\Rightarrow$ { as we can assume the alphabets of P and Q are the same }
   $\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]$ □

   The process with behaviours $\mathcal{T}_{\mathcal{R}}[\![P]\!] \cup \mathcal{T}_{\mathcal{R}}[\![Q]\!]$ represents an angelic non-deterministic choice between processes P and Q. This process can, whenever possible, avoid chaotic , undefined behaviour. Any implementation of this form of non-determinism would require backtracking, which is unsatisfactory. We shall not, therefore, consider this form of non-determinism any further.

3. The semantic function for parallel composition is much simpler that that for SCSP. The absence of refusal information in the model for SRPT means it is only necessary to consider synchronisation on common events. The representation of divergence by no information means that there is no need to distinguish between the cases where one process becomes divergent and neither processes become divergent. Any trace of the composed process,

when restricted to the alphabet of a component process, must represent a behaviour of that component process.

Suppose $P \parallel Q$ has behaviour s and s restricted to the alphabets of P is a maximal behaviour of P. There can be no extension of s in $P \parallel Q$, as any such extension restricted to the alphabet of P cannot be a behaviour of P. So $P \parallel Q$ has s as a maximal trace. Thus we ensure that if one component diverges, the composition must also diverge.

4. When considering the semantic function for hiding we cannot simply consider all traces with the hidden events removed – if we did, in certain circumstances a resolution of internal choice may be made so as to avoid divergence.

   Consider the process P with empty input alphabet and output alphabet $\{a, b\}$

$$P = [!\{a\} \rightarrow \bot] \sqcap [!\{\} \rightarrow [!\{a\} \rightarrow \bot]]$$

   This process has trace set $\mathcal{T}_{\mathcal{R}}[\![P]\!] = \{\langle\rangle, \langle\{a\}\rangle, \langle\{\}\rangle, \langle\{\}, \{a\}\rangle\}$ divergence occurs after one time unit if an a is output at the first time step, otherwise divergence occurs after two time units. If we hide the a we can no longer distinguish between the case where the a occurred initially, causing divergence after the first time step, and the case where nothing occurred initially, delaying divergence. In our demonic approach to non-determinism we assume the worst case occurred so:

$$P \setminus \{a\} = [!\{\} \rightarrow \bot]$$

   this process has trace set $\mathcal{T}_{\mathcal{R}}[\![P \setminus \{a\}]\!] = \{\langle\rangle, \langle\{\}\rangle\}$, demonstrating that $\mathcal{T}_{\mathcal{R}}[\![P \setminus \{a\}]\!] \neq \{s - \{a\} \mid s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\}$.

   The restriction in the semantic function for hiding is necessary to ensure that divergence is not avoided by hiding events.

Corollary 5.9 *The maximal behaviours of terms of SRPT^0 are given over the syntax as follows:*

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![\perp_{I,O}]\!]\sigma \triangleq \{\langle\rangle\}$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![x]\!]\sigma \triangleq \{s \in \pi_{\mathcal{S}}\sigma[\![x]\!] \mid \neg\, \exists s' \in \pi_{\mathcal{S}}\sigma[\![x]\!] \bullet s' > s\}$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \triangleq (\hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma) \cup (\hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma)$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![[!B?X \to P_X]]\!]\sigma \triangleq \{(B \cup Y)^\frown s \mid Y \subseteq I \wedge s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P_Y]\!]\sigma\}$$
$$\text{where } I = \iota[\![[!B?X \to P_X]]\!]\sigma$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![P \parallel Q]\!]\sigma \triangleq \{s \mid s \cap A \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma \wedge s \cap B \in \mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma$$
$$\vee\, s \cap A \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \wedge s \cap B \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma\}$$
$$\text{where } A = \iota[\![P]\!]\sigma \cup o[\![P]\!]\sigma$$
$$B = \iota[\![Q]\!]\sigma \cup o[\![Q]\!]\sigma$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![P \setminus B]\!]\sigma \triangleq \{s - B \mid s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma\} \downarrow \{r - B \mid r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma\}$$

$$\hat{\mathcal{T}}_{\mathcal{R}}[\![P[S]]\!]\sigma \triangleq \{s \mid s[S^{-1}] \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma\}$$

Proof: These all follow from the definition of $\mathcal{T}_{\mathcal{R}}$. □

Theorem 5.10 *The terms of SRPT^0 are well defined with respect to the model.*

Proof: It is necessary and sufficient that $\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma \in RM$ for all process expressions P in SRPT^0. This is demonstrated by structural induction over the syntax.

atomic terms It is clear by construction that $\perp_{I,O}$ is well defined with respect to the model. By definition of σ and as $\mathcal{M}_{\mathcal{R}}[\![x]\!]\sigma = \sigma[\![x]\!]$ the process variables are well defined.

operators It is sufficient to show that the result of applying an operator to well defined process expressions is a well defined process expression. It is a trivial exercise to verify that the semantic image of the application of an operator satisfies the closure conditions I–III in such circumstances. □

Theorem 5.11 *For P a term of SRPT^0, $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is monotonic.*

Proof: We must establish that for $q, q' \in RM$

$$q \leqslant q' \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q/x] \leqslant \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q'/x]$$

If P is an atomic process this follows trivially, either $P = \perp_{I,O}$ in which case the two expressions are constant, or P is a variable and the result is a direct consequence of the definition of variable bindings.

It is sufficient to check that each operator is monotonic in each argument, the required result then follows from the monotonicity of finite compositions of monotonic functions. The proof of monotonicity is presented as Theorem A.3 in Appendix A.2. □

Recursive processes

Definition 5.5 We extend the definition of $\mathcal{M}_{\mathcal{R}}$ to the full syntax of SRPT as follows:

$$\mathcal{M}_{\mathcal{R}}[\![\mu x : I, O \bullet P]\!]\sigma \triangleq \underline{\mathrm{fix}}_{I,O}\, \lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$$
where y does not occur free in P and
$\underline{\mathrm{fix}}_{I,O}$ denotes the function's least fixed point in $(RM^{I,O}, \leqslant)$

◊

In order to establish that $\mathcal{M}_{\mathcal{R}}$ is well defined over SPRT we must ensure that the least fixed points utilised in the above definitions exist.

Lemma 5.12 *If y is not a variable in P and $\lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is continuous in $(RM^{I,O}, \leqslant)$ for all variables x, then $\mathcal{M}_{\mathcal{R}}[\![\mu x : I, O \bullet P]\!]\sigma$ is well defined and $\lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![\mu x : I, O \bullet P]\!]\sigma[y/z]$ is continuous in $(RM^{I,O}, \leqslant)$.*

Proof: $(RM^{I,O}, \leqslant)$ is a complete semi-lattice and $\lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is, by assumptiou, continuous within the semi-lattice. So, by the Knaster-Tarski Fixed Point Theorem, a least fixed point exists. Hence $\underline{\mathrm{fix}}\, \lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is well defined.

Moreover, setting $H_R \triangleq \lambda y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$, the least fixed point is given by the limit, $\bigsqcup_{\leqslant n=0}^{\infty} H_R^n(\mathcal{M}_{\mathcal{R}}[\![\ \perp_{I,O}\ ]\!])$.

As l.u.b. preserves continuity the required result holds. □

In the next chapter we sball establish, via an embedding of RM into SM, that all the operators of SRPT^0 are continuous. Hence the following is a theorem.

Theorem 5.13 *All processes terms P of SRPT are well defined with respect to the model.* ■

5.5 Conclusion

In this chapter we have introduced the language SRPT. This language, in common with SCSP,

- expresses non-determinism, parallelism, hiding and recursion;
- captures both quantitative timing details and the notion of true concurrency through its prefix operator;
- has sufficient algebraic laws to be able to eliminate parallel composition and hiding from expressions.

In contrast to SCSP, SRPT distinguishes between input and output events; all input events are made available in the prefix construct while those output events present in the prefix construct are assumed to occur unrestricted by the environment. These differences in the language capture its receptive nature.

The semantic model for SRPT presented in this chapter is very simple; the receptive nature of the language made it unnecessary to record refusal information in the model. The behaviours of a process are captured completely by traces; each term in the trace is a set of events seen to occur simultaneously and the position of the set in the trace indicates the time it was observed. Traces in the model can be viewed as a mathematical representation of the information captured by the informal timing diagrams [Too93] often used in engineering to clarify the relationship between inputs and outputs of circuit components.

By introducing the concept of maximal behaviours and choosing an interpretation of the model in which the progression of time is not recorded after divergence, divergences were encoded into traces. This led to the consideration of an information ordering on the model in contrast to the usual non-determinism ordering. The model forms a complete partial order under this ordering, providing the mathematical structure required to underpin recursion.

Chapter 6

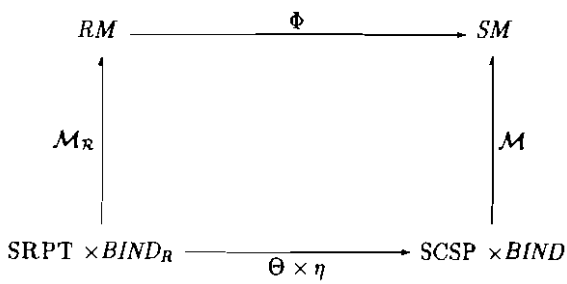
SRPT as a Sublanguage of SCSP

In this chapter we demonstrate how SRPT can be viewed as a receptive sublanguage of SCSP. To achieve this we develop two embeddings; $\Phi : RM \rightarrow SM$ which relates the two models and $\Theta : SRPT \rightarrow SCSP$ which relates the two languages. These two functions are chosen to preserve the intuitive representation of the processes, in the sense that $P \in SRPT$ and its image $\Theta P \in SCSP$ can be seen to represent the same system in the different languages. While $Q \in RM$ and its image $\Phi Q \in SM$ can be interpreted as representing the same system in different models.

We establish a natural relationship between Θ and Φ, such that, given an embedding η, of $BIND_R$ into $BIND$ induced by Φ

$$\forall P \in SRPT\,; \sigma \in BIND_R \bullet \Phi \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma = \mathcal{M}[\![\Theta P]\!]\eta\sigma.$$

Informally the following commutes:



From this we are able to draw on the results established for SCSP to demonstrate the continuity of the operators of SRPT and construct a proof system for closed terms of SRPT which is sound and complete.

6.1 Embedding RM in SM

Processes in *RM* are designed to model receptive systems, where input is never refused and the environment cannot block output, so all possible output happens. Processes in *SM* are not specifically designed to model receptive systems but we can model such systems by making a few assumptions. A process in *SM* modelling a receptive system cannot refuse input so it must always offer any subset of the input alphabet to the environment. In a receptive system, output events that can occur, do occur. We shall model this in the processes in *SM* by assuming that if a proper subset of the possible output occurs at a particular time step, then infeasible behaviour follows at the next time step. Notice that the traces of a process in *SM* corresponding to the environment allowing all possible output are the traces which are saturated with respect to the output alphabet.

Processes in *RM* model divergent or undesirable behaviour by providing no further information about the process. The system is only modelled during the time it is well behaved. By contrast, we use the concept of infeasible behaviour in *SM* to model undesirable behaviour. Despite the difference in representation, both *RM* and *SM* assume there is no possible recovery from undesirable behaviour, allowing us to make a simple correspondence in the case of divergence.

We define Φ as follows and claim it provides a suitable embedding.

Definition 6.1 Taking $A = I \cup O$, we define $\Phi : RM \rightarrow SM$ by *

$$\Phi(I, O, T) = (A, \phi(I, O, T))$$

where,

$$\phi(I, O, T) = \bigcup_{s \in T} \psi(I, O, s) \cup \bigcup_{s \in \widehat{T}} \psi_I(I, O, s)$$

$$\psi(I, O, \langle\rangle) = \{\langle\rangle\}$$

$$\psi(I, O, s^\frown\langle X\rangle) = \{s' \in ST_A \mid \exists\, r \in \mathbb{F}(\bar{O})^* \bullet (s' \cup r = \gamma_O^*(s^\frown\langle X\rangle) \vee \exists\, Y \subseteq X \cap O \bullet (Y \neq \{\} \wedge \gamma_O^*(s)^\frown\langle\gamma_O(X) - Y\rangle \leq s' \cup r))\}$$

$$\psi_I(I, O, s) = \{s'^\frown r \mid s' \in \psi(I, O, s) \wedge r \in \mathbb{F}(\bar{A})^*\}$$

and $\gamma_O(X) = X \cup (\bar{O} - \bar{X})$ is defined for $X \subseteq A$ ◊

*If $f : A \rightarrow B$ is a function then $f^* : A^* \rightarrow B^*$ is the mapping for f onto every element of a trace consisting of elements of type A. $f^*(\langle\rangle) = \langle\rangle$ $f^*(\langle x\rangle^\frown s) = \langle fx\rangle^\frown(f^*(s))$

We shall now consider how Φ maps processes in RM to those in SM. Clearly the alphabet of the process $\Phi(P)$ is the union of the input and output alphabets of P, processes in RM distinguish between events which are to be considered as output and those which are to be considered as input, processes in SM make no such distinction. It remains to consider the way in which a process in RM is mapped to a trace set in SM, this involves considering the map ϕ. Assuming $P = (I, O, T)$ we shall consider the map ϕ by decomposing it into several steps.

Firstly we saturate every trace in T with respect to output by adding refusals, we only add refusals corresponding to output which could not happen so each term in the trace remains feasible. The function γ_O performs such saturation of a set with respect to the alphabet O so we need to map this function on every trace in T giving us the set:

$$T_1 = \{\gamma_O^*(s) \mid s \in T\}$$

Next we add traces corresponding to too little output being allowed by the environment. We have already commented that infeasible behaviour will follow, so recalling closure condition vii any trace may follow. This gives us $T_1 \cup T_2$ where:

$$T_2 = \{\gamma_O^*(s)^\frown(\gamma_O(X) - Y)^\frown r \mid \{\} \subset Y \subseteq (X \cap O) \wedge s^\frown\langle X\rangle \in T \wedge r \in \mathbb{F}(\tilde{A})^*\}$$

Notice that the traces corresponding to insufficient output being allowed are not saturated. So for a process which non-deterministically outputs O_1 or O_2, where $O_2 \subset O_1$, by considering refusal information, we can distinguish between the case where the process chooses to output O_2 and the case where the environment is only willing to perform the events from O_2 from the process' offer of events O_1.

Next we must change our representation of undesirable behaviour from undefined behaviour to infeasible behaviour. For each maximal behaviour s, we must extend the trace $\gamma_O^*(s)$ by infeasible behaviour and, by closure condition vii, by arbitrary behaviour. The set of such extensions is given by:

$$T_3 = \{\gamma_O^*(s)^\frown r \mid s \in \widehat{T} \wedge r \in \mathbb{F}(\tilde{A})^*\}$$

Finally we close all the traces we have thus far obtained under removal of refusal information, we can remove any amount of refusal information without affecting the behaviour of the process. This ensures the set, T', is closed under condition v, where

$$T' = \{s \mid \exists r \in \mathbb{F}(\bar{O})^* \bullet s \cup r \in T_1 \cup T_2 \cup T_3\}$$

Now $T' = \phi(I, O, T)$. By considering the traces in $\phi(I, O, T)$ generated by each trace in T we obtain the sets $\psi(I, O, s)$ or $\psi_1(I, O, s)$ depending on whether $s \in T - \widehat{T}$ or $s \in \widehat{T}$. Thus we obtain the above definition of Φ.

6.1.1 Properties of Φ

We verify that Φ is well-defined and prove that it is monotonic, continuous and injective over a restricted domain.

Lemma 6.1 *For $s \in RT_{I,O}$ and $A = I \cup O$*

$$\psi_I(s) - \psi(s) = \{s' \mid \exists\, r \in \mathbb{F}(\overline{O})^* \bullet \gamma^*_O(s) < s' \cup r\}$$

■

Theorem 6.2 *Φ is well-defined.*

Proof: We must show that if $(I, O, T) \in RM$ then $(I \cup O, \phi(I, O, T))$ is a process in SM. Assuming T satisfies closure conditions I–III with respect to input and output alphabets I and O, it is sufficient to show that $\phi(I, O, T)$ satisfies closure conditions i–vii with respect to alphabet $I \cup O$.

Since I and O remain unchanged throughout this proof we will abuse notation and write $\phi(T)$ for $\phi(I, O, T)$, $\psi(s)$ for $\psi(I, O, s)$ and γ for γ_O.

The pattern of proof being similar for each closure condition, only that for condition vi is presented here.

Assume that $s^\frown\langle B\rangle^\frown s' \in \phi(T)$ and $C = \{a \in I \cup O \mid a \notin B \wedge \bar{a} \notin B\}$, we want to show that either $s^\frown\langle B \cup C\rangle \in \phi(T)$ or $s^\frown\langle B \cup \{\bar{x}\}\rangle^\frown s' \in \phi(T)$ for some $x \in C$.

$$
\begin{array}{ll}
 & s^\frown\langle B\rangle^\frown s' \in \phi(T) \\
\Rightarrow & \{\text{ by the definition of } \phi\ \} \\
 & (\exists\, u \in T \bullet s^\frown\langle B\rangle^\frown s' \in \psi(u)) \vee (\exists\, u \in \widehat{T} \bullet s^\frown\langle B\rangle^\frown s' \in \psi_I(u) - \psi(u)) \\
\Rightarrow & \{\text{ recalling defiuition of } \psi \text{ and } \psi_I\ \} \\
 & \exists\, u \in T\,;\, r^\frown\langle R\rangle^\frown r' \in \mathbb{F}(\overline{O})^* \bullet (s \cup r)^\frown\langle B \cup R\rangle^\frown(s' \cup r') = \gamma^*(u) \qquad (1) \\
 & \vee\ \exists\, u^\frown\langle X\rangle \in T\,;\, r^\frown\langle R\rangle^\frown r' \in \mathbb{F}(\overline{O})^*\,;\, Y \subseteq X \cap O \bullet (Y \neq \{\} \wedge \qquad (2) \\
 & \qquad \gamma^*(u)^\frown\langle\gamma(X) - Y\rangle \leq (s \cup r)^\frown\langle B \cup R\rangle^\frown(s' \cup r')) \\
 & \vee\ \exists\, u \in \widehat{T}\,;\, r^\frown\langle R\rangle^\frown r' \in \mathbb{F}(\overline{O})^* \bullet \gamma^*(u) < (s \cup r)^\frown\langle B \cup R\rangle^\frown(s' \cup r') \qquad (3)
\end{array}
$$

We now proceed by case analysis on the form of traces in $\phi(T)$, so we shall consider $s^\frown\langle B\rangle^\frown s'$ satisfying each of the above disjuncts in turn.

Case 1. If $R - B \neq \{\}$ then, as all elements in the trace $\gamma^*(u)$ are feasible, we can choose $\bar{x} \in R \cap \overline{C}$ and $s^\frown\langle B \cup \{\bar{x}\}\rangle^\frown s' \in \phi(T)$ as required.

If $R \cup B = B$, then $(s \cup r)^\frown\langle B\rangle \leq \gamma^*(u)$ so B must be saturated with respect to output as every element in the trace $\gamma^*(u)$ is. Hence $C \subseteq I$. Then by the closure conditions on T and the nature of γ we can find $u' \in T$ with $\gamma^*(u') = (s \cup r)^\frown\langle B \cup C\rangle$. Thus $s^\frown\langle B \cup C\rangle \in \phi(T)$ as required.

Case 2. If $s \cup r \geq \gamma^*(u)^\frown\langle\gamma(X) - Y\rangle$ then $(s \cup r)^\frown\langle B \cup C\rangle \in \psi(s')$ and we are done.

If $(s \cup r)^\frown\langle B \cup R\rangle \leq \gamma^*(u)^\frown\langle\gamma(X) - Y\rangle$, then when $R - B \neq \{\}$ or the inequality is strict the result follows as before. In the remaining case $\gamma(X) - Y = B$. Now $\gamma(X)$ is feasible and saturated with respect to output, and $Y \subseteq O$ so $Y = C \cap O$. Also $u^\frown\langle X \cup (C \cap I)\rangle \in T$ by condition III on T and $(s \cup r)^\frown\langle B \cup C\rangle = \gamma^*(u^\frown\langle X \cup (C \cap I)\rangle)$. Hence $s^\frown\langle B \cup C\rangle \in \phi(T)$ as required.

Case 3. If $s \cup r \geq \gamma^*(u)$ then clearly $s^\frown\langle B \cup C\rangle \in \psi_I(u)$ and we have the required result. Otherwise $(s \cup r)^\frown\langle B \cup R\rangle \leq \gamma^*(u)$ and the result follows as before.

□

Lemma 6.3 *If* $s, s' \in \mathbb{F}(I \cup O)^*$, *then*

a. $\psi(I, O, s) \subseteq \psi_I(I, O, s)$

b. $s \leq s' \Rightarrow \psi_I(I, O, s') \subseteq \psi_I(I, O, s)$

Proof:

a. Trivial by definition of ψ_I.

b. If $s' = s$ then the result is trivial, so suppose $s' > s$, clearly $s' \neq \langle\rangle$ so we can choose s'' and X such that $s' = s''^\frown\langle X\rangle$.

By the definition of ψ_I we have $r \in \psi_I(I, O, s') \Rightarrow \exists\, r_0 \in \psi(I, O, s') \bullet r_0 \leq r$

Now

$$
\begin{array}{ll}
 & r_0 \in \psi(I, O, s''^\frown\langle X\rangle) \\
\Rightarrow & \{\ \text{by the definition of } \psi\ \} \\
 & \exists\, u_0 \in \mathbb{F}(\bar{O})^* \bullet (r_0 \cup u_0 = \gamma^*(s''^\frown\langle X\rangle) \\
 & \qquad\qquad \vee\ \exists\, Y \subseteq X \cap O \bullet r_0 \cup u_0 \geq \gamma^*(s'')^\frown\langle\gamma(X) - Y\rangle) \\
\Rightarrow & \{\ \text{as } \gamma^*(s) \leq \gamma^*(s'')\ \} \\
 & \exists\, u_0 \in \mathbb{F}(\bar{O})^* \bullet r_0 \cup u_0 > \gamma^*(s) \\
\Rightarrow & \{\ \text{taking appropriate subsequences of } r_0 \text{ and } u_0\ \} \\
 & \exists\, u_1 \in \mathbb{F}(\bar{O})^*; r_1 \in \mathbb{F}(\check{I} \cup \check{O})^* \bullet r_1 \cup u_1 > \gamma^*(s) \wedge r_1 < r_0 \\
\Rightarrow & \{\ \text{by the definition of } \psi\ \} \\
 & \exists\, r_1 \in \psi(I, O, s) \bullet r_1 < r_0
\end{array}
$$

Thus, as $r_1 < r_0 \leq r$ we have that $r \in \psi_I(I, O, s)$ □

Theorem 6.4 Φ *is a monotonic function from* $(RM, \leqslant)$ *to* $(SM, \sqsubseteq)$. *That is,*

$$\forall P, Q \in RM \bullet P \leqslant Q \Rightarrow \Phi(P) \sqsubseteq \Phi(Q)$$

Proof: Suppose $(I, O, T(P))$ and $(I, O, T(Q))$ represent P and Q resectively. Processes in SM are only ordered if they have the same alphabets. For P and Q to be ordered in the information ordering they must have the same input and output alphabets, thus $\Phi(P)$ and $\Phi(Q)$ have the same alphabets. It remains to verify that, if the traces sets of P and Q satisfy $T(Q) \downarrow \widehat{T}(P) = T(P)$ then $\phi(I, O, T(Q)) \subseteq \phi(I, O, T(P))$.

$$
\begin{array}{ll}
s \in \widehat{T}(Q) \Rightarrow & \{ \text{ since } T(Q) \downarrow \widehat{T}(P) = T(P) \ \} \\
& \exists\, s' \in \widehat{T}(P) \bullet s \geq s' \\
\Rightarrow & \{ \text{ by Lemma 6.3 } \} \\
& \exists\, s' \in \widehat{T}(P) \bullet \psi_I(I, O, s) \subseteq \psi_I(I, O, s') \\
\Rightarrow & \{ \text{ by definition of } \phi \ \} \\
& \psi_I(I, O, s) \subseteq \phi(I, O, T(P)) \\
\\
s \in T(Q) \Rightarrow & \{ \text{ since } T(Q) \downarrow \widehat{T}(P) = T(P) \ \} \\
& s \in T(P) \vee \exists\, s' \in \widehat{T}(P) \bullet s \geq s' \\
\Rightarrow & \{ \text{ by Lemma 6.3 } \} \\
& s \in T(P) \vee \exists\, s' \in \widehat{T}(P) \bullet \psi(I, O, s) \subseteq \psi_I(I, O, s') \\
\Rightarrow & \{ \text{ by definition of } \phi \ \} \\
& \psi(I, O, s) \subseteq \phi(I, O, T(P))
\end{array}
$$

Hence, recalling the definition of ϕ, we have $\phi(I, O, T(Q)) \subseteq \phi(I, O, T(P))$ and the result follows. □

Theorem 6.5 *Φ is continuous. That is, for D a $\leqslant$-directed set.*

$$\Phi(\sqcup_{\leqslant} D) = \sqcup\{\Phi(P) \mid P \in D\}$$

Proof: Suppose (I, O, T_P) represents process P. Since the alphabets remain unchanged throughout this proof, we shall abuse notation slightly, writing $\psi(s)$ for $\psi(I, O, s)$.

By monotonicity of Φ, $\phi(\sqcup_{\leqslant} D) \subseteq \bigcap\{\phi(P) \mid P \in D\}$.

It remains to show that $\bigcap\{\phi(P) \mid P \in D\} \subseteq \phi(\sqcup_{\leqslant} D)$. We do this by contradiction.

Recall

$$\bigcap\{\phi(P) \mid P \in D\} = \bigcap_{P \in D} \left(\bigcup_{s \in T_P} \psi(s) \cup \bigcup_{s \in \widehat{T}_P} \psi_I(s) \right)$$

and $$\phi(\sqcup_{\leqslant} D) = \bigcup_{s \in T_{\sqcup_{\leqslant} D}} \psi(s) \cup \bigcup_{s \in \widehat{T}_{\sqcup_{\leqslant} D}} \psi_I(s)$$

Suppose $s \in \bigcap\{\phi(P) \mid P \in D\}$ and $s \notin \phi(\sqcup_{\leqslant} D)$.

From the former we deduce that

$$\forall P \in D \bullet (\exists s' \in T_P \bullet s \in \psi(s')) \vee (\exists s' \in \widehat{T}_P \bullet s \in \psi_1(s'))$$

while from $s \notin \phi(\sqcup_{\leqslant} D)$ we deduce that

$$\forall P \in D \bullet \neg (\exists s' \in T_P \bullet s \in \psi(s')).$$

Combining these we have

$$\forall P \in D \bullet (\exists s' \in \widehat{T}_P \bullet s \in \psi_1(s'))$$

Now consider the set $B \mathrel{\widehat{=}} \{s' \mid \exists P \in D \bullet s' \in \widehat{T}_P \wedge s \in \psi_1(s')\}$

By construction B is finite and non-empty and $\forall P \in D \bullet \exists r \in B \bullet r \in \widehat{T}_P$

We shall show by contradiction that $\exists s' \in B \bullet s' \in \widehat{T}_{\sqcup_{\leqslant} D}$

Suppose not, then by Lemma 5.8 $\forall s' \in B \bullet \exists P \in D \bullet \neg (\exists s_0 \in \widehat{T}_P \bullet s_0 \leq s')$.

For $r \in B$ choose P_r such that $r \in \widehat{T}_{P_r}$ and P'_r such that $\neg (\exists r_0 \in \widehat{T}_{P'_r} \bullet r_0 \leq r)$

By the directed set property, choose $Q \in D$ such that $\forall r \in B \bullet P_r \leqslant Q \wedge P'_r \leqslant Q$

$$T_Q \downarrow \widehat{T}_{P_r} = T_{P_r} \Rightarrow r \in T_Q$$
$$T_Q \downarrow \widehat{T}_{P'_r} = T_{P'_r} \Rightarrow r \notin \widehat{T}_Q$$

Thus $\neg (\exists r \in B \bullet r \in \widehat{T}_Q)$ contrary to our construction of B. So

$$\begin{aligned} \exists s' \in B \bullet s' \in \widehat{T}_{\sqcup_{\leqslant} D} \Rightarrow & \quad \{ \text{ by definition of } B \ \} \\ & \exists s' \bullet s \in \psi_1(s') \wedge s' \in \widehat{T}_{\sqcup_{\leqslant} D} \\ \Rightarrow & \quad \{ \text{ by definition of } \phi \text{ and } \sqcup_{\leqslant} D \ \} \\ & s \in \phi(\sqcup_{\leqslant} D) \end{aligned}$$

Heuce result by contradiction. □

Theorem 6.6 *Given input and output alphabets I and O, the restriction of Φ to $RM^{I,O}$ is injective. That is*

$$\forall P, Q \in RM^{I,O} \bullet \Phi(P) = \Phi(Q) \Rightarrow P = Q.$$

Proof: Suppose that (I, O, T_Q) and (I, O, T_P) represent processes P and Q respectively. As in earlier proofs we shall abuse notation since the alphabets do not vary throughout the proof; we write $\psi(s)$ for $\psi(I, O, s)$ and γ for γ_O. We must show that whenever the trace sets of $\Phi(P)$ and $\Phi(Q)$ agree, so to do the trace sets of P and Q.

$$\phi(P) = \phi(Q) \Rightarrow T_Q = T_P$$

Suppose not, so $\phi(P) = \phi(Q)$ and $T_P \neq T_Q$. Without loss of generality, we can choose $s \in T_P$ such that $s \notin T_Q$.

$s \in T_P$

$\Rightarrow$ { by definition of ϕ }

$\gamma^*(s) \in \phi(P)$

$\Rightarrow$ { by assumption }

$\gamma^*(s) \in \phi(Q)$

$\Rightarrow$ { by definition of ϕ }

$(\exists s' \in T_Q \bullet \gamma^*(s) \in \psi(s')) \vee (\exists s' \in \widehat{T}_Q \bullet \gamma^*(s) \in \psi_I(s') - \psi(s'))$

$\Rightarrow$ { by definition of ψ and noting $\gamma^*(s)$ is saturated w.r.t. output }

$(\exists s' \in T_Q \bullet \gamma^*(s) = \gamma^*(s')) \vee (\exists s' \in \widehat{T}_Q \bullet \gamma^*(s) \in \psi_I(s') - \psi(s'))$

$\Rightarrow$ { since γ is injective and $s \notin T_Q$ }

$\exists s' \in \widehat{T}_Q \bullet \gamma^*(s) \in \psi_I(s') - \psi(s')$

$\Rightarrow$ { by definition of $\psi_I(s') - \psi(s')$ }

$\exists s' \in \widehat{T}_Q \bullet \gamma^*(s) > \gamma^*(s')$

$\Rightarrow$ { γ is injective on $\mathbb{F}(I \cup O)$ }

$\exists s' \in \widehat{T}_Q \bullet s > s'$

Now $\gamma^*(s')^\frown\langle \check{O} \cup \check{I} \rangle \in \phi(Q)$ since $s' \in \widehat{T}_Q$.

Thus $\gamma^*(s')^\frown\langle \check{O} \cup \check{I} \rangle \in \phi(P)$ by assumption. This trace is saturated with respect to output with an infeasible end so $\gamma^*(s')^\frown\langle \check{O} \cup \check{I} \rangle \in \psi_I(s_0) - \psi(s_0)$ where $s_0 \in \widehat{T}_P$. Thus $\gamma^*(s') \geq \gamma^*(s_0)$, giving $s' \geq s_0$

So $s_0 \in \widehat{T}_P$ and $s \in T_P$ with $s > s_0$ contrary to the definition of $\widehat{T}_P$. □

6.2 Relating the Languages SRPT and SCSP

We have embedded the receptive model *RM* into *SM* in a manner which, to a certain extent, relates processes in *RM* to those which can be seen to represent the same system in *SM*. In this section we define a function $\Theta : SRPT \rightarrow SCSP$ over the syntax of SRPT which maps terms of SRPT to terms of SCSP. Like the function Φ, Θ is chosen so that the process expression $P \in SRPT$ and its image $\Theta(P) \in SCSP$ can be regarded as describing the same system in the two different languages.

Definition 6.2 We define the function $\Theta : SRPT \to SCSP$ over the syntax of SRPT as follows:

$$\begin{aligned}
\Theta(\bot_{I,O}) &= \bot_{I \cup O} \\
\Theta(x) &= x \\
\Theta(P \sqcap Q) &= (\Theta P) \sqcap (\Theta Q) \\
\Theta([!B?X \to P_X]) &= [Y \subseteq (B \cup I) \to ((\Theta P_{Y-B}) \textbf{ if } B \subseteq Y \textbf{ else } \bot_{I \cup O})] \\
&\quad \text{where } I \text{ and } O \text{ are input and output alphabets} \\
&\quad \text{of } [!B?X \to P_X] \\
\Theta(P \parallel Q) &= (\Theta P) \parallel (\Theta Q) \\
\Theta(P \setminus A) &= (\Theta P) \setminus A \\
\Theta(P[S]) &= (\Theta P)[S] \\
\Theta(\mu x : I, O \bullet P) &= \mu x : I \cup O \bullet (\Theta P)
\end{aligned}$$

If P has input alphabet I and output alphabet O then ΘP has alphabet $I \cup O$. $\Diamond$

6.3 Deducing results of SRPT from SCSP

We shall demonstrate that the function Θ is closely related to Φ in the sense that, if we define η to be the projection of $BIND_R$ onto $BIND$ induced by Φ then the following holds for all processes $P \in SRPT$ and all variable bindings $\sigma \in BIND_R$.

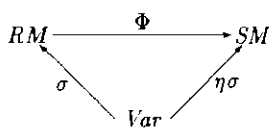
$$\mathcal{M}[\![\Theta P]\!]\eta\sigma = \Phi\mathcal{M}_R[\![P]\!]\sigma$$

This relationship allows us to deduce the continuity of the operators of SRPT. It also enables us to deduce the soundness and completeness of the proof system for SRPT from corresponding results in SCSP.

Definition 6.3 We define $\eta : BIND_R \to BIND$ to be the unique function induced by Φ which maps $BIND_R$ into $BIND$. Formally:

$$\forall \sigma \in BIND_R ; x \in Var \bullet (\eta\sigma)[\![x]\!] \mathrel{\hat{=}} \Phi(\sigma[\![x]\!])$$

Informally the following commutes for all $\sigma \in BIND_R$:



$\Diamond$

Lemma 6.7 *For all processes $P \in SRPT^0$ and all variable bindings $\sigma \in BIND_R$*

$$\mathcal{M}[\![\Theta P]\!]\eta\sigma = \Phi\mathcal{M}_R[\![P]\!]\sigma$$

Proof: By structural induction over the syntax of SRPT$^\theta$

atomic terms

$$
\begin{array}{lll}
\text{a)}\ \mathcal{M}[\![\Theta\ \bot_{I,O}\ ]\!]\eta\sigma & = \mathcal{M}[\![\ \bot_{I\cup O}\ ]\!]\eta\sigma & \{\text{ definition of } \Theta\ \} \\
& = (I \cup O, ST_{I\cup O}) & \{\text{ definition of } \mathcal{M}\ \} \\
& = (I \cup O, \phi(I, O, \{\langle\rangle\})) & \{\text{ definition of } \phi\ \} \\
& = (I \cup O, \phi(\mathcal{M}_{\mathcal{R}}[\![\ \bot_{I,O}\ ]\!]\sigma)) & \{\text{ definition of } \mathcal{M}_{\mathcal{R}}\ \} \\
& = \Phi(\mathcal{M}_{\mathcal{R}}[\![\ \bot_{I,O}\ ]\!]\sigma) & \{\text{ definition of } \Phi\ \}
\end{array}
$$

$$
\begin{array}{lll}
\text{b)}\ \mathcal{M}[\![\Theta x]\!]\eta\sigma & = \mathcal{M}[\![x]\!]\eta\sigma & \{\text{ definition of } \Theta\ \} \\
& = (\eta\sigma)[\![x]\!] & \{\text{ definition of } \mathcal{M}\ \} \\
& = \Phi(\sigma[\![x]\!]) & \{\text{ definition of } \eta\ \} \\
& = \Phi(\mathcal{M}_{\mathcal{R}}[\![x]\!]\sigma) & \{\text{ definition of } \mathcal{M}_{\mathcal{R}}\ \}
\end{array}
$$

operators That the alphabets are the same is a trivial observation. Our task is to verify that the traces sets of the expressions on the left and right sides of the equation correspond. That is

$$T[\![\Theta P]\!]\eta\sigma = \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma)$$

We assume that all arguments of an operator satisfy the above equation and deduce that the application of the operator to these arguments does also. The proof for non-deterministic choice is presented as Theorem A.4 in Appendix A.3. The proofs are in general long and not particularly enlightening, involving examination of the terms on both sides of the equation. □

Lemma 6.8 *If* $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ *is monotonic in* $(RM, \leqslant)$,
$\forall \sigma \in BIND_R \bullet \mathcal{M}[\![\Theta P]\!]\eta\sigma = \Phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma)$
and $\forall \rho \in BIND \bullet \lambda\, y \bullet \mathcal{M}[\![\Theta P]\!]\rho[y/x]$ *is continuous in* $(SM, \sqsubseteq)$
then

a) $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ *is continuous in* $(RM, \leqslant)$.

b) $\mathcal{M}[\![\Theta(\mu\, x : I, O \bullet P)]\!]\eta\sigma = \Phi(\mathcal{M}_{\mathcal{R}}[\![(\mu\, x : I, O \bullet P)]\!]\sigma)$.

Proof: **a)** Suppose D is directed in $(RM, \leqslant)$, we must show that

$$\bigsqcup_{\substack{\leqslant \\ q\in D}} \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q/x] = \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[\cup_{\leqslant} D/x]$$

as $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is monotone $\bigsqcup_{\substack{\leqslant \\ q\in D}} \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q/x]$ is well defined.

Now

$$
\begin{array}{ll}
\Phi(\bigsqcup_{q\in D}^{\leq} \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q/x]) & \\
\quad = \bigsqcup_{q\in D} \Phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[q/x]) & \{\text{ as } \Phi \text{ is continuous }\} \\
\quad = \bigsqcup_{q\in D} (\mathcal{M}[\![\Theta P]\!]\eta(\sigma[q/x])) & \{\text{ by hypothesis }\} \\
\quad = \bigsqcup_{q\in D} (\mathcal{M}[\![\Theta P]\!](\eta\sigma)[(\Phi q)/x]) & \{\text{ by definition of } \eta \text{ }\} \\
\quad = \mathcal{M}[\![\Theta P]\!](\eta\sigma)[(\bigsqcup_{q\in D} \Phi q)/x]) & \{\text{ as } \lambda\, y \bullet \mathcal{M}[\![\Theta P]\!]\rho[y/x] \text{ is continuous }\} \\
\quad = \mathcal{M}[\![\Theta P]\!](\eta\sigma)[(\Phi(\bigsqcup_{q\in D}^{\leq} q))/x]) & \{\text{ as } \Phi \text{ is continuous }\} \\
\quad = \mathcal{M}[\![\Theta P]\!](\eta(\sigma[(\bigsqcup_{q\in D}^{\leq} q)/x])) & \{\text{ by definition of } \eta \text{ }\} \\
\quad = \Phi(\mathcal{M}[\![P]\!]\sigma[(\bigsqcup_{q\in D}^{\leq} q)/x]) & \{\text{ by hypothesis }\}
\end{array}
$$

The result follows as Φ is injective when restricted to processes with the same alphabets.

b) We set $h \mathrel{\hat{=}} \lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$

$$H \mathrel{\hat{=}} \lambda\, y \bullet \mathcal{M}[\![\Theta P]\!](\eta\sigma)[y/x]$$

It can be shown that $\forall\, q \in RM \bullet \Phi h(q) = H(\Phi q)$

Now

$$
\begin{array}{ll}
\mathcal{M}[\![\Theta(\mu\, x : I, O \bullet P)]\!]\eta\sigma & \\
\quad = \mathcal{M}[\![\mu\, x : I \cup O \bullet (\Theta P)]\!]\eta\sigma & \{\text{ by definition of } \Theta \text{ }\} \\
\quad = \underline{\mathrm{fix}}_{I\cup O}\, \lambda\, y \bullet \mathcal{M}[\![\Theta P]\!](\eta\sigma)[y/x] & \{\text{ definition of recursion }\} \\
\quad = \bigsqcup_{n=0}^{\infty} H^n(\mathcal{M}[\![\ \perp_{I\cup O}\ ]\!]) & \{\text{ by definition and continuity of } H \text{ }\} \\
\quad = \bigsqcup_{n=0}^{\infty} H^n(\mathcal{M}[\![\Theta(\perp_{I,O})]\!]) & \{\text{ by definition of } \Theta \text{ }\} \\
\quad = \bigsqcup_{n=0}^{\infty} H^n(\Phi(\mathcal{M}_{\mathcal{R}}[\![\ \perp_{I,O}\ ]\!])) & \{\text{ by Lemma 6.7 }\} \\
\quad = \bigsqcup_{n=0}^{\infty} \Phi(h^n(\mathcal{M}_{\mathcal{R}}[\![\ \perp_{I,O}\ ]\!])) & \{\text{ by above observation }\} \\
\quad = \Phi(\bigsqcup_{n=0}^{\infty} h^n(\mathcal{M}_{\mathcal{R}}[\![\ \perp_{I,O}\ ]\!])) & \{\text{ as } \Phi \text{ is continuous }\} \\
\quad = \Phi(\underline{\mathrm{fix}}_{I,O}\, \lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]) & \{\text{ by definition and continuity of } h \text{ }\} \\
\quad = \Phi(\mathcal{M}_{\mathcal{R}}[\![\mu\ : I, O \bullet P]\!]\sigma) & \{\text{ by definition of recursion }\}
\end{array}
$$

□

6.3.1 Continuity

Now we have established a sufficiently strong link between SRPT and SCSP and the corresponding models to deduce the continuity result required in Section 5.4.

Theorem 6.9 *For all $P \in SRPT$, $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is continuous.*

Proof: After Lemma 5.12 it is sufficient to consider only $P \in \mathrm{SRPT}^0$. If $P \in \mathrm{SRPT}^0$ then $\Theta P \in \mathrm{SCSP}^0$. So by combining the results of Theorem 5.11,

Lemma 3.5 and Lemma 6.7 with Lemma 6.8(a) we have the required result. □

Theorem 6.10 *For all* $P \in SRPT$, $\sigma \in BIND_R$

$$\mathcal{M}[\![\Theta P]\!]\eta\sigma = \Phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma)$$

Proof: This follows by structural induction over the syntax of SRPT, all but the case of recursion are covered in Lemma 6.7. The final case follows from Theorem 6.9 and Lemma 6.8(b). □

6.3.2 A proof system for SRPT

As for SCSP we introduce a proof system for the sublanguage SRPT^l of SRPT cousisting of the non-recursive closed terms of SRPT, we then extend this to allow recursive terms. The logical language cousists of assertions of the form $P \sqsubseteq_R Q$ and $P \equiv_R Q$. We give a set of axioms and inference rules for proving assertions, and show that the system is both sound and complete by relating it to the proof system for SCSP.

The sublanguage SRPT^l

The syntax of SRPT^l is given by:

$$P ::= \perp_{I,O} \mid P \sqcap P \mid [!B?X \rightarrow P_X] \mid P \parallel P \mid P \setminus A \mid P[S]$$

Formulae in the logical language take the form $P \sqsubseteq_R Q$ or $P \equiv_R Q$, where these relations are defined as follows.

Definition 6.4 For process expressions $P, Q \in SRPT$ we say that P is less deterministic that Q, written $P \sqsubseteq_R Q$, if for every possible variable binding the semantics of $P \sqcap Q$ cannot be distinguished from the semantics of P in RM. Formally:

$$P \sqsubseteq_R Q \triangleq \forall \sigma \in BIND_R \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma.$$

recalling that $\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma \Leftrightarrow \mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!] = \mathcal{M}_{\mathcal{R}}[\![P]\!]$

We say that P and Q are equivalent, written $P \equiv_R Q$, if for every possible variable binding P and Q have the same semantics in RM.

$$P \equiv_R Q \triangleq \forall \sigma \in BIND_R \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma = \mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma.$$

◊

In situations where there can be no confusion as to the language being referred to we shall often drop the R suffix from the above relations and simply write $P \sqsubseteq Q$ and $P \equiv Q$.

Notice that the ordering used in the language is the non-determinism ordering. This ordering is more appropriate to the problem of deriving implementations from specifications than the information ordering. In many circumstances we require an implementation to be completely deterministic, such processes are the maximal processes with respect to the non-determinism ordering. On the other hand, specifications often exhibit an element of non-determinism. By encorporating the non-determinism ordering in our language we can refine a non-deterministic specification to a deterministic implementation within our proof system.

Had we used the information ordering we would only be able to find an implementation to a specification which diverged less; the information ordering does not relate non-divergent processes.

The axioms of the system are given in Appendix B.4. The axioms are very similar to the axioms for SCSP^I, the noticeable difference being the equations concerning the prefix construct. We write $\vdash_R \ P \equiv_R Q$ to assert that $P \equiv_R Q$ is provable in the axiom system for SRPT^I.

Notes

1. Notice that as for the proof system for SCSP^I the following results can be derived from the axiom system.

   o–5 $\vdash \ P \equiv_R P$ **o–6** $\vdash \ \bot \sqsubseteq_R P$

   o–7 $\dfrac{P \sqcap Q \equiv_R P}{P \sqsubseteq_R Q}$ **o–8** $\dfrac{P \sqsubseteq_R Q}{P \sqcap Q \equiv_R P}$

   These results clearly demonstrate the link between the ordering $\sqsubseteq_R$ and the non-deterministic choice operator.

2. The strong link between this axiom system and that of SCSP^I is demonstrated by noting that each of the axioms A–1 to A–14 preserves receptiveness. That is, if $\Theta P \in$ SCSP^I and by one of the axioms A–1 to A–14 we can deduce:

   $$\vdash \ \Theta P \equiv Q'$$

   then Q' is the image under Θ of some $Q \in$ SRPT^I. Moreover, $P \equiv_R Q$ is deducible in the proof system for SRPT^I. In many cases this is clear by the definition of Θ. By demonstrating this link between A–5 and a–5 it becomes apparent that the axioms concerning the prefix construct are closely related.

Suppose $C \subseteq B$, then

$$
\begin{array}{ll}
 & \Theta([!B?X \to P_X] \sqcap [!C?Y \to Q_Y]) \\
= & \quad \{ \text{ by definition of } \Theta \ \} \\
 & [X \subseteq B \cup I \to (\Theta P_{X-B} \textbf{ if } B \subseteq X \textbf{ else } \bot_{I \cup O})] \\
 & \quad \sqcap [Y \subseteq C \cup I \to (\Theta Q_{Y-C} \textbf{ if } C \subseteq Y \textbf{ else } \bot_{I \cup O})] \\
\equiv & \quad \{ \text{ by A-5 } \} \\
 & [X \subseteq B \cup I \to P^{\emptyset}_X] \\
 & \quad \sqcap [Y \subseteq C \cup I \to (\Theta Q_{Y-C} \textbf{ if } C \subseteq Y \textbf{ else } \bot_{I \cup O})]
\end{array}
$$

where for $B' \subseteq B \cup I$:

$$
P^{\emptyset}_{B'} = \begin{cases} (\Theta P_{B'-B} \textbf{ if } B \subseteq B' \textbf{ else } \bot_{I \cup O}) & \text{if } B' \not\subseteq C \cup I \\ ((\Theta P_{B'-B} \sqcap \Theta Q_{B'-C}) \textbf{ if } B \cup C \subseteq B' \textbf{ else } \bot_{I \cup O}) & \text{if } B' \subseteq C \cup I \end{cases}
$$

Consider the case when $C = B$ and $C \subset B$ separately.

If $C \subset B$ then $B' \subseteq C \cup I \Rightarrow B \not\subseteq B'$ hence

$$P^{\emptyset}_{B'} = (\Theta P_{B'-B} \textbf{ if } B \subseteq B' \textbf{ else } \bot_{I \cup O})$$

If $C = B$ then

$$P^{\emptyset}_{B'} = (\Theta(P_{B'-B} \sqcap Q_{B'-C}) \textbf{ if } B \subseteq B' \textbf{ else } \bot_{I \cup O})$$

So if $C \subset B$ then A-5 reduces to a trivial equality for receptive processes. If $C = B$ then

$$
\begin{array}{ll}
\vdash & \Theta([!B?X \to P_X] \sqcap [!C?Y \to Q_Y]) \\
 & \quad \equiv \Theta([!B?X \to P_X \sqcap Q_X] \sqcap [!C?Y \to Q_Y])
\end{array}
$$

but $[!B?X \to P_X] \sqcap [!C?Y \to Q_Y] \equiv_R [!B?X \to P_X \sqcap Q_X] \sqcap [!C?Y \to Q_Y]$ is provable as follows:

$$
\begin{array}{l}
[!B?X \to P_X] \sqcap [!B?Y \to Q_Y] \\
\quad \{ \text{ a-3 } \} \equiv_R \ [!B?X \to P_X] \sqcap ([!B?Y \to Q] \sqcap [!B?Y \to Q_Y]) \\
\quad \{ \text{ a-2 } \} \equiv_R \ ([!B?X \to P_X] \sqcap [!B?Y \to Q]) \sqcap [!B?Y \to Q_Y] \\
\quad \{ \text{ a-5 } \} \equiv_R \ [!B?X \to P_X \sqcap Q_X] \sqcap [!B?Y \to Q_Y]
\end{array}
$$

Soundness

If every assertion, provable in the proof system is true, then the system is sound. We shall deduce the soundness of the proof system for SRPT^I by considering the image of the rules under the map Θ. If $P \sqsubseteq_R Q$ and $P \equiv_R Q$ are assertions in the logical langnage for SRPT^I then we take their images under Θ to be $\Theta P \sqsubseteq \Theta Q$ and $\Theta P \equiv \Theta Q$ respectively. We note that the images are assertions in the logical language for SCSP^I.

By the nature of Θ, the image of each axiom of the proof system of SRPT^I under Θ is a provable assertion of SCSP^I. It follows that:

Lemma 6.11 *For all* $P, Q \in SRPT^I$

$$(\vdash_R \ P \sqsubseteq_R Q) \Rightarrow (\vdash \ \Theta P \sqsubseteq \Theta Q)$$

■

From this result and the soundness of the proof system for SCSP^I we deduce:

Theorem 6.12 (Soundness) *For all* $P, Q \in SRPT^I$

$$(\vdash_R \ P \sqsubseteq_R Q) \Rightarrow (\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!])$$

Proof:

$$\begin{array}{lll}
\vdash_R \ P \sqsubseteq_R Q \Rightarrow \ \vdash \ \Theta P \sqsubseteq \Theta Q & \{ \text{ by Lemma 6.11 } \} \\
\Rightarrow \ \mathcal{M}[\![\Theta P]\!] \sqsubseteq \mathcal{M}[\![\Theta Q]\!] & \{ \text{ soundness of SCSP}^I \} \\
\Rightarrow \ \mathcal{M}[\![(\Theta P) \sqcap (\Theta Q)]\!] = \mathcal{M}[\![\Theta P]\!] & \{ \text{ equivalent formulation } \} \\
\Rightarrow \ \mathcal{M}[\![\Theta(P \sqcap Q)]\!] = \mathcal{M}[\![\Theta P]\!] & \{ \text{ definition of } \Theta \ \} \\
\Rightarrow \ \Phi(\mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!]) = \Phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]) & \{ \text{ by Theorem 6.7 } \} \\
\Rightarrow \ \mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!] = \mathcal{M}_{\mathcal{R}}[\![P]\!] & \{ \ \Phi \text{ injective } \}
\end{array}$$

□

Completeness

In order to establish completeness of the system we must show that every true assertion is provable. We mnst show that whenever $\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq \mathcal{M}_{\mathcal{R}}[\![Q]\!]$ then the formula $P \sqsubseteq_R Q$ is provable. We shall define a class of normal forms and show that every term is provably equivalent to a unique normal form. Finally, by using the map Θ we can relate these normal forms to those of SCSP. Through this we deduce that the system consisting of the class of normal form processes is complete.

Normal form

The normal forms of SRPT^I have a similar structure to those of SCSP^I. $\bot_{I,O}$ is a normal form, all other normal forms are the non-deterministic choice between a finite number of output prefixed processes. The output sets in the prefix constructs are unique, ensuring normal forms are unique.

Definition 6.5 We say a process, $P \in \mathrm{SCSP}^I$, with alphabets I and O is in normal form if it is $\bot_{I,O}$ or takes the form:

$$P = \prod_{B \in \mathcal{B}} [!B?X \rightarrow P_{B,X}]$$

where

- $\mathcal{B}$ is a non-empty finite subset of $\mathbb{P}\, O$.
- $P_{B,X}$ is in normal form for all $B \in \mathcal{B}$ and $X \subseteq I$.

$\Diamond$

The proof that every process in SRPT^I is provably equivalent to a process in normal from follows the form of the equivalent proof in SCSP^I, so is not presented here.

Lemma 6.13 *Every process in $SRPT^I$ is provably equivalent to a process in normal form.* ■

Lemma 6.14 *If $P \in SRPT^I$ is in normal form with alphabets I and O, then $\Theta P \in SCSP^I$ is in normal form.*

Proof: By structural induction on P.

base case: $P = \bot_{I,O}$. Then $\Theta P = \bot_{I \cup O}$ is in normal form.

inductive step:

$$P = \prod_{B \in \mathcal{B}} [!B?X \rightarrow P_{B,X}].$$

Then by definition of Θ:

$$\Theta P = \prod_{B \in \mathcal{B}} [X \subseteq B \cup I \rightarrow P'_{B,X}]$$

where $P'_{B,X} = (\Theta P_{B,X-B})$ **if** $B \subseteq X$ **else** $\bot_{I \cup O}$

By induction $P'_{B,X}$ is in normal form.

It remains to show $B \cup I \subseteq B' \cup I \Rightarrow \forall X \subseteq B \cup I \bullet P'_{B',X} \sqsubseteq P_{B,X}$

Now $B \cup I \subseteq B' \cup I \Rightarrow B \subseteq B'$

If $B = B'$ then as the choice sets are unique in the definition of P, we have $P'_{B',X} = P'_{B,X}$ and we are done.

If $B \subset B'$ then $X \subseteq B \cup I \Rightarrow B' \not\subseteq X$ so by definition $P'_{B',X} = \perp_{I \cup O}$ and the result follows. □

Lemma 6.15 *If $P, Q \in SRPT^I$ are in normal form with alphabets I and O then*

$$\vdash \Theta P \sqsubseteq \Theta Q \Rightarrow \vdash_R \quad P \sqsubseteq_R Q$$

Proof: By structural induction over P.

base case: $P = \perp_{I,O}$. the result follows by o-6.

inductive step:

$$P = \prod_{B \in \mathcal{B}} [!B?X \rightarrow P_{B,X}]$$

by the definition of Θ:

$$\Theta P = \prod_{B' \in \mathcal{B}'} [Y \subseteq B' \rightarrow P'_{B',Y}]$$

where $P'_{B',Y} = (\Theta(P_{B' \cap O, Y \cap I})$ **if** $B' = Y \cup I$ **else** $\perp_{I \cup O})$ and $\mathcal{B}' = \{B \cup I \mid B \in \mathcal{B}\}$. By Lemma 6.14 ΘP and ΘQ are in normal form in SCSP^I. Moreover as the proof system for SCSP^I is sound $\mathcal{M}[\![\Theta P]\!] \sqsubseteq \mathcal{M}[\![\Theta Q]\!]$. So following the argument of Lemma 3.16, ΘQ must take the form of a non-deterministic choice of set prefixed constructs. Hence Q must also take this form.

$$Q = \prod_{C \in \mathcal{C}} [!C?X \rightarrow Q_{C,X}]$$

with:

$$\Theta Q = \prod_{C' \in \mathcal{C}'} [Y \subseteq C' \rightarrow Q'_{C',Y}]$$

where $Q'_{C',Y} = (\Theta(Q_{C' \cap O, Y \cap I})$ **if** $C' = Y \cup I$ **else** $\perp_{I \cup O})$ and $\mathcal{C}' = \{C \cup I \mid C \in \mathcal{C}\}$.

following the argument of Lemma 3.16, $\mathcal{C}' \subseteq \mathcal{B}'$, hence $\mathcal{C} \subseteq \mathcal{B}$. So by axiom o–1

$$P \sqsubseteq \prod_{C \in \mathcal{C}} [!C?X \rightarrow P_{C,X}]$$

it is sufficient to show that $\vdash \ (\Theta P_{C,X}) \sqsubseteq (\Theta Q_{C,X})$ for all $C \in \mathcal{C}$, $X \subseteq I$ then the result follows by induction and the monotonicity of operators.

As $\mathcal{M}[\![\Theta P]\!] \sqsubseteq \mathcal{M}[\![\Theta Q]\!]$ it follows that:

$\forall C' \in \mathcal{C}' ; Y \subseteq C' \bullet \mathcal{M}[\![P'_{C',Y}]\!] \sqsubseteq \mathcal{M}[\![Q'_{C',Y}]\!]$
$\Rightarrow$ { by definition of P' and Q' }
$\forall C' \in \mathcal{C}' ; Y \subseteq C' \bullet C' = Y \cup I \Rightarrow \mathcal{M}[\![\Theta P_{C' \cap \mathcal{O}, Y \cap I}]\!] \sqsubseteq \mathcal{M}[\![\Theta Q_{C' \cap \mathcal{O}, Y \cap I}]\!]$
$\Rightarrow$ { by definition of $\mathcal{C}'$ }
$\forall C \in \mathcal{C} ; X \subseteq I \bullet \mathcal{M}[\![\Theta P_{C,X}]\!] \sqsubseteq \mathcal{M}[\![\Theta Q_{C,X}]\!]$
$\Rightarrow$ { as the proof system for SCSP^I is complete }
$\forall C \in \mathcal{C} ; X \subseteq I \bullet \vdash \ \Theta P_{C,X} \sqsubseteq \Theta Q_{C,X}$

giving the required result. □

Theorem 6.16 (Completeness) *For $P, Q \in SRPT^I$ with alphabets I and O,*

$$\mathcal{M}_R[\![P]\!] \sqsubseteq_R \mathcal{M}_R[\![Q]\!] \Rightarrow (\vdash_R \ P \sqsubseteq_R Q)$$

Proof: By Lemma 6.13 we can find $P', Q' \in \mathrm{SRPT}^I$ in normal form with $\vdash_R \ P \equiv_R P'$ and $\vdash_R \ Q \equiv_R Q'$.

As the proof system for SRPT^I is sound

$\mathcal{M}_R[\![P' \sqcap Q']\!] = \mathcal{M}_R[\![P \sqcap Q]\!] = \mathcal{M}_R[\![P]\!] = \mathcal{M}_R[\![P']\!]$.

So it is sufficient to prove $\mathcal{M}_R[\![P' \sqcap Q']\!] = \mathcal{M}_R[\![P']\!] \Rightarrow \vdash_R \ P' \sqsubseteq_R Q'$

$\mathcal{M}_R[\![P' \sqcap Q']\!] = \mathcal{M}_R[\![P']\!]$
$\Rightarrow \Phi\mathcal{M}_R[\![P' \sqcap Q']\!] = \Phi\mathcal{M}_R[\![P']\!]$ { as Φ is a function }
$\Rightarrow \mathcal{M}[\![(\Theta P') \sqcap (\Theta Q')]\!] = \mathcal{M}[\![\Theta P']\!]$ { by defn. of Θ and Theorem 6.10 }
$\Rightarrow \vdash \ \theta P' \sqsubseteq \Theta Q'$ { by Theorem 3.17 }
$\Rightarrow \vdash_R \ P' \sqsubseteq_R Q'$ { by Lemma 6.15 }

□

An extended proof system

As for SCSP, we extend our proof system to cover the full language of closed terms in SRPT. This involves characterising each process by its set of finite syntactic approximations, thus enabling us to reason about infinite processes.

Definition 6.6 The relation $\prec_R$ is the smallest relation on closed terms of SRPT with alphabets I and O satisfying:

$$\begin{array}{rcl}
\bot & \prec_R & P \\
P & \prec_R & P \\
P[(\mu\, x \bullet P)/x] & \prec_R & \mu\, x \bullet P \\
P \prec_R Q \prec_R R & \Rightarrow & P \prec_R R \\
P_1 \prec_R Q_1, P_2 \prec_R Q_2 & \Rightarrow & (P_1 \sqcap P_2) \prec_R (Q_1 \sqcap Q_2) \\
\forall X \subseteq I \bullet P_X \prec_R Q_X & \Rightarrow & [!B?X \to P_X] \prec_R [!B?X \to Q_X] \\
P_1 \prec_R Q_1, P_2 \prec_R Q_2 & \Rightarrow & (P_1 \parallel P_2) \prec_R (Q_1 \parallel Q_2) \\
P \prec_R Q & \Rightarrow & (P \setminus A) \prec_R (Q \setminus A) \\
P \prec_R Q & \Rightarrow & P[S] \prec_R Q[S]
\end{array}$$

If $P \prec_R Q$ then we say that P is a *syntactic approximation* of Q. ◊

It can be shown by structural induction that the ordering given by $\prec_R$ is weaker that the information ordering, and hence the non-determinism ordering.

$$P \prec_R Q \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!] \leqslant \mathcal{M}_{\mathcal{R}}[\![Q]\!] \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]$$

Given a closed process P, we construct the set of its finite syntactic approximations $FIN_R(P)$. We say a process is finite exactly when it is a term in the language SRPT^f. Thus we have the following:

Definition 6.7 $FIN_R(P) \triangleq \{Q \in \mathrm{SRPT}^f \mid Q \prec_R P\}$ ◊

$FIN_R(P)$ forms a directed set under $\prec_R$ and consequently the semantic image forms a directed set under $\sqsubseteq_R$ and $\leqslant$. Following the argument used to deduce Theorem 3.19 we obtain the result below.

Lemma 6.17 $\mathcal{M}_{\mathcal{R}}[\![P]\!] = \bigsqcup^{\leqslant}_{Q \in FIN_R(P)} \mathcal{M}_{\mathcal{R}}[\![Q]\!]$ ■

Now we can show that $\bigsqcup_{Q \in FIN_R(P)} \mathcal{M}_{\mathcal{R}}[\![Q]\!]$ (the least upper bound under the $\sqsubseteq_R$ ordering) coincides with $\bigsqcup^{\leqslant}_{Q \in FIN_R(P)} \mathcal{M}_{\mathcal{R}}[\![Q]\!]$.

Lemma 6.18 *If D is a set of closed terms in the language SRPT and $\{\mathcal{M}_{\mathcal{R}}[\![P]\!] \mid P \in D\}$ is $\leqslant$-directed, then*

$$\bigsqcup_{P \in D} \mathcal{M}_{\mathcal{R}}[\![P]\!] = \bigsqcup^{\leqslant}_{P \in D} \mathcal{M}_{\mathcal{R}}[\![P]\!]$$

Proof: Using the links established between RM and SM we can establish that $\mathcal{M}_{\mathcal{R}}[\![P]\!] \leq \mathcal{M}_{\mathcal{R}}[\![Q]\!] \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!] \Rightarrow \Phi\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq \Phi\mathcal{M}_{\mathcal{R}}[\![Q]\!]$.
While from Theorem 6.5

$$\Phi(\bigsqcup_{P \in D}^{\leq} \mathcal{M}_{\mathcal{R}}[\![P]\!]) = \bigsqcup_{P \in D} (\Phi\mathcal{M}_{\mathcal{R}}[\![P]\!])$$

it follows that

$$\Phi(\bigsqcup_{P \in D}^{\leq} \mathcal{M}_{\mathcal{R}}[\![P]\!]) = \Phi(\bigsqcup_{P \in D} \mathcal{M}_{\mathcal{R}}[\![P]\!])$$

The result follows as Φ, restricted to processes with the same alphabets, is injective. □

Combining these results we have:

Theorem 6.19 $\mathcal{M}_{\mathcal{R}}[\![P]\!] = \bigsqcup_{Q \in FIN_R(P)} \mathcal{M}_{\mathcal{R}}[\![Q]\!]$. ■

We extend the proof system for SRPT^I with the following:

a–17 $\quad \vdash P[(\mu x \bullet P)/x] \equiv_R \mu x \bullet P$

R–1 $\quad \dfrac{\forall Q \in FIN_R(P) \bullet Q \sqsubseteq_R R}{P \sqsubseteq_R R}$

These rules are similar to those presented in Section 3.4.2. They allow us to extend the proof system to all closed terms of SRPT.

Soundness and completeness

The least fixed point construction of the semantics of recursive constructs guarantees the soundness of axiom a–17. While the inference rule is sound by Theorem 6.19.

Theorem 6.20 (Soundness) *For all closed terms P and Q in SRPT*

$$(\vdash_R \ P \sqsubseteq_R Q) \Rightarrow \mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!]$$

■

Completeness is established by considering the characterisation of a process by its syntactic approximation in the same way as presented in Theorem 3.21.

Theorem 6.21 (Completeness) *For all closed terms P and Q in SRPT*

$$\mathcal{M}_{\mathcal{R}}[\![P]\!] \sqsubseteq_R \mathcal{M}_{\mathcal{R}}[\![Q]\!] \Rightarrow (\vdash_R \ P \sqsubseteq_R Q).$$

■

6.4 Conclusion

By considering the way in which one might model receptive systems in the language SCSP, we have established an embedding of the language SRPT and its associated model into the language and model of SCSP. The embedding between models RM and SM was proved to be continuous; this made it possible to establish many of the mathematical results concerning SRPT via the equivalent results for SCSP and the embedding. Results assumed in Chapter 5 have been proved here; the semantic images of operators in the language were shown to be continuous. Also, the soundness and completeness of the proof system for SRPT was deduced from similar results in SCSP. The embedding and the results drawn from it demonstrate how SRPT can be viewed as a receptive sublanguage of SCSP.

Chapter 7

Timewise Abstraction

We are already familiar with abstraction as a powerful development tool. During development it is often necessary to be aware of communication between components of the system which ultimately should be hidden from the environment. Abstraction allows us to hide from the environment those details of the specification which can be viewed as specific only to the internal working of the system. The languages presented in this thesis allow communication abstraction via the hiding operator.

Similarly, when modelling a system by a discrete time algebra, it may be appropriate to view the internal working of components of a system in a different time frame to that appropriate to the ultimate interaction between the whole system and the environment. Specifying components of the system in the context of a short clock cycle we could ascertain the details of the internal behaviour. Then the specification could be translated to a time frame with a longer clock cycle appropriate to the system as a whole. The ability to slow down the speed at which the process is viewed would allow us to reason about systems in the time frame as well as the communication level most appropriate to the final application. The procedure of translating a process in a discrete time algebra to a slower time frame will be referred to as *timewise abstraction*.

We recall that, when considering communication abstraction, we had to make assumptions concerning the circumstances under which hidden events should be performed internally; we used a maximal progress assumption. In the same way, when considering timewise abstraction we must take care to avoid ambiguity by making clear the context in which we can translate the time frame. Timewise abstraction can be usefully incorporated in the receptive language SRPT without ambiguity.

Suppose we have a receptive system modelled in SRPT in a given time frame, then by making certain assumptions we can derive a model for the same system in a new time frame. The new time frame is chosen such that a unit of time in

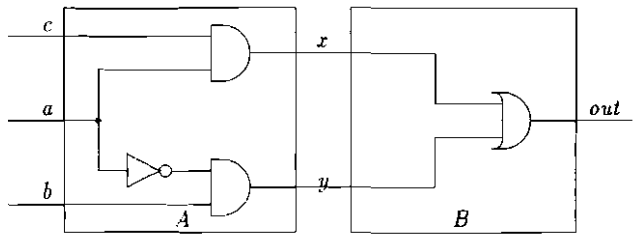


Figure 7.1: A conditioual circuit.

the new frame is an integer multiple of a unit of time in the original frame. We assume that input only changes at the tick of the clock in the new frame, so there is less opportunity for input to vary. In the new model, the value of output is only recorded on the tick of the new, slower clock. The new model provides less information about the system's behaviour as it takes 'snapshots' of the system less frequently. For systems in which such assumptions are appropriate we will provide algebraic methods to perform timewise abstraction, changing the time frame in which the system is viewed.

The language SRPT gives us a way of modelling components with latched inputs. The input received by such a component at the start of the clock cycle is latched and held constant for the remainder of the clock cycle. In SRPT we are able to record the behaviour of the component by giving the input and output observed on the tick of the clock. We give a stroboscopic view of the system which, if timed to coincide with the system clock, gives us a useful representation of components.

Suppose we require a clock speed such that the output is stable after one time unit. A system is often composed of several subcomponents. Even if each of the subcomponents stabilises within one time unit, it is not necessarily the case that the whole system will stabilise within one time unit. The system as a whole should be latched at a speed which ensures that it is stable after a single clock cycle. Timewise abstraction allows us to investigate the effect of latching the whole system at a slower speed.

Example: A conditional circuit

A conditional circuit can be built from components A and B as shown in Figure 7.1. Suppose both components stabilise within time d. If we take a unit of time in SRPT to be length d then we can model components A and B as follows (using the conventions introduced in Chapter 5 and the definition of OR from Section 5.2).

$$A \mathrel{\widehat{=}} [!\{\}?X \rightarrow A(X)]$$

where

$$A(X) = A_{ac} \textbf{ if } a \in X \wedge c \in X \textbf{ else } (A_b \textbf{ if } a \notin X \wedge b \in X \textbf{ else } A)$$
$$A_{ac} \mathrel{\hat{=}} [!\{x\}?X \rightarrow A(X)]$$
$$A_b \mathrel{\hat{=}} [!\{y\}?X \rightarrow A(X)]$$

and

$$B \mathrel{\hat{=}} OR[x/a, y/b, out/c]$$

Now to consider the behaviour of the complete circuit we must consider the process $(A \parallel B) \setminus \{x, y\}$. It can be shown using the laws of SRPT that

$$(A \parallel B) \setminus \{x, y\} \equiv Con$$

where

$$Con \mathrel{\hat{=}} [!\{\}?X \rightarrow (Con' \textbf{ if } (a \in X \wedge c \in X) \vee (b \in X \wedge a \notin X) \textbf{ else } Con)]$$
$$Con' \mathrel{\hat{=}} [!\{\}?X \rightarrow (Con'_l \textbf{ if } (a \in X \wedge c \in X) \vee (b \in X \wedge a \notin X) \textbf{ else } Con_l)]$$
$$Con_l \mathrel{\hat{=}} [!\{out\}?X \rightarrow (Con' \textbf{ if } (a \in X \wedge c \in X) \vee (b \in X \wedge a \notin X) \textbf{ else } Con)]$$
$$Con'_l \mathrel{\hat{=}} [!\{out\}?X \rightarrow (Con'_l \textbf{ if } (a \in X \wedge c \in X) \vee (b \in X \wedge a \notin X) \textbf{ else } Con_l)]$$

We have been able to hide the internal communication but clearly the whole circuit will take two time units to stabilise. It would be easier to reason about the circuit if it stabilised in one time unit. We would like to look at the whole system in a timeframe where each time unit has length $2d$. In such a time frame we would expect the circuit to be modelled by the process:

$$Cond \mathrel{\hat{=}} [!\{\}?X \rightarrow Cond(X)]$$

where

$$Cond(X) = Cond' \textbf{ if } (a \in X \wedge c \in X) \vee (a \notin X \wedge b \in X) \textbf{ else } Cond$$
$$Cond' \mathrel{\hat{=}} [!\{out\}?X \rightarrow Cond(X)]$$

Our aim in this chapter is to provide a simple algebraic method to perform timewise abstraction and hence derive process $Cond$ from process Con.

In this chapter we develop the theory of timewise abstraction; we provide simple algebraic laws for its application in SRPT and verify that it is consistent with the model. We also show how timewise abstraction can be usefully employed to reason about pipes. Through the examples it will become clear that timewise abstraction often reduces the number of states of a process, making the system easier to reason about.

7.1 Timewise Abstraction in SRPT

Rather than extend the syntax of SRPT, timewise abstraction will be defined in terms of a map between two copies of the closed terms of SRPT. We regard the two copies of the language as having different time frames, in that the unit of time has a different absolute length in the two copies of the language. We could view each copy of the language to be associated with a clock; observations are only made when this clock ticks. Timewise abstraction maps processes with a given associated clock to processes with an associated clock which ticks less frequently. The effect of timewise abstraction is to slow down the frequency at which observations are made of the system, under certain assumptious about the behaviour at the times observations are no longer made.

Suppose P is a process which models a system in a time frame with time t between ticks of the clock. If C is a set of events in the input alphabet of P and n and m are natural numbers, with n non-zero, theu $Slow(n, m, C, P)$ is a process which models the system in a time frame with time $n.t$ between ticks of the clock, such that:

- The first clock tick in the new frame coincides with the $(m+1)^{th}$ tick of the old clock.
- Subsequent clock ticks are made at $n.t$ time intervals; coinciding with the $(m+1+k.n)^{th}$ ticks of the clock in the old frame, for $k \in \mathbb{N}$.
- Until the first clock tick is made in the new frame, input to the system modelled by $Slow(n, m, C.P)$ is assumed to be C.
- The input is assumed to be held fixed at the value seen at a tick of the clock in the new frame until the subsequent clock tick occurs.

Timewise abstractiou preserves alphabets, so if P has input alphabet I and output alphabet O, so too does process $Slow(n, m, C, P)$.

From Section 6.3.2, closed process terms can be characterised by their finite syntactic approximations and each finite closed process is equivalent to a process in normal form. So we can define $Slow(n, m, C, P)$, up to $\equiv$ equivalence, over the closed terms of SRPT by the following axioms:

a–18 : $Slow(n, m, C, \bot_{I,O}) \equiv \bot_{I,O}$

a–19 : $Slow(n, m, C, P \sqcap Q) \equiv Slow(n, m, C, P) \sqcap Slow(n, m, C, Q)$

a–20 : $$Slow(n, m, C, [!B?X \to P_X]) \equiv \begin{cases} [!B?X \to Slow(n, n-1, X, P_X)] & \text{if } m = 0 \\ Slow(n, m-1, C, P_C) & \text{if } m > 0 \end{cases}$$

We also have the following law, which demonstrates that communication abstraction and timewise abstraction are independent of one another.

l–11 : $Slow(n, m, C, P \setminus A) \equiv (Slow(n, m, C, P)) \setminus A$

Notes

1. If Chaotic behaviour can occur at any time between two consecutive clock ticks in the new frame, then we assume it does occur.

2. If $m = 0$ then the first clock tick in the new frame coincides with the first clock tick in the old frame and there is no need to record the value of input prior to the first clock tick in the new frame. Thus, if I is the input alphabet of P then

$$\forall C, C' \in \mathbb{F}\, I \bullet Slow(n, 0, C, P) = Slow(n, 0, C', P).$$

3. If $n = 1$ then the two time frames are identical and $Slow(1, m, C, P)$ behaves like process P with the input held fixed at the value C for the first m time units. It follows that

$$Slow(1, 0, C, P) = P.$$

4. Had we introduced $Slow(n, m, C, P)$ into the syntax of the language, then expressions like

$$\mu P \bullet [!\{\}?X \rightarrow Slow(n, m, C, P)]$$

would be valid. The exact meaning of such an expression is not clear however, because timewise abstraction alters the time frame and the time frame is assumed to be fixed in the language. Considering timewise abstraction as a map between copies of the language allows us to realise the implications of timewise abstraction on the time frames.

7.2 Examples

7.2.1 A conditional circuit

Let us return to the example presented in the introduction to this chapter. We are interested in considering the process Con in a time frame where the clock ticks half as frequently and the initial clock tick in the new frame coincides with the

initial clock tick in the original time frame. So we are now interested in evaluating $Slow(2, 0, \{\}, Con)$.

Recall

$$\begin{array}{lll} Con & \widehat{=} & [!\{\}?X \rightarrow (Con' \textbf{ if } B(X) \textbf{ else } Con)] \\ Con' & \widehat{=} & [!\{\}?X \rightarrow (Con'_t \textbf{ if } B(X) \textbf{ else } Con_t)] \\ Con_t & \widehat{=} & [!\{out\}?X \rightarrow (Con' \textbf{ if } B(X) \textbf{ else } Con)] \\ Con'_t & \widehat{=} & [!\{out\}?X \rightarrow (Con'_t \textbf{ if } B(X) \textbf{ else } Con_t)] \end{array}$$

where $B(X)$ is the boolean given by:

$$B(X) \widehat{=} (a \in X \wedge c \in X) \vee (b \in X \wedge a \notin X)$$

Now:

$Slow(2, 0, \{\}, Con)$

$\equiv$ { expanding definition of Con }

$Slow(2, 0, \{\}, [!\{\}?X \rightarrow (Con' \textbf{ if } B(X) \textbf{ else } Con)])$

$\equiv$ { by a-20 }

$[!\{\}?X \rightarrow (Slow(2, 1, X, Con') \textbf{ if } B(X) \textbf{ else } Slow(2, 1, X, Con))]$

$\equiv$ { expanding definition of Con and Con' }

$[!\{\}?X \rightarrow (Slow(2, 1, X, [!\{\}?Y \rightarrow (Con'_t \textbf{ if } B(Y) \textbf{ else } Con_t)])$
$\textbf{if } B(X) \textbf{ else}$
$Slow(2, 1, X, [!\{\}?Y \rightarrow (Con' \textbf{ if } B(Y) \textbf{ else } Con)]))]$

$\equiv$ { by a-20 }

$[!\{\}?X \rightarrow (Slow(2, 0, X, Con'_t) \textbf{ if } B(X) \textbf{ else } Slow(2, 0, X, Con))]$

$\equiv$ { recalling note 2 }

$[!\{\}?X \rightarrow (Slow(2, 0, \{\}, Con'_t) \textbf{ if } B(X) \textbf{ else } Slow(2, 0, \{\}, Con))]$

similarly we can show that

$$Slow(2, 0, \{\}, Con'_t) \equiv$$
$$[!\{out\}?X \rightarrow (Slow(2, 0, \{\}, Con'_t) \textbf{ if } B(X) \textbf{ else } Slow(2, 0, \{\}, Con))]$$

so by uniqueness of solutions to guarded recursive equations:

$$Slow(2, 0, \{\}, Con) \equiv Cond$$

where $Cond$ is as given in the introduction to this chapter (page 122).

7.2.2 A grey-code counter

Our aim in this example is to verify the design of a 2-bit grey-code counter, such a counter should output, in sequence, the bit patterns

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow \ldots$$

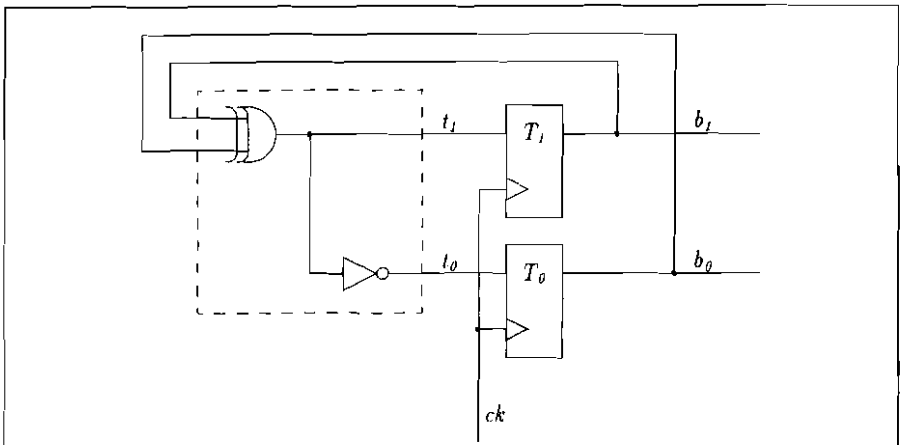


Figure 7.2: A grey-code counter

A change in output is triggered by the system clock. Notice that each increment of the counter only involves one bit changing: such codes eliminate the risk of glitches in the counter, which could be encountered when using a simple binary counter. The component we shall investigate is a sequential circuit consisting of a combinatorial part and two clocked T-type flip-flops configured as shown in Figure 7.2. This circuit exhibits feedback. Unlike previous circuits we have considered output depends on the previous state of the system rather than the values of last input. The only input to this system is the clock. The output of the system is determined by two wires b_0 and b_1 which encode the grey code in their voltage levels.

We shall investigate the behaviour of this circuit, takiug advantage of timewise abstraction to model each component in the most appropriate time frame.

The combinatorial circuit

The combinatorial circuit comprises of two gates, an *EXOR* gate and a *NOT* gate. We model both these gates in a time frame which ensures that the output is stable one time unit after the input is made available. The definition of a *NOT* gate with input wire a and output wire b, is given by:

$$\iota NOT = \{a\} \qquad oNOT = \{b\}$$
$$NOT \triangleq [!\{b\}?X \rightarrow (NOT \textbf{ if } X = \{\} \textbf{ else } NOT')]$$
$$NOT' \triangleq [!\{\}?X \rightarrow (NOT \textbf{ if } X = \{\} \textbf{ else } NOT')]$$

While we recall the definition of an *EXOR* gate from Section 5.2.1

$$\iota EXOR = \{a, b\} \qquad oEXOR = \{c\}$$
$$EXOR \triangleq [!\{\}?X \rightarrow (EXOR' \textbf{ if } |X| = 1 \textbf{ else } EXOR)]$$
$$EXOR' \triangleq [!\{c\}?X \rightarrow (EXOR' \textbf{ if } |X| = 1 \textbf{ else } EXOR)]$$

The combinatorial circuit is then given by:

$$EXOR[b_1/a, b_0/b, t_1/c] \parallel NOT[t_1/a, t_0/b].$$

By applications of the axioms of SRPT, we can eliminate parallel composition from the above and prove the following identity:

$$EXOR[b_1/a, b_0/b, t_1/c] \parallel NOT[t_1/a, t_0/b] \equiv EN.$$

where
$$EN \triangleq [!\{t_0\}?X \rightarrow (EN' \textbf{ if } |X| = 1 \textbf{ else } EN)]$$
$$EN' \triangleq [!\{t_0, t_1\}?X \rightarrow (EN_1' \textbf{ if } |X| = 1 \textbf{ else } EN_1)]$$
$$EN_1' \triangleq [!\{t_1\}?X \rightarrow (EN_1' \textbf{ if } |X| = 1 \textbf{ else } EN_1)]$$
$$EN_1 \triangleq [!\{\}?X \rightarrow (EN' \textbf{ if } |X| = 1 \textbf{ else } EN)]$$

We notice that this circuit takes longer that one time unit to stabilise. The input must be held constant for sufficient time for the race condition on the wires t_1 and t_0, (caused by the *NOT* gate), to pass before output is used. We are interested in the combinatorial circuit being modelled in a time frame which ensures output is stable after one time unit. If input is held fixed for two units the output after this time is stable. We consider the circuit in a time frame which is a factor of 2 slower.

$$COMB \triangleq Slow(2, 0, \{\}, EN)$$

defining $COMB' \triangleq Slow(2, 0, \{\}, EN_1')$ we can use the axioms for timewise abstraction to obtain the following expansion of *COMB*.

$$COMB \equiv [!\{t_0\}?X \rightarrow (COMB' \textbf{ if } |X| = 1 \textbf{ else } COMB)]$$
$$COMB' \equiv [!\{t_1\}?X \rightarrow (COMB' \textbf{ if } |X| = 1 \textbf{ else } COMB)]$$

A clocked T-type flip-flop

A rising edge triggered T-type flip-flop has one output a and two inputs t and ck. The value of the output remains fixed unless the input t is high on the rising edge of the clock ck. If the input t is high on the rising edge of the clock, then the output toggles, that is, changes from high to low or from low to high.

We shall assume that the flip-flop is modelled in a time frame such that all changes in the system clock ck coincide with observations in the model. The value on the wire ck should be seen to represent the state of the clock after any change at the time of observation in the model. The value on the wire t should be seen to

represent the stable state of the wire at the time of observation. We also assume that output is stable within one time unit of any change of input. So assuming that the clock is initially low and the output initially low we obtain the following description of a rising edge triggered T-type flip-flop:

$$\iota TFF = \{t, ck\} \qquad oTFF = \{a\} \qquad TFF \mathrel{\hat{=}} TFF_L$$

where $TFF_L \mathrel{\hat{=}} [!\{\}?X \rightarrow (TFF_L \textbf{ if } ck \notin X \textbf{ else } (TFF'_H \textbf{ if } t \in X \textbf{ else } TFF_H))]$
$TFF_H \mathrel{\hat{=}} [!\{\}?X \rightarrow (TFF_H \textbf{ if } ck \in X \textbf{ else } TFF_L)]$
$TFF'_H \mathrel{\hat{=}} [!\{a\}?X \rightarrow (TFF'_H \textbf{ if } ck \in X \textbf{ else } TFF'_L)]$
$TFF'_L \mathrel{\hat{=}} [!\{a\}?X \rightarrow (TFF'_L \textbf{ if } ck \notin X \textbf{ else } (TFF_H \textbf{ if } t \in X \textbf{ else } TFF'_H))]$
(L and H represent the current state of the clock.)

The complete circuit

We are now in a position to investigate the complete circuit, we assume that the flip-flops and combinatorial circuit are modelled in the same time frame, one in which each of these units stabilises within one time unit. So the complete circuit, modelled in this time frame, is given by:

$$GREY \mathrel{\hat{=}} (COMB \parallel TFF[t_0/t, b_0/a] \parallel TFF[t_1/t, b_1/a]) \setminus \{t_0, t_1\}$$

In order to establish the behaviours of this process, we use the algebraic laws of SRPT to reduce the above expression to a form which does not involve parallel composition or hiding operators.

$$\begin{array}{ll} & GREY \\ \equiv & \{\text{by definition of } GREY\ \} \\ & (COMB \parallel TFF[t_0/t, b_0/a] \parallel TFF[t_1/t, b_1/a]) \setminus \{t_0, t_1\} \\ \equiv & \{\text{by l-2}\ \} \\ & (COMB \parallel (TFF[t_0/t, b_0/a] \parallel TFF[t_1/t, b_1/a])) \setminus \{t_0, t_1\} \end{array}$$

Now by a-10 and a-16 and the definition of TFF

$$\begin{array}{l} TFF[t_0/t, b_0/a] \parallel TFF[t_1/t, b_1/a] \\ \quad \equiv [!\{\}?X \rightarrow ((TFF_L[t_0/t, b_0/a] \parallel TFF_L[t_1/t, b_1/a]) \textbf{ if } ck \notin X \textbf{ else} \\ \qquad\qquad ((TFF'_H[t_0/t, b_0/a] \parallel TFF'_H[t_1/t, b_1/a]) \textbf{ if } \{t_0, t_1\} \subseteq X \textbf{ else} \\ \qquad\qquad ((TFF_H[t_0/t, b_0/a] \parallel TFF'_H[t_1/t, b_1/a]) \textbf{ if } t_1 \in X \textbf{ else} \\ \qquad\qquad ((TFF'_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a]) \textbf{ if } t_0 \in X \textbf{ else} \\ \qquad\qquad (TFF_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a])))))] \end{array}$$

So

$$
\begin{array}{l}
\quad (COMB \parallel TFF[t_0/t, b_0/a] \parallel TFF[t_1/t, b_1/a]) \setminus \{t_0, t_1\} \\
\equiv \ \{ \text{ expanding } COMB \text{ and from above } \} \\
\quad ([!\{t_0\}?Y \rightarrow (COMB' \textbf{ if } |Y| = 1 \textbf{ else } COMB)] \\
\quad \|[!\{\}?X \rightarrow ((TFF_L[t_0/t, b_0/a] \parallel TFF_L[t_1/t, b_1/a]) \textbf{ if } ck \notin X \textbf{ else} \\
\qquad\qquad ((TFF'_H[t_0/t, b_0/a] \parallel TFF'_H[t_1/t, b_1/a]) \textbf{ if } \{t_0, t_1\} \subseteq X \textbf{ else} \\
\qquad\qquad ((TFF_H[t_0/t, b_0/a] \parallel TFF'_H[t_1/t, b_1/a]) \textbf{ if } t_1 \in X \textbf{ else} \\
\qquad\qquad ((TFF'_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a]) \textbf{ if } t_0 \in X \textbf{ else} \\
\qquad\qquad (TFF_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a])))))] \setminus \{t_0, t_1\}) \\
\equiv \ \{ \text{ by a-10 } \} \\
\quad [!\{t_0\}?X \rightarrow ((COMB \parallel (TFF_L[t_0/t, b_0/a] \parallel TFF_L[t_1/t, b_1/a])) \\
\qquad\qquad \textbf{if } ck \notin X \textbf{ else} \\
\qquad\qquad (COMB \parallel (TFF'_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a])))] \setminus \{t_0, t_1\} \\
\equiv \ \{ \text{ by a-13 } \} \\
\quad [!\{\}?X \rightarrow ((COMB \parallel (TFF_L[t_0/t, b_0/a] \parallel TFF_L[t_1/t, b_1/a])) \setminus \{t_0, t_1\} \\
\qquad\qquad \textbf{if } ck \notin X \textbf{ else} \\
\qquad\qquad (COMB \parallel (TFF'_H[t_0/t, b_0/a] \parallel TFF_H[t_1/t, b_1/a])) \setminus \{t_0, t_1\})]
\end{array}
$$

By continuing to eliminate parallel composition and hiding constructs we can demonstrate that

$$GREY \equiv G(L, 0)$$

where
$$
\begin{array}{lll}
G(H,0) & \hat{=} & [!\{\}?X \rightarrow (G(H,0) \textbf{ if } ck \in X \textbf{ else } G(L,0))] \\
G(L,0) & \hat{=} & [!\{\}?X \rightarrow (G(L,0) \textbf{ if } ck \notin X \textbf{ else } G(H,1))] \\
G(H,1) & \hat{=} & [!\{b_0\}?X \rightarrow (G(H,1) \textbf{ if } ck \in X \textbf{ else } G(L,1))] \\
G(L,1) & \hat{=} & [!\{b_0\}?X \rightarrow (G(L,1) \textbf{ if } ck \notin X \textbf{ else } G(H,2))] \\
G(H,2) & \hat{=} & [!\{b_0, b_1\}?X \rightarrow (G(H,2) \textbf{ if } ck \in X \textbf{ else } G(L,2))] \\
G(L,2) & \hat{=} & [!\{b_0, b_1\}?X \rightarrow (G(L,2) \textbf{ if } ck \notin X \textbf{ else } G(H,3))] \\
G(H,3) & \hat{=} & [!\{b_1\}?X \rightarrow (G(H,3) \textbf{ if } ck \in X \textbf{ else } G(L,3))] \\
G(L,3) & \hat{=} & [!\{b_1\}?X \rightarrow (G(L,3) \textbf{ if } ck \notin X \textbf{ else } G(H,0))]
\end{array}
$$

The numerical parameter of G indicates the current phase of the counter.

Incorporating the clock

By way of example, suppose we have an asymmetric system clock which has low time twice as long as its high time. This can be described by the process CLK or pictorially as in Figure 7.3.

We want to incorporate the clock into the system and abstract away from details of the clock which are considered internal to the system. To achieve this we consider the process $(CLK \parallel GREY)$. By hiding the event ck we can make internal the

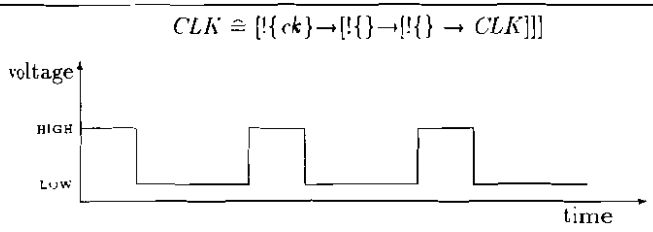


Figure 7.3: Two descriptions of an asymmetric clock

behaviour of the clock. As we are not interested in the mechanics of the clock it also makes sense to consider the whole system in a time frame where a unit of time coincides with a complete cycle of the system clock. We choose to make observations coincide with the rising edge of the clock cycle. So we evaluate the following:

$$SYS \triangleq Slow(3, 0, \{\}, (GREY \parallel CLK) \setminus \{ck\})$$

Now

$$\begin{array}{ll} & GREY \parallel CLK \\ \equiv & \{\text{ by definition of } GREY,\ CLK \text{ and a–10 }\} \\ & [!\{ck\} \rightarrow (G(H,1) \parallel [!\{\} \rightarrow [!\{\} \rightarrow CLK]])] \\ \equiv & \{\text{ by definition of } G(H,0) \text{ and a–10 }\} \\ & [!\{ck\} \rightarrow [!\{b_0\} \rightarrow [!\{b_0\} \rightarrow (G(L,1) \parallel CLK)]]] \end{array}$$

Thus, noting that SYS has an empty input alphabet,

$$\begin{array}{ll} & SYS \\ \equiv & \{\text{ by definition of } SYS \ \} \\ & Slow(3, 0, \{\}, (GREY \parallel CLK) \setminus \{ck\}) \\ \equiv & \{\text{ by l–11 }\} \\ & Slow(3, 0, \{\}, (GREY \parallel CLK)) \setminus \{ck\} \\ \equiv & \{\text{ noting the above expansion and by a–20 }\} \\ & [!\{ck\} \rightarrow \\ & \quad Slow(3, 2, \{\}, [!\{b_0\} \rightarrow [!\{b_0\} \rightarrow (G(L,1) \parallel CLK)]])] \setminus \{ck\} \\ \equiv & \{\text{ by two applications of a–20 }\} \\ & [!\{ck\} \rightarrow Slow(3, 0, \{\}, (G(L,1) \parallel CLK))] \setminus \{ck\} \\ \equiv & \{\text{ by a–13 and l–11 }\} \\ & [!\{\} \rightarrow (Slow(3, 0, \{\}, (G(L,1) \parallel CLK) \setminus \{ck\}))] \end{array}$$

continuing in this manner and by the uniqueness of solutions to guarded recursive equations we can demonstrate that:

$$SYS \equiv \mu P \bullet [!\{\}\rightarrow[!\{b_0\}\rightarrow[!\{b_0, b_1\}\rightarrow[!\{b_1\}\rightarrow P]]]]$$

So our system clearly behaves like a grey-code counter.

By using timewise abstraction we have been able to present the process in a time frame in which its behaviour is easy to verify. The coarser time frame abstracts away from all the delays which arise when data abstraction is performed.

7.3 Relating Timewise Abstraction to the Model

In this section we shall establish the semantics of $Slow(n, m, C, P)$ in RM. Once we have defined the semantics of timewise abstraction we can verify that it is well defined and ensure that the axioms presented in Section 7.1 are sound with respect to the model. Before we present the semantics of $Slow(n, m, C, P)$ we introduce some notation.

7.3.1 Notation

We define two functions on traces, both of which will be associated with the concept of slowing down the frequency of observation.

Choose

The first function, $choose(n, m, s)$, takes as its arguments two natural numbers n, m and a trace s. It returns a trace which has as its $(k+1)^{th}$ element the $(m+1+n.k)^{th}$ element of s. If s corresponds to a trace of observations made in the original time frame, then $choose(n, m, s)$ corresponds to a trace of observations in a new time frame, where the clock runs slower that that in the original frame by a factor n, and the first tick of the clock in the new frame coincides with the $(m+1)^{th}$ clock tick in the original frame.

The formal definition of $choose$ is given for $n > 0$:

$$\begin{array}{ll} choose(n, m, \langle\rangle) & = \langle\rangle \\ choose(n, 0, \langle B\rangle^\frown s) & = \langle B\rangle^\frown choose(n, n-1, s) \\ choose(n, m+1, \langle B\rangle^\frown s) & = choose(n, m, s) \end{array}$$

Properties

- $|choose(n, m, s)| = \lceil (|s| - m)/n \rceil$. for $n \neq 0$.
- $choose(1, 0, s) = s$.

- $s \leq s' \Rightarrow choose(n, m, s) \leq choose(n, m, s')$

Trace multiplication

The operator $\otimes$ takes as its arguments a natural number and a trace. The result, $n \otimes s$ is a trace n times as long as s with each term duplicated n times. When considering systems viewed under the new, slower time frame we assume that the input to the system can only be changed at the (less frequent) ticks of the new clock. We will have to examine the effect of this input pattern on the system viewed in the original time frame in order to establish the output which will be observed in the new frame of reference.

We define $\otimes$ as follows:

$$\begin{array}{ll} n \otimes \langle\rangle & = \langle\rangle \\ n \otimes (\langle X \rangle^\frown s) & = \langle X \rangle^{n\frown} (n \otimes s) \end{array}$$

Properties

- $|n \otimes s| = n.|s|$

7.3.2 The semantics of timewise abstraction

We extend the semantic function $\mathcal{M}_{\mathcal{R}}$ to incorporate timewise abstraction. Recalling the definition of $\mathcal{M}_{\mathcal{R}}$, it is sufficient to define ι, o and $\mathcal{T}_{\mathcal{R}}$ of $\mathcal{S}low(n, m, C, P)$.

Definition 7.1 For $m \in \mathbb{N}$, $n \in \mathbb{N}^+$, $C \in \mathbb{F}\Sigma$, $P \in SRPT$ and $\sigma \in BIND_R$ we extend the definitions of ι, o and $\mathcal{T}_{\mathcal{R}}$ as follows:

$$\iota[\![\mathcal{S}low(n, m, C, P)]\!]\sigma \triangleq \iota[\![P]\!]\sigma$$
$$o[\![\mathcal{S}low(n, m, C, P)]\!]\sigma \triangleq o[\![P]\!]\sigma$$

$$\begin{array}{rl} \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma \triangleq \{s \mid & (\exists\, s' \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \bullet choose(n, m, s') = s \\ & \quad \wedge\ s' \cap I \leq \langle C \rangle^{m\frown}((n \otimes s) \cap I)) \\ & \wedge\ (\neg\ \exists\, s'' \in \widehat{\mathcal{T}}_{\mathcal{R}}[\![I']\!]\sigma \bullet choose(n, m, s'') < s \\ & \quad \wedge\ s'' \cap I \leq \langle C \rangle^{m\frown}((n \otimes s) \cap I))\} \\ & \text{where } I = \iota[\![P]\!]\sigma \end{array}$$

◊

Notes

1. Observing $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma$, it is clear that when $m = 0$ the semantics of $\mathcal{S}low(n, m, C, P)$ are independent of the value of C.

2. Notice that if $C \not\subseteq \iota[\![P]\!]\sigma$ and $m > 0$ then

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma = \{\langle\rangle\} = \mathcal{T}_{\mathcal{R}}[\![\ \bot\ ]\!]\sigma.$$

Until the first tick of the new clock the environment attempts to input to the process, if this input is not in the alphabet of the process the result is chaotic behaviour.

3. We can explain the definition of the semantics for $\mathcal{S}low(n, m, C, P)$ as follows. If s is a trace giving a possible behaviour of $\mathcal{S}low(n, m, C, P)$ then it is derived from a behaviour s' of P. By requiring $choose(n, m, s') = s$, we ensure that the $(k+1)^{th}$ observation in s coincides with the $(m+1+n.k)^{th}$ observation in s'. By requiring $s' \cap I \leq (C)^m \frown ((n \otimes s) \cap I)$, we guarantee that the behaviour s is derived from a behaviour in which input does not change between observations made in the new time frame.

   Finally, we exclude any behaviours which are extensions of behaviours derived from maximal behaviours of the original process. So if a behaviour could have been derived from a behaviour which results in the original process becoming uninformative, we assume that the process viewed in the new time frame becomes uninformative.

Theorem 7.1 *$\mathcal{S}low$ is well defined with respect to the model RM.*

Proof: It is necessary and sufficient that $\mathcal{M}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma \in RM$ for all process expressions P in SRPT, $m \in \mathbb{N}$, $n \in \mathbb{N}^+$ and $C \in \mathbb{F}\Sigma$. To achieve this we must show that $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma$ satisfies closure conditions I–III (see Section 5.3.2) with respect to alphabets $\iota[\![P]\!]\sigma$ and $o[\![P]\!]\sigma$, under the assumption that $\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma \in RM$. The proofs of all three conditions involve careful examination of the construction of the set defining $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma$; the proof of III is presented as Theorem A.5 in Appendix A.4. □

Theorem 7.2 *Axioms a-18, a-19 and a-20 are sound with respect to the model RM.*

Proof: It is necessary to prove that the following equalities hold for $m \in \mathbb{N}$, $n \in \mathbb{N}^+$ and C a subset of the input alphabet:

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, \bot)]\!] = \mathcal{T}_{\mathcal{R}}[\![\ \bot\ ]\!]$$

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P \sqcap Q)]\!] = \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P) \sqcap \mathcal{S}low(n, m, C, Q)]\!]$$

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, 0, C, [!B?X \rightarrow P_X])]\!] = \mathcal{T}_{\mathcal{R}}[\![[!B?X \rightarrow \mathcal{S}low(n, n-1, X, P_X)]]\!]$$

and for $m > 0$

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, [!B?X \rightarrow P_X])]\!] = \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m-1, C, P_C)]\!]$$

This can be shown by set analysis. The proof for a-19 is presented as Theorem A.8 in Appendix A.4. □

7.4 Pipes

Pipelining is a commonly used technique for obtaining speed up in sequential circuits. If many pieces of data require processing by a sequential circuit, and this sequential circuit can be decomposed into several smaller components which process the data in turn, then pipelining enables data processing to be overlapped. Each input does not have to wait for the previous one to be output. Typically such pipelining is achieved by latching intermediate results.

For example the conditional circuit introduced at the beginning of this chapter could be easily presented as a pipeline of length two units by latching the output from component A. Then, assuming both components A and B stabilise in unit time, new data can be input every unit and the corresponding output is available two units later. Such a circuit is described by the process Con.

Unfortunately as a pipeline increases in length, the process describing it must have more states, recording the data latched in the pipeline. Moreover the number of states may increase exponentially with the length of the pipeline. In this section we shall demonstrate how timewise abstraction can be used to simplify the reasoning about pipes.

Definition 7.2 A closed process P is a *pipe* if there exists $\ell_P > 0$ and a function $\mathcal{E}_P : \mathbb{P}(\iota P) \rightarrow (\mathbb{P}(\mathbb{P}(oP)) - \{\})$ such that

$$
\begin{array}{l}
P \text{ is a pipe of length } \ell_P \text{ with effect } \mathcal{E}_P \\
\quad \Leftrightarrow \; s \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \Rightarrow \exists\, X \in \mathbb{P}(O) \bullet s^\frown\langle X\rangle \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \\
\qquad \wedge \\
\qquad s^\frown\langle X\rangle^\frown s'^\frown\langle Y\rangle^\frown s'' \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \wedge |s'| = \ell_P - 1 \\
\qquad\qquad \Rightarrow\; Y \cap o[\![P]\!] \in \mathcal{E}_P(X \cap I) \\
\qquad\qquad\quad \wedge \\
\qquad\qquad\quad \forall B \subseteq I\,;\, Z \in \mathcal{E}_P(B) \bullet \\
\qquad\qquad\qquad s^\frown\langle (X \cap O) \cup B\rangle^\frown s'^\frown\langle (Y \cap I) \cup Z\rangle^\frown s'' \in \mathcal{T}_{\mathcal{R}}[\![P]\!]
\end{array}
$$

where $O = o[\![P]\!]$ and $I = \iota[\![P]\!]$. ◊

Notes

1. ℓ_P represents the non-zero length of the pipe. At any time $t > \ell_P$ output is solely dependent on the input ℓ_P units previously.

2. For a pipe P, the relationship between inpnt and output ℓ_P units of time later is characterised by the function $\mathcal{E}_P$. $\mathcal{E}_P$ defines the set of possible values of outpnt resulting from a given input set. We refer to $\mathcal{E}_P$ as the *effect* of P since this function describes all the possible consequences of processing input through the pipe. Clearly $\mathcal{E}_P$ is uniquely defined for a given process P.

Notice that the empty set is omitted from the range of $\mathcal{E}_P$. This follows our assumption that pipes are non-divergent. Terms in the range take the form of sets of possible output configurations. This makes provision for non-determinism in pipes. If a pipe P responds deterministically to a given input set B then $\mathcal{E}_P(B)$ is a singleton set.

3. The first condition on the behaviours states that P is non-divergent by ensuring that every behaviour of P can be extended.

4. The second condition states that output ℓ_P units after a given input B is one of the possible effects of the pipe on B. Any one of the possible effects in $\mathcal{E}_P(B)$ could have been observed without altering the subsequent behaviour. Moreover altering the input at any time only effects the output ℓ_P units later.

5. The output seen at the first ℓ_P ticks of the system clock will depend on the initial state of the pipe. It is only after this period of initialisation that the possible outputs can be deduced from earlier input.

We shall identify two pipes P and Q if they have the same length and the same effect. This identification disregards output during the period of initialisation; so P and Q may have different output during the first ℓ_P units of time. After this period of initialisation P and Q have the same behaviours.

Definition 7.3 If P and Q are pipes with the same input and output alphabets, then we say P and Q are *equivalent pipes*, $P \sim_p Q$ exactly when they have the same length and the same effect, formally:

$$P \sim_p Q \mathrel{\hat{=}} \mathcal{E}_P = \mathcal{E}_Q \wedge \ell_P = \ell_Q.$$

$\Diamond$

It is common practice in development to compose pipes to obtain longer pipes, we present the chaining operator for this purpose.

Definition 7.4 If $P, Q \in SRPT$ with $oP = \iota Q$ and $\iota P \cap oQ = \{\}$ then we define the chaining operator $\gg$ as follows:

$$P \gg Q \mathrel{\hat{=}} (P \parallel Q) \setminus oP$$

$\Diamond$

The chaining operator is a composite operator in which all communication between the processes being chained together is made internal. It follows that the process $P \gg Q$ has input alphabet ιP and output alphabet oQ. The following is an obvious law of the chaining operator:

I–12 : if $\iota Q \cap oR = \{\}$, $(P \gg Q) \gg R \equiv P \gg (Q \gg R)$

Proof: For the above to be well defined we can make the following deductions concerning the alphabets of P, Q and R.

$$oP = \iota Q \quad \text{and} \quad oQ = \iota R$$
$$\iota P \cap oQ = \{\}, \quad \iota P \cap oR = \{\} \quad \text{and} \quad \iota Q \cap oR = \{\}$$

From this we can show that $oP \cap \alpha R = \{\}$ and $oQ \cap \alpha P = \{\}$. Now

$$\begin{array}{lll}
(P \gg Q \gg R & \equiv ((P \gg Q) \parallel R) \setminus o(P \gg Q) & \{ \text{ defn. of chaining } \} \\
& \equiv ((P \gg Q) \parallel R) \setminus oQ & \{ \text{ defn. of alphabets } \} \\
& \equiv (((P \parallel Q) \setminus oP) \parallel R) \setminus oQ & \{ \text{ defn. of chaining } \} \\
& \equiv (((P \parallel Q) \parallel R) \setminus oP) \setminus oQ & \{ \text{ by I-4 } \} \\
& \equiv ((P \parallel (Q \parallel R)) \setminus oP) \setminus oQ & \{ \text{ by I-2 } \} \\
& \equiv (P \parallel (Q \parallel R)) \setminus oQ) \setminus oP & \{ \text{ by I-3 } \} \\
& \equiv (P \parallel ((Q \parallel R) \setminus oQ)) \setminus oP & \{ \text{ by I-4 } \} \\
& \equiv (P \parallel (Q \gg R)) \setminus oP & \{ \text{ by defn. chaining } \} \\
& \equiv P \gg (Q \gg R) & \{ \text{ by defn. chaining } \}
\end{array}$$ □

Theorem 7.3 *If P is a pipe of length ℓ_P with effect $\mathcal{E}_P$, Q is a pipe of length ℓ_Q with effect $\mathcal{E}_Q$, $oP = \iota Q$ and $oQ \cap \iota P = \{\}$, then $P \gg Q$ is a pipe of length $\ell_P + \ell_Q$ with effect $\mathcal{E}_{P \gg Q}$, where*

$$\forall B \subseteq \iota P \bullet \mathcal{E}_{P \gg Q}(B) \triangleq \bigcup_{C \in \mathcal{E}_P(B)} \mathcal{E}_Q(C)$$

Proof: This follows directly from the definitions of chaining and pipes. □

7.4.1 Timewise abstraction and pipes

In this section we look at the ways we can use timewise abstraction when reasoning about pipes. First we look at some of the properties of pipes and the timewise abstraction of pipes. Then we discuss the ways in which these results can be utilised to aid verification of the behaviour of pipes.

Theorem 7.4 *If P is a pipe of length ℓ_P and effect $\mathcal{E}_P$ then $\mathcal{S}low(\ell_P, 0, \{\}, P)$ is a pipe of length 1 with effect $\mathcal{E}_P$.* ■

Intuitively $\mathcal{S}low(\ell_P, 0, \{\}, P)$ represents the pipe P viewed in a time frame where a unit of time has the same length as the pipe. So there is sufficient time between observations for the pipe to completely process the input. Hence $\mathcal{S}low(\ell_P, 0, \{\}, P)$ behaves like a pipe of length 1.

Theorem 7.5 *If P and Q are pipes then*

$$Slow(\ell_P + \ell_Q, 0, \{\}, P \gg Q) \equiv$$
$$Slow(2, 0, \{\}, (Slow(\ell_P, 0, \{\}, P) \gg Slow(\ell_Q, 0, \{\}, Q)))$$

Proof: This follows noting that the initial output of both sides of the equivalence is the initial output of Q. □

Theorem 7.6 *If P and Q are pipes of length ℓ, with input and output alphabets I and O then:*

$$P \sim_p Q \Leftrightarrow \forall B \subseteq I \bullet Slow(1, \ell, B, P) \equiv Slow(1, \ell, B, Q)$$

Proof:

$\Rightarrow$: If P and Q are equivalent pipes of length ℓ then, if provided with the same input, their behaviours are the same after ℓ units of time. $Slow(1, \ell, B, P)$ gives a process with behaviours corresponding to those of P after ℓ units of time, assuming the input over those first ℓ units is the set B. Clearly the implication follows.

$\Leftarrow$: P is a pipe of length ℓ and by the definition of $Slow(1, \ell, B, P)$, the initial output of $Slow(1, \ell, B, P)$ corresponds to the effect of pipe P on input B. As $\forall B \subseteq I \bullet Slow(1, \ell, B, P) \equiv Slow(1, \ell, B, Q)$ it follows that the pipes P and Q have the same effect on all input sets $B \subseteq I$. Hence P and Q are equivalent pipes. □

When designing pipes the main concern is the pipe's behaviour once it is initialised. It is often unnecessary to concern ourselves with the output of the pipe during the period of initialisation. This is the time before the pipe outputs the data corresponding to the first input. If this is the case, then to verify the behaviour of pipe P of length ℓ_P it is sufficient to consider $Slow(\ell_P, 0, \{\}, P)$. This pipe has the same effect as P although it has length 1.

A pipe of length 1 can only have as many states as there are possible configurations of input. Supposing there are k configurations of input, a pipe of length ℓ_P can have as many as k^{ℓ_P} states. Clearly the pipe of length 1 is easier to verify that that of length ℓ_P since there are in general fewer states to consider.

Often, during development, a pipe is broken down into several components. where these components correspond to various stages in the pipeline. If each of these components is itself a pipe, as is often the case in such circumstances, then by making use of Theorem 7.5 we need never consider the full expansion of the overall pipeline when verifying its behaviour. Suppose P is a process which can be decomposed into components $P_1, P_2, ...P_n$ where each of the P_i is a pipe and

$$P \equiv P_1 \gg P_2 \gg ... \gg P_n$$

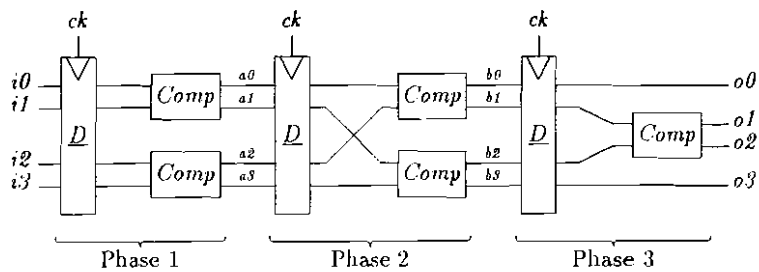


Figure 7.4: A Sorter

It follows that P is a pipe from Theorem 7.3. Moreover, as each of the P_i is a shorter pipe than P it is easier to verify that each of the P_i are pipes, especially if they are of length 1, than to show that P is a pipe directly. To investigate the effect of pipe P we can use the following iterative technique.

Setting $\quad Q_1 \triangleq Slow(\ell_{P_1}, 0, \{\}, P_1)$

$$Q_{m+1} \triangleq Slow(2, 0, \{\}, Q_m \gg Slow(\ell_{P_{m+1}}, 0, \{\}, P_{m+1}))$$

We can show by induction that

$$Q_m \equiv Slow(\Sigma_{i=1}^{m} \ell_{P_i}, 0, \{\}, P_1 \gg ... \gg P_m)$$

hence Q_n will have the same effect as pipe P. Using the above iterative technique to evaluate Q_n we only ever chain pipes of unit length.

In the example that follows we shall see these techniques applied in order to verify the behaviour of a pipelined sorter.

7.4.2 Example: A sorter

Taking simple 2-bit comparators as our basic components we shall construct a pipeline of length 3 and demonstrate that it sorts four bits. We say a pipe sorts four bits input on the set of wires $\{i0, i1, i2, i3\}$ if the output on the wires $\{o0, o1, o2, o3\}$ has the same number of high wires as the corresponding input and if $j \in 0..3$ then

$$oj \text{ high} \Rightarrow \forall k \in 0..j \bullet ok \text{ high}$$

Data is latched along the pipeline by D-type flip-flops. The configuration of the circuit being shown in Figure 7.4, where *Comp* is a comparator and $\underline{D}$ is an array of four D-type flip-flops.

We shall consider the algebraic representation in SRPT of the basic components from which the above sorter is comprised. From these we shall derive the three components which make up the three phases of the pipeline. In a time frame in which one unit of time has the same length as one cycle of the system clock each of these three components is a pipe of length 1. Then, by application of the results of the previous section, we shall establish that the complete system is a pipe of length 3 which sorts four bits.

Comparator

A simple comparator can be constructed from an *AND* gate and an *OR* gate as shown in Figure 7.5. Using the definitions of *AND* and *OR* from Section 5.2.1 the comparator circuit can be described as follows:

$$Comp \triangleq OR \parallel AND[d/c].$$

It is a simple exercise to show that

$$Comp \equiv [!\{\}?X \rightarrow Comp(X)]$$

where

$$Comp(X) = Comp \textbf{ if } X = \{\} \textbf{ else } (Comp'' \textbf{ if } X = \{a, b\} \textbf{ else } Comp')$$
$$Comp' \equiv [!\{c\}?X \rightarrow Comp(X)]$$
$$Comp'' \equiv [!\{c, d\}?X \rightarrow Comp(X)]$$

Rising edge triggered D-type flip-flop

A D-type flip-flop is a 1-bit storage device. It latches the value on wire d at the time of the rising edge of the clock signal. This data is available as output until the flip-flop is reset at the next rising edge in the clock signal. Making the

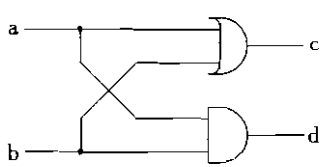


Figure 7.5: A simple comparator

same assumptions as were made when describing the clocked T-type flop-flop in Section 7.2.2, the following describes a D-type flip-flop.

$$\iota Dff = \{d, ck\} \qquad oDff = \{q\} \qquad Dff \triangleq Dff_L$$

where $Dff_L \triangleq [!\{\}?X \rightarrow (Dff_L \textbf{ if } ck \notin X \textbf{ else } (Dff'_H \textbf{ if } d \in X \textbf{ else } Dff_H))]$
$Dff_H \triangleq [!\{\}?X \rightarrow (Dff_H \textbf{ if } ck \in X \textbf{ else } Dff_L)]$
$Dff'_H \triangleq [!\{q\}?X \rightarrow (Dff'_H \textbf{ if } ck \in X \textbf{ else } Dff'_L)]$
$Dff'_L \triangleq [!\{q\}?X \rightarrow (Dff'_L \textbf{ if } ck \notin X \textbf{ else } (Dff'_H \textbf{ if } d \in X \textbf{ else } Dff_H))]$

Here L and H represent the value of the clock signal.

The clock

In this example we shall assume each of the three phases is controlled by a symmetric clock, described by the process:

$$CK \triangleq \mu P \bullet [!\{ck\} \rightarrow [!\{\} \rightarrow P]]$$

This clock is deterministic and has no inputs so its behaviour is governed precisely by time. We shall take advantage of this observation and absorb the clock into the implicit timing of the model. We shall ultimately model each phase of the pipeline in a time frame such that one time unit corresponds to a single clock cycle. Moreover, we shall assume the 'tick' in the model coincides with the rising edge of the clock signal.

The components of the pipeline

We want a representation of the three phases of the pipeline which corresponds to the system being viewed in a time frame where one time unit corresponds to a single clock cycle. We also require the representation to be in a form from which it is easy to deduce that each phase is a pipe. We shall only demonstrate the derivation of the component which makes up the first phase here. The algebraic representations for the other two phases are simply stated, their derivations being similar.

The first phase is built of four D-type flip-flops and two comparators:

$$\begin{aligned} Phase1 \triangleq (&Comp[t0/a, t1/b, a0/c, a1/d] \parallel Dff[i0/d, t0/q] \parallel Dff[i1/d, t1/q] \\ &\parallel Comp[t2/a, t3/b, a2/c, a3/d] \parallel Dff[i2/d, t2/q] \parallel Dff[i3/d, t3/q]) \\ &\backslash \{t0, t1, t2, t3\} \end{aligned}$$

this can be considered as two components in parallel:

$$Phase1 \triangleq Q[i0/d0, i1/d1, a0/c, a1/d] \parallel Q[i2/d0, i3/d1, a2/c, a3/d]$$

where $Q \triangleq (Dff[d0/d, a/q] \parallel Dff[d1/d, b/q] \parallel Comp) \backslash \{a, b\}$.

By eliminating parallel composition and hiding from the above expression we can demonstrate that:

$$Q \equiv S(L, \{\})$$

where S is given by:

$$S(L, Y) \triangleq [!g(Y)?X \rightarrow (S(L, Y) \textbf{ if } ck \notin X \textbf{ else } [!g(Y)?Z \rightarrow (S(H, X - \{ck\}) \textbf{ if } ck \in Z \textbf{ else } S(L, X - \{ck\}))])]$$
$$S(H, Y) \triangleq [!g(Y)?X \rightarrow (S(H, Y) \textbf{ if } ck \in X \textbf{ else } S(L, Y))]$$

where $g : \mathbb{P}\{d0, d1\} \rightarrow \mathbb{P}\{c, d\}$ is defined by:

$$\left.\begin{array}{ll} g(\{\}) = \{\} & g(\{d0\}) \\ g(\{d0, d1\}) = \{c, d\} & g(\{d1\}) \end{array}\right\} = \{c\}$$

The derivation is given in Appendix C.2.1.

The rising edge of the system clock coincides with the $2k + 1^{th}$ ticks of the time frame in which the system is modelled. We shall incorporate the details of the clock into the model of this phase of the pipeline and abstract away from the details of the clock.

$$Phs1 \triangleq (Phase1 \parallel CK) \setminus \{ck\}$$

Then we translate the model to a time frame in which a unit of time corresponds to a single cycle of the system clock and observations occur at the rising edge of the clock cycle. Thus we absorb the system clock into the implicit timing of the model.

$$Phase1' \triangleq Slow(2, 0, \{\}, Phs1)$$

By applications of the laws of SRPT we calculate an expansion of $Phs1$ which does not involve parallel composition. We then apply the laws for timewise abstraction (as shown in Appendix C.2.1) to demonstrate that

$$Phase1' \equiv P1(\{\})$$

where $P1$ is given in Figure 7.6. Clearly $Phase1'$ is a pipe of length 1 with effect given by $\mathcal{E}_{Phase1'}(X) = \{f_1(X)\}$.

Absorbing the system clock into the implicit timing of the model we obtain similar descriptions of the final two phases of the sorter. The pipes $Phase2'$ and $Phase3'$, both of length 1, which make up the final two phases of the pipeline are described as follows:

$$Phase2' \triangleq P2(\{\}) \qquad Phase3' \triangleq P3(\{\})$$

where $P2$ and $P3$ are given in Figure 7.7

Composing pipes

When we compose processes in SRPT we assume the components are modelled in the same time frame and the components in a parallel composition evolve in lockstep. We assume that all three components of the composed pipeline share the same time frame, and due to the way in which the components are modelled, the rising edges of the clocks controlling these three components coincide.

By applying the theory of Section 7.4.1 we know that

$$(Phase1' \gg Phase2') \gg Phase3'$$

is a pipe of length 3 with the same effect as the pipe of length 1 given by:

$$Slow(3, 0, \{\}, (Phase1' \gg Phase2') \gg Phase3')$$

Now by Theorem 7.5

$$\begin{aligned} &Slow(3, 0, \{\}, (Phase1' \gg Phase2') \gg Phase3') \\ &\quad \equiv Slow(2, 0, \{\}, Slow(2, 0, \{\}, Phase1' \gg Phase2') \gg Phase3') \end{aligned}$$

So we can evaluate an expansion for $Slow(2, 0, \{\}, Phase1' \gg Phase2')$. From this and the algebraic laws of SRPT (as demonstrated in Appendix C.2.2) we can deduce that

$$Slow(3, 0, \{\}, (Phase1' \gg Phase2') \gg Phase3') \equiv Sort(0)$$

$$P1(X) \mathrel{\widehat{=}} [!f_1(X)?Y \rightarrow P1(Y)]$$

where f_1 is defined over the domain $\mathbb{P}\{i0, i1, i2, i3\}$ as follows:

$$\left.\begin{array}{l} f_1(\{i0\}) \\ f_1(\{i1\}) \end{array}\right\} = \{a0\}$$

$$\left.\begin{array}{l} f_1(\{i2\}) \\ f_1(\{i3\}) \end{array}\right\} = \{a2\}$$

$$\left.\begin{array}{l} f_1(\{i0, i2\}) \\ f_1(\{i1, i2\}) \\ f_1(\{i0, i3\}) \\ f_1(\{i1, i3\}) \end{array}\right\} = \{a0, a2\}$$

$$f_1(\{i0, i1\}) = \{a0, a1\}$$

$$f_1(\{i2, i3\}) = \{a2, a3\}$$

$$\left.\begin{array}{l} f_1(\{i0, i1, i2\}) \\ f_1(\{i0, i1, i3\}) \end{array}\right\} = \{a0, a1, a2\}$$

$$\left.\begin{array}{l} f_1(\{i0, i2, i3\}) \\ f_1(\{i1, i2, i3\}) \end{array}\right\} = \{a0, a2, a3\}$$

$$f_1(\{i0, i1, i2, i3\}) = \{a0, a1, a2, a3\}$$

$$f_1(\{\}) = \{\}$$

Figure 7.6: The first phase of the sorter pipeline

where $Sort(0) \triangleq [!\{\}?X \rightarrow Sort(|X|)]$
$Sort(1) \triangleq [!\{o0\}?X \rightarrow Sort(|X|)]$
$Sort(2) \triangleq [!\{o0, o1\}?X \rightarrow Sort(|X|)]$
$Sort(3) \triangleq [!\{o0, o1, o2\}?X \rightarrow Sort(|X|)]$
$Sort(4) \triangleq [!\{o0, o1, o2, o3\}?X \rightarrow Sort(|X|)]$

Clearly this pipeline sorts 4 bits. Hence so too does the pipeline

$$(Phase1' \gg Phase2') \gg Phase3'$$

and we have the required result.

$P2(X) \triangleq [!f_2(X)?Y \rightarrow P2(Y)]$
$P3(X) \triangleq [!f_3(X)?Y \rightarrow P3(Y)]$

with f_2 and f_3 defined by

$$f_2(\{\})=\{\}$$
$$\left.\begin{array}{l} f_2(\{a0\}) \\ f_2(\{a2\}) \end{array}\right\}=\{b0\}$$
$$\left.\begin{array}{l} f_2(\{a1\}) \\ f_2(\{a3\}) \end{array}\right\}=\{b2\}$$
$$\left.\begin{array}{l} f_2(\{a0, a1\}) \\ f_2(\{a1, a2\}) \\ f_2(\{a0, a3\}) \\ f_2(\{a2, a3\}) \end{array}\right\}=\{b0, b2\}$$
$$f_2(\{a0, a2\})=\{b0, b1\}$$
$$f_2(\{a1, a3\})=\{b2, b3\}$$
$$\left.\begin{array}{l} f_2(\{a0, a1, a2\}) \\ f_2(\{a0, a2, a3\}) \end{array}\right\}=\{b0, b1, b2\}$$
$$\left.\begin{array}{l} f_1(\{a0, a1, a3\}) \\ f_1(\{a1, a2, a3\}) \end{array}\right\}=\{b0, b2, b3\}$$
$$f_2(\{a0, a1, a2, a3\})=\{b0, b1, b2, b3\}$$

$$f_3(\{\})=\{\}$$
$$f_3(\{b0\})=\{o0\}$$
$$\left.\begin{array}{l} f_3(\{b1\}) \\ f_3(\{b2\}) \end{array}\right\}=\{o1\}$$
$$f_3(\{b3\})=\{o3\}$$
$$\left.\begin{array}{l} f_3(\{b0, b1\}) \\ f_3(\{b0, b2\}) \end{array}\right\}=\{o0, o1\}$$
$$\left.\begin{array}{l} f_3(\{b1, b3\}) \\ f_3(\{b2, b3\}) \end{array}\right\}=\{o1, o3\}$$
$$f_3(\{b1, b2\})=\{o1, o2\}$$
$$f_3(\{b0, b3\})=\{o0, o3\}$$
$$f_3(\{b0, b1, b2\})=\{o0, o1, o2\}$$
$$f_3(\{b1, b2, b3\})=\{o1, o2, o3\}$$
$$\left.\begin{array}{l} f_3(\{b0, b1, b3\}) \\ f_3(\{b0, b2, b3\}) \end{array}\right\}=\{o0, o1, o3\}$$
$$f_3(\{b0, b1, b2, b3\})=\{o0, o1, o2, o3\}$$

Figure 7.7: The final two phases of the sorter pipeline

7.5 Conclusion

In this chapter we have introduced the notion of timewise abstraction in discrete time algebras. In frameworks where timewise abstraction can be applied unambiguously, such as SRPT, it provides a mechanism for translating processes to a less detailed time frame, that is one where observations are made less frequently. We have applied timewise abstraction in SRPT, using it to investigate the behaviour of components, initially described in a time frame appropriate to recording gate delays, when their input is latched at the slower speed governed by the system clock. We have also demonstrated how timewise abstraction may be used to evaluate the behaviour of pipes. The technique involves translating a process which represents a pipe of length ℓ to a time frame a factor of ℓ slower. The resultant process is a pipe of unit length with the same overall effect; the problem of verifying the behaviour of a unit length pipe is simpler than the same investigation of a longer pipe.

Chapter 8

Summary and Related Work

8.1 Summary

We have explored the result of adopting a synchronous view of concurrency, that is one in which components evolve in lockstep. The languages SCSP, a variant of the familiar CSP formalism, and SRPT, a synchronous version of Receptive Process Theory [Jos92], were introduced and given a denotational semantics. Despite variations in the way behaviours were described, the semantic models chosen for SCSP and SRPT both capture failure and divergence information concerning processes. We have formally demonstrated the way in which SRPT may be viewed as a sublanguage of SCSP by means of embeddings which map the language and model of the former into those of the latter. A syntactic extension to SCSP facilitated reasoning about the communication of data via channels. Finally, we introduced the theory of timewise abstraction, applying it to the language SRPT.

One interpretation of synchronous communication is that all components of a process running in parallel evolve together on the tick of a global clock. In this sense, both SCSP and SRPT are discrete time process algebras and are applicable to a domain of problems which includes systems with time-critical requirements. By working in a synchronous framework, we can express the relationship between timing of components and timing of the whole system. Interaction between components can affect the way in which they evolve, but not their speed. The result is a very simple model of timed behaviour.

As several components of a system evolve in absolute synchrony it is possible that the components may perform distinct events at the same time; yet in process algebras it is desirable to be able to eliminate the parallel composition operator from process expressions. Therefore, in both the languages presented in this thesis we allowed prefixing of processes with sets of events, the implication being that events drawn from such a set may occur concurrently. The languages thus support the notion of true concurrency in a very explicit manner.

The final significant feature of both the languages we have presented here is the strong causality assumption. We assume that time must pass between cause and effect; simultaneously occurring events do so independently of one another. Moreover, the occurrence of one event at a given time cannot preempt the occurrence of another event at that time. This results in the environmental choice being a choice as to which subset of the set of events, offered by the process at a given time, should be performed. This design assumption reflects the fact that in circuits, for example, the value on one wire at a given time cannot, in general, influence the value seen at the same time on an independent wire.

The receptive language, SRPT, contrasts with SCSP by dividing the alphabet of events associated with a process into input events and output events. The result is an implied direction of communication between components. Two processes P and Q cooperating on an event a, which is classed as an output event of P and an input event of Q, are seen as communicating by P sending Q a signal along channel a. By insisting that a process is always ready to perform any of its input events and output events occur as soon as they are made available, we have been able to capture the notion of receptiveness of processes in SRPT. Being both a synchronous and receptive theory makes SRPT highly appropriate as a theory for modelling and reasoning about synchronous circuits, where components are latched by a global clock and communication along wires between components is inherently receptive.

Both the languages SCSP and SRPT were given a denotational semantics. The semantic models chosen for the two languages capture the same behavioural information, namely occurrence and refusal of events and divergence. The way in which this information was captured differed in the two cases. In the model for SCSP, the refusal of a process to perform an event was considered as observable as the occurrence of an event. So refusal information was captured in the traces along with the events which occurred. By introducing the concept of infeasible behaviour into the traces a simple method was devised for encoding divergences into the traces. The non-determinism ordering, familiar to CSP, provided a simple ordering on this model. This allowed us to draw on a wealth of experience, accumulated in the development of semantics for CSP, to obtain comparable results in our model and a mathematical underpinning of the language SCSP.

The features of SRPT which characterise its receptive nature make refusal information directly deducible from the events performed. Input events are never refused, while all output events which could occur do, so those not seen to occur would be refused. This observation resulted in a model which did not record refusal information. In order to capture divergences within the traces once more, we chose to represent divergence by the absence of information; consequently an information ordering on the model became a natural choice in contrast to the usual non-determinism ordering.

As the languages have so many features in common, it seems appropriate that there should be a relationship hetween SCSP and SRPT. By considering a method for describing receptive systems in SCSP (assuming that refusal of events representing output results in divergence and making available all events representing input in all prefix constructs) it was possible to devise a natural embedding of SRPT into SCSP. This embedding preserves the intuitive representation of processes and results in a continuous map from the semantic model of SRPT to that of SCSP. So SRPT can be viewed as a receptive sublanguage of SCSP, as well as a synchronous variant of Receptive Process Theory.

We have shown in this thesis that both languages have sufficient algebraic laws for us to establish proof systems which are sound and complete with respect to the semantic models. The algebraic laws make it possible to eliminate parallel composition and hiding from process expressions; this allows us to derive representations of systems with all concurrency explicitly modelled in the prefix construct. Such representations give a clear perspective on the concurrency in the system. A complete proof system of algebraic laws enables refinement relationships deducible in the model to be established from algebraic manipulation of process expressions. As is shown in the examples throughout this thesis, this makes it unnecessary to consider the underlying semantics when reasoning about processes.

Adaptations were made to the language SCSP, in Chapter 4, which provide a framework in which to describe directional communication. The modifications to the language involved defining new notation in terms of existing syntax. This ensured the changes were well defined with respect to the model. The new notation made it easy to model the flow of data between components of a system along directional channels. The transfer of data between components is often a feature of complex systems whose development may benefit from the formal analysis offered by process algebras. With the enhanced notation, SCSP offers itself as a viable framework for such analysis.

Finally, we introduced the notion of timewise abstraction which provides a procedure for translating a process, in a discrete time algebra, to a slower time frame. Timewise abstraction was easy to define for SRPT, the interpretation being that input is held fixed for the length of the new longer clock cycle, changes to input and output are only recorded on the tick of the new clock. This interpretation was found to be particularly suited to the problem of modelling synchronous circuits; components could be developed at speeds appropriate to individual gate delays, then reinterpreted in a time frame corresponding to the speed at which input is latched. Timewise abstraction was also applied to aid the verification of pipelines modelled in SRPT.

Through the development of the languages SCSP and SRPT, we have shown how a synchronous view of communication may be adopted in process algebras. Within this thesis the lockstep progression of processes is used to capture discrete

time, enabling problems with quantitative timing details to be analysed with the languages presented here. We have striven for simplicity in both the design of the languages and the design of the models in this thesis. We have also demonstrated that the languages are applicable in diverse problem domains.

8.2 Comparisons

A variety of formalisms have been applied to the problem of specifying and verifying real-time systems. Originally, many of these formalisms took a qualitative approach to time, restricting their concern to the relative *ordering* of events within a system. More recently, reflecting the increased use of computer systems in time-critical applications, these formalisms have been extended to take a quantitative approach to time, encompassing the relative *timing* of events within a system. We shall briefly consider four approaches to formal analysis of real-time systems, and then compare and contrast the various features of other authors' approaches with those of the work presented in this thesis. Finally, we consider some of the formalisms which have been applied specifically to VLSI design.

Temporal Logics [Pnu77, MP92] are logical languages which include statements about variation in time, such as 'eventually' or 'until'. These allow qualitative timing conditions to be specified, for example, $p\,\mathcal{U}\,q$ is true in a given computation if p holds until q becomes true. The languages have been enhanced to allow statements which capture quantitative timing requirements both by allowing predicates that make statements about time in the language [PH88] and by annotating the modal operators (such as $\mathcal{U}$) with times [KdR85].

Graphical methods are exemplified by Petri nets [Rei85, Pet77] and Statecharts [Har87]. Petri nets are bipartite directed graphs consisting of a set of places (or conditions), a set of transitions (or events) and directed arcs from places to transitions and from transitions to places. A net is marked by tokens at places. A transition is enabled and may fire when all places with arcs to that transition (input places) are marked. When a transition is fired a token is removed from each of the input places and a token is added to each of the places reached by arcs from the transition. There are several variants of Petri nets, all based on the above scheme. Quantitative timing restraints have been added to transitions [MF76] by associating a time interval (t_l, t_g) with each transition: the transition must be enabled for a time t_l before it may fire and cannot be enabled for a time in excess of t_g without firing. Another approach to incorporating temporal details into Petri nets is presented in [CR85] where a minimum token holding time is associated with each place.

Statecharts is a visnal specificatiou language dealing with hierarchy, concurrency and communication. In its simplest form a Statechart is a labelled directed graph with nodes represeuting states and arcs labelled by events. Timed Statecharts [KP92] annotate each arc with a time interval (l, u) denoting lower and upper time bounds of the event in a manner comparable to [MF76].

Programming languages such as ESTEREL [BG92], SIGNAL [BlGSS92] and CSML [CLM91] have been applied to reactive and real-time systems. Both ESTEREL and SIGNAL are deterministic languages based on the synchrony hypothesis: this assumes that communication and elementary computation take no time. A program is conceptually executed on au infinitely fast machine; delays result only from iuteraction with the environment. ESTEREL is an imperative language which can be compiled into finite automata and used to program reactive kernels controlling the state of the system. SIGNAL is a relational style language suited to data flow analysis. CSML is a deterministic imperative language based on a weaker version of the synchrony hypothesis in which all reactions take oue clock cycle. Tbis makes it particularly suited to simulation of synchronous circuits.

Process Algebras provide the final class of formalisms to be considered here. Typically process algebras are given a structured operational semantics [Plo81] which takes the form of transition rules. These transition rules allow one to generate a tree representing the possible transitions of a process. Both specification and implementation of a system are represented by processes; an implementation is verified with respect to a specification by establishing that the two processes are equivalent under certain relations. Bisimnlations [Mil89] provide one class of such relations; these compare the behaviours as viewed by an external agent. Various bisimulations exist, some of which take into account internal actions or the passage of time (in real-time algebras) in their comparisons. Algebraic laws, representing rewrite rules which preserve bisimnlation, allow processes to be compared by means of algebraic manipulatiou. Alternatively, processes can be related under a testing equivalence [Hen88], whereby the success of experiments performed on processes is compared. Testing preorders can be established on an algebra and these allow comparison of processes.

The original process algebras CCS [Mil89] and ACP [BK84], and the ISO standard language LOTOS [BB88], which do not display any quantitative concept of timing, have been extended to record temporal information in a variety of ways. ACP_ρ [BB91a], the real-time extension of ACP, associates an absolute time stamp witb actions; while that associated with actions in TIC [QAF90], a timed calculus for LOTOS, is relative to the previous action. Timing is also assumed relative in the time stamping of actions witb intervals in Liang Chen's Timed CCS [Che91] and CCSiT [Dan92].

An alternative approach to time-stamping actions is the introduction of a time prefix as taken in versions of Timed CCS presented by Moller and Tofts [MT90] and Wang Yi [Yi91] and in the timed extensions of LOTOS, ρI [BL92] and Timed LOTOS [QF87]. In TPL [HR90] a distinguished action representing idling for one clock cycle is added to standard CCS.

Other timed algebras include ATP [NS90] which, like TPL, has a distinguished time action. PARTY [HSZFH92] includes two forms of time prefix in its language, a busy wait of fixed duration and an idle wait with arbitrary duration. Milner's SCCS [Mil83], MEIJE developed by Boudol [Bou85] and Jeffrey's Discrete Time CSP [Jef91a] introduce the notion of a clock tick in their action prefix constructs. In these languages the action prefix, like the set prefix of SCSP, takes one time unit to evolve.

An alternative approach to the use of operational semantics in the development of process algebras is used in this thesis. This approach, used by CSP [Hoa85] and many of its timed extensions, associates a denotational semantics with the algebra and gives the meaning of a process in terms of a mathematical model, the semantic model. Both specification and implementation can be represented by processes. The former can be shown to be refined by the latter using algebraic manipulation, aided by laws which preserve the (refinement) ordering on the semantics of expressions. Alternatively, specifications can take the form of predicates, representing semantic requirements, which can be shown, with the aid of a proof system to be satisfied by a process corresponding to the implementation. The latter approach is taken by Davies and Schneider [Dav91, Sch90] with Timed CSP.

Quantitative timing has been incorporated into extensions of CSP. Ortega-Mallén and Fruntos-Escrig assume that each action has an associated duration in their Timed Observations [OMdFE91], while the Timed CSP proposed by Reed and Roscoe [RR87] introduces a delay operator and associates a non-zero, minimum recovery time with the occurrence of actions in sequential processes. This minimum recovery time has been dropped in later versions of Timed CSP [DS92].

8.2.1 Features of formal methods for real-time systems

Time domain

Partially ordered time domains have been applied to process algebras by Baeten and Bergstra [BB90] and Jeffrey [Jef91b], but in general the time domain used in models of real-time systems is a totally ordered set such as $\mathbb{R}$ or $\mathbb{N}$. We shall limit our discussion to such domains. Totally ordered time domains can be divided into two classes; continuous — where typically $\mathbb{R}^+$ is used to represent time, and discrete — where for example $\mathbb{N}$ is used to model time.

Process algebras such as PARTY, and the timed extensions of CCS presented in

[MT90, Yi91, Che91] admit either continuous or discrete time. Timed CSP, ACP_ρ and CCSiT provide examples of formalisms where the time domain is continuous. On the other hand, in common with the languages SCSP and SRPT presented here, the measure of time is discrete in many formalisms including TPL, ATP, SCCS, MEIJE and the timed extensions of LOTOS in [QAF90, BL92].

Real-time process algebras using a continuous time domain provide a more realistic, and hence more complex, description of time than their discrete time counterparts. A continuous time domain gives rise to algebras which are very expressive, although the gain in expressibility is often off-set by the increase in complexity. Only the discrete time version of Temporal CCS [MT90] admits a complete axiomatisation. Similarly there is not an adequate axiomatisation of Timed CSP, although much work has been done to ease the use of Timed CSP as a specification language. In [Sch90], Schneider developed a compositional proof system for behavioural specifications in Timed CSP based on the language's semantics, while Jackson [Jac92] uses a language based on temporal logic to describe and verify programs in Timed CSP.

The model of concurrency

There are two recognised models of concurrency: interleaving concurrency and true concurrency. In the context of process algebras, interleaving concurrency cannot distinguish between a process which concurrently offers two independent events and a process which offers the choice between the sequential performance of two events in either order. This can be summarised by the existence of an equation equivalent to the CCS equation:

$$a|b = a.b + b.a$$

in the algebra. Such algebras allow complete elimination of concurrency from expressions.

In process algebras exhibiting true concurrency, the simultaneous occurrence of concurrent events can be distinguished from sequential occurrence of events. Concurrency cannot be completely eliminated from expressions in process algebras exhibiting true concurrency. Often the simultaneous occurrence of events is made explicit in the language: to this purpose [Jef91a] uses bags of events in Discrete Time CSP; the languages SCCS and MEIJE are built from monoids of indivisible actions; ACP represents concurrency of events explicitly as $a|b$; while the languages presented in this thesis use sets of events. Alternatively, true concurrency can be implicit in the inability to eliminate parallel composition from expressions, as is the case in the Timed CSP of [RR86]. In the Timed Observation semantics for CSP [OMdFE91] concurrency cannot be eliminated without extending the language to incorporate a bag prefix operator, which makes explicit the simultaneous occurrence of events.

Those real-time process algebras, such as SCSP and theories presented in [Mil83, RR86, Jef91a], which associate an inherent delay with sequential composition or action prefix necessarily exhibit true concurrency. It is interesting to note that the model of Timed CSP presented by [DS92], in which the delay, δ, associated with prefixing is removed, exhibits interleaving concurrency, like CSP. In general it appears that timed algebras which extend untimed models by time-stamping actions (eg. ACP_ρ or Liang Chen's TCCS [Che91]), or with a distinguished time action (eg. TPL), or by introducing a delay construct (eg. Temporal CCS [MT90]) exhibit the same form of concurrency as their untimed counterpart. In particular, ACP_ρ, like ACP, exhibits true concurrency: while ATP and Temporal CCS follow the interleaving concurrency approach of CCS.

True concurrency is also exhibited by graphical methods: Condition/Event systems in Petri Net theory [Thi86] define a transition as a set of events firing concurrently, while Statecharts allow simultaneous transitions via labelled arcs in concurrent components of a system. However the timed Petri net models [MF76, CR85] and Timed Statecharts [KP92] restrict themselves to a single transition at a time (involving a single event), removing the true concurrency aspect from these models.

Persistent and urgent actions

Persistency and urgency are attributes associated with actions in timed process algebras. An action is said to be persistent if, once it is offered by a process, the passage of time alone cannot result in the process withdrawing the offer of the action. In SCSP, the derived event prefix construct, $a \rightsquigarrow P$, allows us to model persistency. However, events offered in the primitive set prefix construct need not be persistent. For example, the process $[X \subseteq B \rightarrow P_X]$ may idle for one unit and evolve into a process $P_{\{\}}$ which is unable to offer those events in set B. In SCCS persistent actions can be represented in a similar manner to SCSP, by explicitly allowing the choice between performing the action and idling until the action occurs.

TPL and Timed CSP consider actions to be persistent in their prefix constructs, although the provision of a timeout construct allows the offer of events to be withdrawn. Wang Yi also uses persistent actions in the Timed CCS of [Yi91], where, by parameterising the process with time, the behaviour, subsequent to the performance of an action, may vary depending on the time at which the action was performed. In CCSiT and Laing Chen's Timed CCS, actions are made available over the duration of an interval; by allowing the interval to extend to infinity, actions can be made persistent. The extension of LOTOS in [BL92], pl, considers actions to be persistent unless explicitly marked as urgent.

An action is said to be urgent if it must be performed as soon as it is made available. In SRPT output events can be seen to be urgent; we assume that they can be performed when they are made available and make no provision for the failure of

occurrence of offered output events. In SCSP the only way to represent an urgent event is to assume that the result of idling is divergence, as in the following process

$$[\{a\} \rightarrow P \rhd \bot].$$

In many process algebras (eg. Temporal CCS [MT90], ATP and PARTY) the failure to perform an urgent event will result in timestop with the process unable to progress in time. In Temporal CCS and PARTY, action prefix is urgent, although the provision of an idle prefix δ enables a persistent action to be represented by $\delta.a.P$. Actions in ATP are also urgent; persistency of event a can be captured using the timeout construct $recX.[aP](X)$. In TIC and ACP_ρ all actions are time-stamped and urgent. The provision of a choice set of terms in TIC and an integral construct representing the choice of an action over a continuum of times in ACP_ρ allows some degree of freedom in interaction with the environment in these algebras. The Timed CCS of [Che91] and CCSiT can represent urgency in their actions by reducing the associated interval to a single instant. In the case of [Che91] the process cannot progress in time beyond the time interval until the action is performed, while in CCSiT time may progress beyond the time interval associated with the action but the process prefixed by the action cannot evolve. Finally, actions in SCCS can be considered urgent as a consequence of the way that parallel components proceed in absolute synchrony.

Communication

In the original process algebras, CCS, CSP and ACP, one assumes that concurrently running components in a system progress at arbitrary speeds. Synchronisation occurs whenever communication is required between two components. To achieve this may require one process to wait for the other; such waiting is not recorded explicitly. Communication is referred to as asynchronous [Mil83] in circumstances where it involves the arbitrary delay of components. Like the algebras mentioned above, some of the timed process algebras also take an asynchronous view of communication. With the introduction of timing, the delays are made explicit. However the delays before communication takes place can be arbitrary, as modelled in Timed CSP, TPL and Wang Yi's Timed CCS for example. A common feature of all these algebras is the persistent nature of actions and this allows such arbitrary waiting.

Communication is regarded as synchronous, in the sense of Milner, if communicating components proceed in lockstep, cooperation on an action being possible if the action is simultaneously made available by all components. This view of synchronous communication is taken in SCCS: components of a system proceed in absolute synchrony; a component's evolution may be governed by its interaction with the environment and other components but the speed of this evolution is independent of the component's interactions. The same view of synchronous

communication is taken in SCSP: set prefix represents an opportunity for the environment to control the evolution of a process and will evolve after one time unit regardless of the availability of events from the environment. In [Jef91a], Jeffrey uses the same technique to represent synchronous communication.

By using time-stamps on actions ACP_ρ, TIC, CCSiT and Liang Chen's Timed CCS exhibit a synchronous model of communication. Other algebras in which actions are urgent (eg. Temporal CCS [MT90], PARTY and ATP) also provide a synchronous view of communication, although that of ATP can also be viewed as following the synchrony hypothesis proposed by Berry [BB91b]. In this view time is only marked at the points at which environmental interaction takes place; the system is assumed sufficiently fast for all necessary internal interactions to be completed before further environmental interaction is attempted. So at the lower level of internal interaction, ATP behaves asynchronously, while synchronisation of components on time actions ensures that components progress synchronously with respect to the environment.

Most of the algebraic formalisms which present a synchronous view of communication provide an arbitrary wait construct, which allows local desynchronisation to be modelled via persistent actions. In contrast, parallel composition in MEIJE is asynchronous but synchronisation can be achieved by use of a ticking operation: this can be seen as marking an agent with an authorisation signal sent by a synchroniser.

Causality

One of the decisions made in designing the language SCSP was to insist that, if the observation of one event is dependent on the occurrence of another, then time must pass between these two events. That is, time must pass between cause and effect (time dependent causality). One way to examine whether such causality assumptions have been made is to consider whether the visible behaviours of a system can be completely described by a bag of time-stamped actions; if this is the case then the ordering of events occurring at a given instant is not significant. Those timed process algebras (eg. SCCS, Discrete Time CSP and the Timed CSP presented in [RR86]), which associate an inherent delay with sequential composition or action prefix, prohibit the simultaneous occurrence of causally related events. ACP_ρ ensures that time passes between cause and effect via its axioms. PARTY takes a novel view that $a\,;b$ cannot be distinguished from $b\,;a$ or $a|b$. (It assumes time passes between cause and effect, but does not reflect this in its sequential composition construct.)

In SCSP the causality requirement is slightly stronger than that in some of the other models, notably Discrete Time CSP and Timed CSP. In SCSP the performance of one event at a particular time cannot preempt another event at the

same time; this results in the inability to instantly resolve (or even model) external choice. Moreover, unlike the other algebras which exhibit time dependent causality, SCSP does not support *auto-concurrence*, the ability to perform multiple copies of an event at a single instant. This is consistent with the strong causality requirement and results in a model in which refusal information at a given time is not dependent on the events which occurred at that time.

The alternative approach to time dependent causality is to allow *instant causality*; the occurrence of an event a at time t may effect the occurrence of other events at time t. Instant causality is demonstrated by many algebras including Temporal CCS [MT90], TPL, ATP and the model of Timed CSP advocated in [DS92]. Instant causality in TPL is a result of the notion that timing constraints are not always explicit. Idling is only made explicit when it must occur; on other occasions timing considerations are left arbitrary and unrecorded, as in CCS. In ATP instant causality is due to a similar lack of concern as to temporal details between time actions. This view is shared by the languages ESTEREL and SIGNAL which, based on the synchrony hypothesis, assume that there is no delay between the receipt of input and the production of the consequent output; the only delays modelled in these languages are those resulting from awaiting environmental interaction.

Timing relations

In Chapter 7 of this thesis the notion of timewise abstraction was developed and applied to the language SRPT. This provides a mechanism for slowing down the time frame in which components of a system are modelled. Other authors have considered timing relations which may be incorporated into formal methods; some of these will be considered here.

Schneider [Sch90] introduced timewise refinement into Timed CSP. This formalised the concept of a simple process in CSP being refined by processes in Timed CSP which introduce timing considerations. This notion was complemented by a mapping on processes in Timed CSP which removes all explicit timing information, giving their untimed counterparts in CSP. The original timed process is a timewise refinement of its image. It is then possible to verify those properties of a system which are preserved by timewise refinement (eg. safety requirements) in the simpler untimed model.

By considering a subcalculus of Temporal CCS in which all actions are persistent, ℓTCCS, Moller and Tofts [MT91] are able to consider a 'faster than' relation on processes. A process P is faster that Q if it may perform actions sooner. By insisting that all actions are persistent, P is always capable of progressing at the same speed as Q and, after performing an action earlier than Q could, P is capable of idling to allow Q to 'catch up'. This allows comparison of processes which are behaviourally equivalent (in the untimed sense), but which operate at different

speeds, without loosing sight of all temporal considerations.

Daniels annotates events with intervals during which they may occur. This leads to the definition of a time based refinement relation $\gg$, in [Dan92]. This relation $S \gg R$ can be interpreted as meaning than R has a more precise timing specification that S. Hence S includes the same visible behaviour as R but intervals in R when actions are enabled are included in the corresponding intervals in S.

8.2.2 Formalisms for clocked circuit design

It has been shown in this thesis how SRPT may be employed in the verification of synchronous circuits. It is therefore appropriate to consider other approaches which have been applied in this area. Many of the approaches are mechanised to some extent, which is required to cope with the scale of realistic circuit design problems. The following approaches are just a representative selection and should not be considered exhaustive.

Algebraic descriptions of circuits are provided both by CIRCAL [Mil86] and HOP (Hardware viewed as Objects and Processes) [GMA89]. Both these algebras adopt the concept of lockstep synchronisation of concurrent components associated with SCCS. CIRCAL is viewed as giving a relative description of the occurrence of events, with the provision for modelling simultaneously occurring events. Actual timing is modelled by clocked components having a special timing port which receives tick events from an abstract timer also modelled in CIRCAL. Specification and implementation of circuits can be shown to be equivalent by mechanical algebraic manipulation in the CIRCAL system. HOP represents circuits as finite state transition systems. The overall behaviour of composed components is examined with the aid of the PARCOMP tool which automates process composition. Within the language, output is seen to occur at the same time as the input which caused it, giving a model comparable to synchronous languages like ESTEREL.

In the imperative programming language SML [CLM91], programs represent synchronous circuits, and their semantics are based on the hardware implementation of a state machine. In SML, control constructs determine the next state and are assumed to execute in zero time. Assignments change state and are assumed to take one clock cycle. Timing rules can prevent complicated relationships from being described without delaying more than one clock cycle. In order to remove excessive delays the language incorporates a 'compress' statement which assumes all assignments within its scope take place in a single clock cycle. This compress facility can be used to a similar effect as timewise abstraction in SRPT.

μFP [She86] and Ruby [JS90] adopt a functional approach to the design of circuits. Primitive circuit descriptions are composed using higher-order-functions (or relations) to give a description of the complete circuit. Circuits are developed by transforming a correct design to an implementable description using algebraic laws

to manipulate functional expressions. Functions which describe circuit behaviour take streams of inputs (over time) and produce streams of outputs, giving a discrete time model of circuit behaviour suitable for synchronous circuits. All reasoning is done at a functional level, the data streams are not made explicit unlike most methods. A functional approach has also been taken by [BT89] where formal verification of synchronous circuits using a string-functional semantics is mechanised using the Boyer-Moore theorem prover [BM79].

A variety of logics have been applied to circuit design. Higher-order logic is used by Gordon [Gor86] to specify and verify circuits. Here, devices are modelled as predicates on input and output; only when output matches input is the predicate true. In systems where values on wires vary over time the input and output are represented by functions from time to boolean values and predicates modelling components are also time dependent. The HOL theorem prover [Gor85] provides assistance in the verification of circuit descriptions. Linear time temporal logic is used in [FKTMO86] to capture formally timing requirements in circuits. Again this approach considers circuits as predicates on their input and output. A logic programming language, Tokio, is presented which allows computer aided verification.

8.3 Future work

We have already suggested that SCSP is a simple language with a miuimal number of operators. In Chapter 4 the language was enhanced by the provision of a mechanism for describing value passing in communication. It would be useful to consider a number of other extensions to the language. Some would be easy to implement, while others would require extension of the syntax and, in some cases, modification of the semantic model.

A rendezvous on set B, $B \rightsquigarrow P$, could be defined in terms of the existing language

$$\{\} \rightsquigarrow P \mathrel{\widehat{=}} P$$
$$B \rightsquigarrow P \mathrel{\widehat{=}} [X \subseteq B \rightarrow ((B - X) \rightsquigarrow P)] \qquad \text{if } B \neq \{\}.$$

This is a generalisation of the derived event prefix construct and only allows $B \rightsquigarrow P$ to evolve to P once all the events in B have occurred. It would be useful to establish circumstances in which such an operator might be useful.

Timeouts and timed interrupts are features that any model of real-time systems should be able to capture. SCSP captures timeout implicitly in the definition of set prefix. In order to make it easier to apply SCSP to the description of real-time systems it would be advantageous to extend the syntax of the language to incorporate an explicit timeout operator. One possibility is to define a timeout

$P \overset{(n,B)}{\triangleright} Q$ parameterised by both a time and a set of events. The interpretation of such an operator is that $P \cong P_1 \overset{(n,B)}{\triangleright} P_2$ behaves like P_1 for the first n units of time, then a timeout occurs and it behaves like P_2 if P had only performed events from set B prior to the timeout. The traditional timeout would correspond to $B = \{\}$ while a timed interrupt, comparable to that presented in [Sch90] for Timed CSP, would be represented by the case $B = \alpha P$. Clearly extending the syntax of the language must be accompanied by verification that the new terms are well defined with respect to the model. The proof system would also have to be modified to incorporate the new terms.

SCSP does not incorporate the notion of successful termination [Hoa85]: either a process is defined recursively so as not to terminate or it terminates in chaos. Incorporating successful termination into SCSP would allow us to model systems which are required to terminate in their normal behaviour. It would also provide a framework in which to define sequential composition of processes. If successful termination is to be represented in the language then it must be supported by the model of SCSP: this would involve modification of the semantic model, the effects of which must be incorporated into all results involving the model.

Currently zero delay gates cannot be modelled in SRPT. As output would necessarily be dependent on the simultaneously occurring input in such circumstances, it is contrary to the underlying language design assumptions to allow process representing zero delay gates. A possible solution to this problem is to simulate zero delay gates by functions which may be composed with processes. For example, consider a zero delay combinatorial circuit which preprocesses input to a larger circuit with delays. We could represent the combinatorial circuit by the function f and the remainder of the circuit by process P. Then the process $f \circ P$ could represent the required circuit, where '$\circ$' is a functional composition operator. Such an extension to SRPT may have applications in the study of synchronous circuits.

Another area of future development lies in the construction of operational semantics for both SCSP and SRPT. By developing a structured operational semantics in the style of Plotkin [Plo81] we would be in a position to make more thorough comparisons between our work and the process algebras of other authors presented with semantics in this form. An operational semantics is a prerequisite to the development of software tools. Tools such as FDR [For92], a refinement checker for CSP, provide mechanical techniques for comparison of process expressions; such mechanical assistance makes it feasible to consider problems substantially larger than would be practical by hand. Both SCSP and SRPT would benefit from the availability of such tools. As has already been suggested, the scale of circuit design problems means that for a formal development method to be applicable in practice it must be supported by software tools.

It may be argued that tbere are certain circumstances when specifications in terms of predicates on behaviours may be more appropriate than defining a specification in terms of a process. It is ofteu easier to formulate an abstract requirement using predicates; a one place buffer can be characterised by a simple relationship between input and ontput. This is easily captured by suitable predicates while the process specifying such a buffer would be the least deterministic process with the reqnired hehaviour. Predicate based specifications can also be more appropriate for the provision of partial requirements for a system. By using different techniques for representing specifications and implementations we could make a clear distinction between the two stages of development. Taking advantage of the denotational semantics of SCSP, a compositional proof system based on the quantification of predicates over behaviours could be developed. A possible approach is the development of a system using the **sat** notatiou employed by [Hoa85]. Alternatively, linear time temporal logics could be used to provide a basis for a specification language for SCSP. To consider the latter approach adequately would require ns to extend the model to an infinite traces model in which concepts used in temporal logics, such as 'eventually', could be specified.

Finally, as with all new formalisms, the langnages presented in this thesis would beuefit from the experience gained through applicatiou. It is only by using such formalisms as those developed here that we can really appreciate their worth: to this end the analysis of larger case studies using SCSP and SRPT would be appropriate.

Appendix A

Proofs of Stated Results

A.1 Results in the model for SCSP

Theorem A.1 *Assuming $\mathcal{T}[\![P]\!]\rho$ and $\mathcal{T}[\![Q]\!]\rho$ satisfy the closure conditions with respect to alphabets $\alpha[\![P]\!]$ and $\alpha[\![Q]\!]$ respectively. Then $\mathcal{T}[\![P \parallel Q]\!]\rho$ satisfies vi with respect to alphabet $\alpha[\![P \parallel Q]\!]$.*

Proof: As the variable bindings will remain unchanged throughout this proof we will not make them explicit in our argument. By the construction of $\mathcal{T}[\![P \parallel Q]\!]$, if s' in $\mathcal{T}[\![P \parallel Q]\!]$ there are two cases to consider when establishing the closure conditions are satisfied, the case where s' results from agreement of both processes for the whole time of observation;

$$s' \in \{s \mid \exists\, s_1, s_2 : \mathbb{P}(\bar{A} \cap \bar{B})^* \bullet s_1 \cap s_2 = \langle\{\}\rangle^{|s|} \wedge s \cap \dot{A} - s_1 \in \mathcal{T}[\![P]\!] \wedge s \cap \dot{B} - s_2 \in \mathcal{T}[\![Q]\!]\}.$$

and the case where s' is the result of divergence of one of the component processes.

$$s' \in \{s^\frown r \mid \exists\, s_1, s_2 : \mathbb{P}(\bar{A} \cap \bar{B})^* \bullet s_1 \cap s_2 = \langle\{\}\rangle^{|s|} \wedge r \neq \langle\rangle \wedge (((s \cap \dot{A} - s_1)^\frown \dot{A} \in \mathcal{T}[\![P]\!] \wedge s \cap \dot{B} - s_2 \in \mathcal{T}[\![Q]\!]) \vee (s \cap \dot{A} - s_1 \in \mathcal{T}[\![P]\!] \wedge (s \cap \dot{B} - s_1)^\frown \dot{B} \in \mathcal{T}[\![Q]\!]))\}.$$

Notice, we only need to consider situations where $r \neq \langle\rangle$ here, since, by closure condition ii, if $r = \langle\rangle$ then s' is included in the first case. Without loss of generality we shall assume that the divergence is caused by the divergence of P.

Suppose $s^\frown\langle C\rangle^\frown s' \in \mathcal{T}[\![P \parallel Q]\!]$

case 1: We can find $s_1^\frown\langle X_1\rangle^\frown s_1'$ and $s_2^\frown\langle X_2\rangle^\frown s_2'$ such that

$(s \cap \dot{A} - s_1)^\frown\langle C \cap \dot{A} - X_1\rangle^\frown(s' \cap \dot{A} - s_1') \in \mathcal{T}[\![P]\!]$
$\wedge\ (s \cap \dot{B} - s_2)^\frown\langle C \cap \dot{B} - X_2\rangle^\frown(s' \cap \dot{B} - s_2') \in \mathcal{T}[\![Q]\!]$

We shall assume that X_1 and X_2 are chosen to be minimal in the following sense:

$$\forall\, Y \subset X_1 \bullet (s \cap \grave{A} - s_1)^\frown\langle C \cap \grave{A} - Y\rangle^\frown(s' \cap \grave{A} - s_1') \notin \mathcal{T}[\![P]\!].$$

Set $D_1 = \{a \in A \mid a \notin C \wedge \bar{a} \notin C - X_1\}$
$D_2 = \{b \in B \mid b \notin C \wedge \bar{b} \notin C - X_2\}$

and $D = \{a \in A \cup B \mid a \notin C \wedge \bar{a} \notin C\}$

Now by condition vi on $\mathcal{T}[\![P]\!]$

$$(s \cap \grave{A} - s_1)^\frown\langle (C \cap \grave{A} - X_1) \cup D_1\rangle \in \mathcal{T}[\![P]\!]$$
$$\vee\, \exists x \in D_1 \bullet (s \cap \grave{A} - s_1)^\frown\langle (C \cap \grave{A} - X_1) \cup \{\bar{x}\}\rangle^\frown(s' \cap \grave{A} - s_1') \in \mathcal{T}[\![P]\!]$$

Now by the minimality of X_1, $\bar{x} \notin X_1$, so $\bar{x} \notin C$ and $x \in D$.

If the latter case holds then

$$\exists x \in D \bullet (s \cap \grave{A} - s_1)^\frown\langle (C \cup \{x\}) \cap \grave{A} - X_1\rangle^\frown(s' \cap \grave{A} - s_1') \in \mathcal{T}[\![P]\!]$$
$$\wedge\, (s \cap \grave{B} - s_2)^\frown\langle (C \cup \{\bar{x}\}) \cap \grave{B} - X_2'\rangle^\frown(s' \cap \grave{B} - s_2') \in \mathcal{T}[\![Q]\!]$$

where $X_2' = \begin{cases} X_2 & \text{if } x \in A - B \\ X_2 \cup \{\bar{x}\} & \text{if } x \in A \cap B \end{cases}$

Clearly X_1 and X_2' are disjoint so

$$\exists x \in D \bullet s^\frown\langle C \cup \{\bar{x}\}\rangle^\frown s' \in \mathcal{T}[\![P \parallel Q]\!]$$

A similar result is obtained by considering vi on $\mathcal{T}[\![Q]\!]$.

The remaining case is when

$$(s \cap \grave{A} - s_1)^\frown\langle (C \cap \grave{A} - X_1) \cup D_1\rangle \in \mathcal{T}[\![P]\!]$$
$$\wedge\, (s \cap \grave{B} - s_2)^\frown\langle (C \cap \grave{B} - X_2) \cup D_2\rangle \in \mathcal{T}[\![Q]\!]$$

Now $D \cap \grave{A} \subseteq D_1$ and $D \cap \grave{B} \subseteq D_2$ hence by condition iii.

$$(s \cap \grave{A} - s_1)^\frown\langle ((C \cup D) \cap \grave{A} - X_1)\rangle \in \mathcal{T}[\![P]\!]$$
$$\wedge\, (s \cap \grave{B} - s_2)^\frown\langle ((C \cup D) \cap \grave{B} - X_2)\rangle \in \mathcal{T}[\![Q]\!]$$

Thus $s^\frown\langle C \cup D\rangle \in \mathcal{T}[\![P \parallel Q]\!]$ as required.

case 2: If divergence occurs for trace $r < s^\frown\langle C\rangle^\frown s'$ then either $r \leq s$ in which case the result follows by construction, otherwise for some $r' < s'$

$$((s^\frown\langle C\rangle^\frown r') \cap \grave{A} - s_1)^\frown\langle\bar{A}\rangle \in \mathcal{T}[\![P]\!] \wedge (s^\frown\langle C\rangle^\frown r') \cap \grave{B} - s_2 \in \mathcal{T}[\![Q]\!]$$

in which case everything follows as for case 1 to give condition vi. □

Theorem A.2 *Hiding is continuous with respect to the partial order.*

Proof: We must show

$$\mathcal{T}[\![x \setminus A']\!]\rho[\sqcup D/x] = \bigcap_{d \in D} \mathcal{T}[\![x \setminus A']\!]\rho[d/x]$$

where D is a directed set in $(SM^A, \sqsubseteq)$.

Now $\mathcal{T}[\![x]\!]\rho = \pi_2 \rho[\![x]\!]$. As the projection π_2 is a continuous function, we deduce that $\{\pi_2 d \mid d \in D\}$ forms a directed set in $(SM^A_T, \subseteq)$ and $\pi_2(\sqcup D) = \bigcap_{d \in D} \pi_2 d$. Now

$$\begin{aligned}
&\mathcal{T}[\![x \setminus A']\!]\rho[\sqcup D/x] \\
&\quad = \{s \mid \exists\, s' \bullet s = s' - \grave{A}' \wedge s' \in \textstyle\bigcap_{d \in D} \pi_2 d \wedge \mathit{saturated}_{A' \cap A}(s')\} \\
&\bigcap_{d \in D} \mathcal{T}[\![x \setminus A']\!]\rho[d/x] \\
&\quad = \textstyle\bigcap_{d \in D} \{s \mid \exists\, s' \bullet s = s' - \grave{A}' \wedge s' \in \pi_2 d \wedge \mathit{saturated}_{A' \cap A}(s')\}
\end{aligned}$$

Clearly $\mathcal{T}[\![x \setminus A']\!]\rho[\sqcup D/x] \subseteq \bigcap_{d \in D} \mathcal{T}[\![x \setminus A']\!]\rho[d/x]$. We must show the converse.

Suppose $s \in \bigcap_{d \in D} \mathcal{T}[\![x \setminus A']\!]\rho[d/x]$, then the number of ways of saturating s is finite. Let J be a finite indexing set such that $\{s_j \mid j \in J\}$ is the set of all possible saturations. Assume $s \notin \mathcal{T}[\![x \setminus A']\!]\rho[\sqcup D/x]$ Then there is no s_j such that $s_j \in \bigcap_{d \in D} \pi_2 d$. For each saturation s_j we can find $d_j \in D$ such that $s_j \notin \pi_2 d_j$ Then by the property of directed sets and since J is finite, we can find k such that $\forall j \in J \bullet \pi_2 d_k \subseteq \pi_2 d_j$. So $\forall j \in J \bullet s_j \notin \pi_2 d_k \Rightarrow s \notin \mathcal{T}[\![x \setminus A']\!]\rho[d_k/x] \Rightarrow s \notin \bigcap_{d \in D} \mathcal{T}[\![x \setminus A']\!]\rho[d/x]$. Hence result by contradiction. □

A.2 Results in the model for SRPT

Theorem A.3 *The operators of SPRT are monotonic in each argument.*

Proof: For environment σ and P a process term which takes the form of an application of an operator of SPRT on process terms, such that one of the arguments of the operator is the process variable x and all other arguments are independent of x. We must show $\lambda\, y \bullet \mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma[y/x]$ is monotonic.

We shall consider the structure of the trace sets of the operators.

For each P of the above form and $q \in RM$, $\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x]$ takes the form

$$\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x] = T'(\pi_3 q) \downarrow M'(\widehat{\pi_3 q})$$

for example, if $P \triangleq Q \sqcap x$ and Q is independent of x then:

$$\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x] = (\mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma \cup \pi_{\mathcal{S}} q) \downarrow (\hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \cup \widehat{\pi_{\mathcal{S}} q})$$

$T'(\pi_{\mathcal{S}} q)$ is a trace set with contribution $T'(\{s\}) - T'(\{\})$ due to each trace $s \in \pi_{\mathcal{S}}$. The contribution due to a given trace $s \in \pi_{\mathcal{S}} q$ is independent of the structure of q. Hence

$$q \leqslant q' \Rightarrow T'(\pi_{\mathcal{S}} q) \subseteq T'(\pi_{\mathcal{S}} q')$$

$M'(\widehat{\pi_{\mathcal{S}} q})$ is a set whos members are governed by the traces in $\widehat{\pi_{\mathcal{S}} q}$ such that

$$q \leqslant q' \Rightarrow T \downarrow M'(\widehat{\pi_{\mathcal{S}} q}) \subseteq T \downarrow M'(\widehat{\pi_{\mathcal{S}} q'}) \qquad \text{for any trace set } T.$$

Hence by the transitivity of $\subseteq$: $\quad q \leqslant q' \Rightarrow \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x] \subseteq \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q'/x]$

We also observe that

1. every maximal element of $\pi_{\mathcal{S}} q$ only contributes to maximal traces in $T'(\pi_{\mathcal{S}} q)$.
2. by the nature of restriction, if $s \in T'(\pi_{\mathcal{S}} q)$ is not present in $\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x]$ then there is a maximal element $s' \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x]$ with $s' < s$.
3. if $s' < s \in \pi_{\mathcal{S}} q$ and $r \in T'(\pi_{\mathcal{S}} q)$ is a trace contributed by s then there is a prefix of r in $T'(\pi_{\mathcal{S}} q)$ which is a contribution from s'.

We uow have sufficient information to show that, if $q \leqslant q'$ then

$$s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q'/x] \wedge s \notin \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma[q/x] \Rightarrow \exists r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma[q/x] \bullet r < s$$

To show the above it is sufficient to note that, if $s \in T'(\pi_{\mathcal{S}} q)$ then the result follows from observation 2. Otherwise s must be a contribution from $s_1 \in (\pi_{\mathcal{S}} q') - (\pi_{\mathcal{S}} q)$, then as $q \leqslant q'$ we can find $r_1 \in \widehat{\pi_{\mathcal{S}} q}$ with $r_1 < s_1$, by observations 1 and 3 there is a maximal contribution r to $T'(\pi_{\mathcal{S}} q)$ with $r < s$. The result follows. □

A.3 Results relating SRPT to SCSP

Theorem A.4 *For all processes $P, Q \in SRPT^{\emptyset}$ with $P \sqcap Q$ well defined, and for all $\sigma \in BIND_R$ if $\mathcal{T}[\![\Theta P]\!]\eta\sigma = \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma)$ and $\mathcal{T}[\![\Theta Q]\!]\eta\sigma = \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma)$ then*

$$\mathcal{T}[\![\Theta(P \sqcap Q)]\!]\eta\sigma = \phi(\mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma)$$

Proof: Firstly note that:

$$\begin{array}{lll}\mathcal{T}[\![\Theta(P \sqcap Q)]\!]\eta\sigma & = \mathcal{T}[\![(\Theta P) \sqcap (\Theta Q)]\!]\eta\sigma & \{\text{ defn. of } \Theta \ \} \\ & = \mathcal{T}[\![\Theta P]\!]\eta\sigma \cup \mathcal{T}[\![\Theta Q]\!]\eta\sigma & \{\text{ defn. of } \mathcal{T} \ \} \\ & = \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \cup \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma) & \{\text{ by hypothesis } \}\end{array}$$

So it is sufficient to show that for all $\sigma \in BIND_R$

$$\phi(\mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma) = \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \cup \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma)$$

Since the alphabets remain unchanged throughout this proof, we shall abuse notation slightly and write $\psi(s)$ for $\psi(I, O, s)$.

Recall $\phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) = \bigcup_{s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma} \psi(s) \cup \bigcup_{s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma} \psi_1(s)$

Now

$$\begin{array}{ll} & s \in \mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \\ \Rightarrow & s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \vee s \in \mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma & \{\text{ defn. of } \mathcal{T}_{\mathcal{R}} \ \} \\ \Rightarrow & s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \vee s \in \mathcal{T}_{\mathcal{R}}[\![Q]\!]\sigma & \{\text{ defn. of restriction } \} \\ \Rightarrow & \psi(s) \subseteq \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \cup \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma) & \{\text{ defn. of } \phi \ \}\end{array}$$

Similarly $s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \Rightarrow \psi_1(s) \subseteq \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \cup \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma)$

Hence $\phi(\mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma) \subseteq \phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \cup \phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma)$

It remains to prove the reverse inclusion.

$$\begin{array}{ll} & s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \\ \Rightarrow & \{\text{ logic, law of excluded middle } \} \\ & s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \wedge ((\exists s' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \bullet s' < s) \vee \neg (\exists s' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \bullet s' < s)) \\ \Rightarrow & \{\text{ definition of restriction } \} \\ & s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \vee (s \in \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \wedge \exists s' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \bullet s' < s) \\ \Rightarrow & \{\text{ definition of } \mathcal{T}_{\mathcal{R}} \text{ and as } s' \text{ is non maximal in } \mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma \ \} \\ & s \in \mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \vee ((\exists s' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \bullet s' < s) \wedge \neg (\exists s'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma \bullet s'' < s')) \\ \Rightarrow & \{\text{ definition of restriction } \} \\ & s \in \mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \vee (\exists s' \in (\hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!]\sigma \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma) \bullet s' < s) \\ \Rightarrow & \{\text{ definition of } \hat{\mathcal{T}}_{\mathcal{R}} \ \} \\ & s \in \mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \vee (\exists s' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma \bullet s' < s) \\ \Rightarrow & \{\text{ definition of } \phi \ \} \\ & \psi(s) \subseteq \phi(\mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma) \vee (\exists s' \bullet s' < s \wedge \psi_1(s') \subseteq \phi(\hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma)) \\ \Rightarrow & \{\text{ by Lemma 6.3 } \} \\ & \psi(s) \subseteq \phi(\mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma)\end{array}$$

Similarly $s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]\sigma \Rightarrow \psi_1(s) \subseteq \phi(\mathcal{T}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma)$

Hence $\phi(\mathcal{M}_{\mathcal{R}}[\![P]\!]\sigma) \subseteq \phi(\mathcal{M}_{\mathcal{R}}[\![P \sqcap Q]\!]\sigma)$

We can obtain a symmetric result for $\phi(\mathcal{M}_{\mathcal{R}}[\![Q]\!]\sigma)$ and thus deduce the required result. □

A.4 Results involving timewise abstraction

Theorem A.5 *Assuming $\mathcal{T}_{\mathcal{R}}[\![P]\!]\sigma$ satisfies the closure conditions for model RM with respect to alphabets $\iota[\![P]\!]\sigma$ and $o[\![P]\!]\sigma$, then for $m \in \mathbb{N}$, $n \in \mathbb{N}^+$ and $C \in \mathbb{F}\Sigma$, $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]\sigma$ satisfies closure condition III with respect to alphabets $\iota[\![P]\!]\sigma$ and $o[\![P]\!]\sigma$.*

Proof: As the variable bindings, σ, will remain unchanged throughout this proof we will not make them explicit here. By the remarks of note 2 on page 132, if $C \not\subseteq \iota[\![P]\!]$ the result follows trivially so we shall only consider the case $C \subseteq \iota[\![P]\!]$. Assuming that $s^\frown\langle X\rangle \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]$ and $Y \subseteq \iota[\![P]\!]$ we must show that:

$$s^\frown\langle (X \cap o[\![P]\!]) \cup Y\rangle \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]$$

Now writing I for $\iota[\![P]\!]$ and O for $o[\![P]\!]$:

$$\begin{array}{ll}
 & s^\frown\langle X\rangle \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!] \\
\Rightarrow & \{\text{ by definition of } \mathcal{T}_{\mathcal{R}} \} \\
 & (\exists\, r \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \bullet \mathit{choose}(n, m, r) = s^\frown\langle X\rangle \\
 & \qquad \wedge\ r \cap I \leq \langle C\rangle^m{}^\frown(n \otimes ((s^\frown\langle X\rangle) \cap I))) \\
 & \wedge \neg\, (r' \in \widehat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet \mathit{choose}(n, m, r') < s^\frown\langle X\rangle \\
 & \qquad \wedge\ r' \cap I \leq \langle C\rangle^m{}^\frown(n \otimes ((s^\frown\langle X\rangle) \cap I))) \\
\Rightarrow & \{\text{ by condition II on } \mathcal{T}_{\mathcal{R}}[\![P]\!], \text{ where } r_0{}^\frown\langle X\rangle \leq r \} \\
 & (\exists\, r_0 \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \bullet r_0{}^\frown\langle X\rangle \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \wedge |r_0| = n.|s| + m \\
 & \qquad \wedge\ \mathit{choose}(n, m, r_0) = s \\
 & \qquad \wedge\ (r_0{}^\frown\langle X\rangle) \cap I \leq \langle C\rangle^m{}^\frown(n \otimes ((s^\frown\langle X\rangle) \cap I))) \\
 & \wedge \neg\, (r' \in \widehat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet \mathit{choose}(n, m, r') \leq s \\
 & \qquad \wedge\ r' \cap I \leq \langle C\rangle^m{}^\frown(n \otimes (s \cap I))) \\
\Rightarrow & \{\text{ by condition III on } \mathcal{T}_{\mathcal{R}}[\![P]\!] \text{ and considering lengths } \} \\
 & (\exists\, r_0 \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \bullet r_0{}^\frown\langle (X \cap O) \cup Y\rangle \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \\
 & \quad \wedge\ \mathit{choose}(n, m, (r_0{}^\frown\langle (X \cap O) \cup Y\rangle)) = s^\frown\langle (X \cap O) \cup Y\rangle \\
 & \quad \wedge\ (r_0{}^\frown\langle (X \cap O) \cup Y\rangle) \cap I \leq \langle C\rangle^m{}^\frown(n \otimes ((s^\frown\langle (X \cap O) \cup Y\rangle) \cap I))) \\
 & \wedge \neg\, (r' \in \widehat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet \mathit{choose}(n, m, r') < s^\frown\langle (X \cap O) \cup Y\rangle \\
 & \qquad \wedge\ (r' \cap I \leq \langle C\rangle^m{}^\frown(n \otimes ((s^\frown\langle (X \cap O) \cup Y\rangle) \cap I)))) \\
\Rightarrow & \{\text{ by defintion of } \mathcal{T}_{\mathcal{R}} \} \\
 & s^\frown\langle (X \cap O) \cup Y\rangle \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n, m, C, P)]\!]
\end{array}$$

□

Lemma A.6 *The maximal set of* $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!]$ *is given by:*

$$\begin{aligned}\hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] = \{s \mid \exists\, r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet\ & choose(n,m,r) = s \\ & \wedge\ r \cap I \leq \langle C\rangle^m{}^\frown(n \otimes (s \cap I)) \\ & \neg\ (\exists\, r' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet\ choose(n,m,r') < s \\ & \wedge\ r' \cap I \leq \langle C\rangle^m{}^\frown(n \otimes (s \cap I)))\}\end{aligned}$$

where $I = \iota[\![P]\!]$. *Moreover,*

$$\begin{aligned}\exists\, u \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u < s \Leftrightarrow \exists\, r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet\ & choose(n,m,r) < s \\ & \wedge\ r \cap I < \langle C\rangle^m{}^\frown(n \otimes (s \cap I))\end{aligned}$$

Proof: The first part follows from the definition of $\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!]$

It is trivial from the definition of $\hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!]$ that

$$\begin{aligned}\exists\, u \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u < s \Rightarrow \exists\, r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet\ & choose(n,m,r) < s \\ & \wedge\ r \cap I < \langle C\rangle^m{}^\frown(n \otimes (s \cap I))\end{aligned}$$

To prove the remainder, suppose $\exists\, r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet choose(n,m,r) < s \wedge r \cap I < \langle C\rangle^m{}^\frown(n \otimes (s \cap I))$.

Now consider the set

$$\{r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \mid choose(n,m,r) < s \wedge r \cap I < \langle C\rangle^m{}^\frown(n \otimes (s \cap I))\}$$

This set is non-empty, by our assumption, and finite, since the output alphabet is finite and s is of fixed finite length. We take the minimum of this set, r_0, and choose s_0 to be the prefix of s of length $\lceil(|r_0| - m)/n\rceil$. Then

$r_0 \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \wedge choose(n,m,r_0) = s_0 \wedge r_0 \cap I < \langle C\rangle^m{}^\frown(n \otimes (s_0 \cap I)) \wedge s_0 < s$.

Moreover, since we choose r_0 to be minimal

$\neg\ (\exists\, r' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet choose(n,m,r') < s_0 \wedge r' \cap I < \langle C\rangle^m{}^\frown(n \otimes (s_0 \cap I))$.

Thus $\exists\, u \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u < s$ as required. □

Corollary A.7

$$\begin{aligned}s \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \Leftrightarrow \exists\, r \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \bullet\ & choose(n,m,r) = s \\ & \wedge\ r \cap I \leq \langle C\rangle^m{}^\frown(n \otimes (s \cap I)) \\ & \wedge\ \neg\ (\exists\, u \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u < s).\end{aligned}$$

■

Theorem A.8 *For* $m \in \mathbb{N}$, $n \in \mathbb{N}^+$, $C \subseteq \iota[\![P]\!]\sigma$ *and* $P \sqcap Q$ *well defined, then*

$$\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P \sqcap Q)]\!]\sigma = \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P) \sqcap \mathcal{S}low(n,m,C,Q)]\!]\sigma$$

Proof: As the variable bindings, σ, will remain unchanged throughout this proof we will not make them explicit here.

Recalling the definition of $\mathcal{T}_{\mathcal{R}}$ it is sufficient to show

$$\begin{aligned}\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P \sqcap Q)]\!] = {} & \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \\ & \cup \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!]\end{aligned}$$

Now

$$\begin{aligned}
& s \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P \sqcap Q)]\!] \\
\Rightarrow \quad & \{ \text{ by definition of } \mathcal{T}_{\mathcal{R}} \ \} \\
& \exists\, r \in (\mathcal{T}_{\mathcal{R}}[\![P]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] \cup \mathcal{T}_{\mathcal{R}}[\![Q]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]) \bullet \\
& \qquad choose(n,m,r) = s \wedge r \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \wedge \neg\ \exists\, r' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!] \bullet \\
& \qquad choose(n,m,r') < s \wedge r' \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
\Rightarrow \quad & \{ \text{ since } r \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \cup \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] \Rightarrow \exists\, s \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P \sqcap Q]\!] \bullet s \leq r \ \} \\
& \exists\, r \in (\mathcal{T}_{\mathcal{R}}[\![P]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] \cup \mathcal{T}_{\mathcal{R}}[\![Q]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]) \bullet \\
& \qquad choose(n,m,r) = s \wedge r \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \wedge \neg\ \exists\, r' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!] \bullet \\
& \qquad choose(n,m,r') < s \wedge r' \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \wedge \neg\ \exists\, r'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] \bullet \\
& \qquad choose(n,m,r'') < s \wedge r'' \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
\Rightarrow \quad & \{ \text{ by Lemma A.6 } \} \\
& \exists\, r \in (\mathcal{T}_{\mathcal{R}}[\![P]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![Q]\!] \cup \mathcal{T}_{\mathcal{R}}[\![Q]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![P]\!]) \bullet \\
& \qquad choose(n,m,r) = s \wedge r \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \qquad \wedge \neg\ \exists\, u' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u' < s \\
& \qquad \wedge \neg\ \exists\, u'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \bullet u'' < s \\
\Rightarrow \quad & \{ \text{ by definition of restriction } \} \\
& (\exists\, r \in \mathcal{T}_{\mathcal{R}}[\![P]\!] \bullet choose(n,m,r) = s \wedge r \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \qquad \wedge (\neg\ \exists\, u' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u' < s) \\
& \qquad \wedge (\neg\ \exists\, u'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \bullet u'' < s)) \\
& \vee (\exists\, r \in \mathcal{T}_{\mathcal{R}}[\![Q]\!] \bullet choose(n,m,r) = s \wedge r \cap I \leq \langle C \rangle^m (n \otimes (s \cap I)) \\
& \qquad \wedge (\neg\ \exists\, u' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u' < s) \\
& \qquad \wedge (\neg\ \exists\, u'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \bullet u'' < s)) \\
\Rightarrow \quad & \{ \text{ by previous corollary } \} \\
& (s \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \wedge (\neg\ \exists\, u'' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \bullet u'' < s)) \\
& \vee (s \in \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \wedge (\neg\ \exists\, u' \in \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \bullet u' < s)) \\
\Rightarrow \quad & \{ \text{ by definition of restriction } \} \\
& s \in (\mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \\
& \cup \mathcal{T}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,Q)]\!] \downarrow \hat{\mathcal{T}}_{\mathcal{R}}[\![\mathcal{S}low(n,m,C,P)]\!])
\end{aligned}$$

The reverse follows similarly. □

Appendix B

Proof Rules

B.1 Proof system for SCSP^I

Here we present the proof system for the language SCSP^I, the language of finite closed terms from SCSP.

Axioms for non-deterministic choice:

A–1	$\vdash P \sqcap Q \equiv Q \sqcap P$
A–2	$\vdash (P \sqcap Q) \sqcap R \equiv P \sqcap (Q \sqcap R)$
A–3	$\vdash P \sqcap P \equiv P$
A–4	$\vdash P \sqcap \bot \equiv \bot$

Axiom for set prefix:

A–5	$C \subseteq B \quad \vdash \quad [X \subseteq B \to P_X] \sqcap [Y \subseteq C \to Q_Y] \equiv [X \subseteq B \to R_X] \sqcap [Y \subseteq C \to Q_Y]$ where $R_{B'} \triangleq \begin{cases} P_{B'} \sqcap Q_{B'} & \text{if } B' \subseteq C \\ P_{B'} & \text{if } B' \not\subseteq C \end{cases}$

Axioms for parallel composition:

A–6	$\vdash \bot_A \parallel P \equiv \bot_{A \cup \alpha P}$
A–7	$\vdash P \parallel \bot_A \equiv \bot_{A \cup \alpha P}$
A–8	$\vdash (P \sqcap Q) \parallel R \equiv (P \parallel R) \sqcap (Q \parallel R)$
A–9	$\vdash (P \parallel (Q \sqcap R) \equiv (P \parallel Q) \sqcap (P \parallel R)$
A–10	$\vdash [X \subseteq A' \to P_X] \parallel [Y \subseteq B' \to Q_Y] \equiv [Z \subseteq (A' \cap B') \cup (A' - \alpha Q_{B'}) \cup (B' - \alpha P_{A'}) \to P_{Z \cap A'} \parallel Q_{Z \cap B'}]$

Axioms for hiding:

A–11	$\vdash \bot_A \setminus B \equiv \bot_{A-B}$
A–12	$\vdash (P \sqcap Q) \setminus A \equiv (P \setminus A) \sqcap (Q \setminus A)$
A–13	$\vdash [X \subseteq B \to P_X] \setminus A \equiv [Y \subseteq (B - A) \to (P_{Y \cup (B \cap A)} \setminus A)]$

Axioms for renaming:

A–14	$\vdash \bot_A [S] \equiv \bot_{A[S]}$
A–15	$\vdash (P \sqcap Q)[S] \equiv P[S] \sqcap Q[S]$
A–16	$\vdash [X \subseteq B \to P_X][S] \equiv [X \subseteq B[S] \to P_{X[S^{-1}]}[S]]$

Ordering rules:

O–1	$\vdash P \sqcap Q \sqsubseteq P$	**O–3**	$\dfrac{P \equiv Q}{P \sqsubseteq Q \sqsubseteq P}$
O–2	$\dfrac{P \sqsubseteq Q \sqsubseteq P}{P \equiv Q}$	**O–4**	$\dfrac{P \sqsubseteq Q \sqsubseteq R}{P \sqsubseteq R}$

Monotonicity rules:

M–1	$\dfrac{P_1 \sqsubseteq P_2 \wedge Q_1 \sqsubseteq Q_2}{P_1 \sqcap Q_1 \sqsubseteq P_2 \sqcap Q_2}$
M–2	$\dfrac{\forall X \subseteq B \bullet P_X \sqsubseteq Q_X}{[X \subseteq B \to P_X] \sqsubseteq [X \subseteq B \to Q_X]}$

B.2 Proof system for SCSP

The proof system for the closed terms of SCSP consists of all the rules in the previous section, with the following additions.

Axioms for recursion:

A–17	$\vdash_R P[(\mu x \bullet P)/x] \equiv \mu x \bullet P$
A–18	$\vdash_R P_j[\langle x_i \mathrel{\hat{=}} P_i \rangle_k / x_k] \equiv \langle x_i \mathrel{\hat{=}} P_i \rangle_j$

Least fixed point rule:

R–1	$\dfrac{\forall Q \in FIN(P) \bullet Q \sqsubseteq R}{P \sqsubseteq R}$

B.3 Derivations in the proof system for SCSP

Theorem B.1 $\quad \vdash P \parallel Q \equiv Q \parallel P$

Proof: By considering the characterisation of infinite processes by their finite syntactic approximations and recalling all finite processes can be expressed in normal form (Corollary 3.15) it is sufficient to assume both P and Q are in normal form. We define a rank function d on processes in normal form:

$$\begin{array}{rcll} d(\bot) & = & 0 & \\ d(\bigsqcap_{B\in\mathcal{B}} P_B) & = & \Sigma_{B\in\mathcal{B}} d(P_B) + |\mathcal{B}| - 1 & \text{if } |\mathcal{B}| \geq 1 \\ d([X \subseteq A \rightarrow P_X]) & = & \max_{X\subseteq A} d(P_X) + 1 & \end{array}$$

and proceed by induction on $d(P) + d(Q)$.

base case: $d(P) + d(Q) = 0$.

Here $P = Q = \bot$ and the result follows from A–6.

inductive step:

If $P = \bot$ or $Q = \bot$ then the result follows from A–6 and A–7.

If $P = P_1 \sqcap P_2$ then $d(P_1) < d(P)$ and $d(P_2) < d(P)$ so:

$$\begin{array}{lll} (P_1 \sqcap P_2) \parallel Q \equiv (P_1 \parallel Q) \sqcap (P_2 \parallel Q) & \{ \text{ by A–8 } \} \\ \qquad\qquad \equiv (Q \parallel P_1) \sqcap (Q \parallel P_2) & \{ \text{ by inductive hypothesis } \} \\ \qquad\qquad \equiv Q \parallel (P_1 \sqcap P_2) & \{ \text{ by A–9 } \} \end{array}$$

If $Q = Q_1 \sqcap Q_2$ then the result follows similarly.

Finally if $P = [X \subseteq A \rightarrow P_X]$ and $Q = [Y \subseteq B \rightarrow Q_Y]$ then $d(P_X) + d(Q_Y) < d(P) + d(Q)$ for all $X \subseteq A$, $Y \subseteq B$.

$$\begin{array}{ll} & [X \subseteq A \rightarrow P_X] \parallel [Y \subseteq B \rightarrow Q_Y] \\ \equiv & \quad \{ \text{ by A–10 } \} \\ & [Z \subseteq (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P) \rightarrow P_{Z\cap A} \parallel Q_{Z\cap B}] \\ \equiv & \quad \{ \text{ by inductive hypothesis } \} \\ & [Z \subseteq (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P) \rightarrow Q_{Z\cap B} \parallel P_{Z\cap A}] \\ \equiv & \quad \{ \text{ as intersection and union are commutative } \} \\ & [Z \subseteq (B \cap A) \cup (B - \alpha P) \cup (A - \alpha Q) \rightarrow Q_{Z\cap B} \parallel P_{Z\cap A}] \\ \equiv & \quad \{ \text{by A–10 } \} \\ & [Y \subseteq B \rightarrow Q_Y] \parallel [X \subseteq A \rightarrow P_X] \end{array}$$

□

B.4 Proof system for SRPT^I

Here we present the proof system for the language SRPT^I, the language of finite closed terms from SRPT.

Axioms for non-deterministic choice:

a–1	$\vdash_R P \sqcap Q \equiv_R Q \sqcap P$
a–2	$\vdash_R (P \sqcap Q) \sqcap R \equiv_R P \sqcap (Q \sqcap R)$
a–3	$\vdash_R P \sqcap P \equiv_R P$
a–4	$\vdash_R P \sqcap \bot \equiv_R \bot$

Axiom for set prefix:

a–5	$\vdash_R [!B?X \to P_X] \sqcap [!B?Y \to Q_Y] \equiv_R [!B?X \to P_X \sqcap Q_X]$

Axioms for parallel composition:

a–6	$\vdash_R \bot_{I,O} \parallel P \equiv_R \bot_{(I \cup_i P)-(O \cup_o P),(O \cup_o P)}$
a–7	$\vdash_R P \parallel \bot_{I,O} \equiv_R \bot_{(I \cup_i P)-(O \cup_o P),(O \cup_o P)}$
a–8	$\vdash_R (P \sqcap Q) \parallel R \equiv_R (P \parallel R) \sqcap (Q \parallel R)$
a–9	$\vdash_R P \parallel (Q \sqcap R) \equiv_R (P \parallel Q) \sqcap (P \parallel R)$
a–10	$\vdash_R [!B?X \to P_X] \parallel [!C?Y \to Q_Y] \equiv_R$ $[!B \cup C?Z \to P_{(Z \cup C) \cap_i P_{\{\}}} \parallel Q_{(Z \cup B) \cap_i Q_{\{\}}}]$

Axioms for hiding:

a–11	$\vdash_R \bot_{I,O} \setminus B \equiv_R \bot_{I,O-B}$
a–12	$\vdash_R (P \sqcap Q) \setminus A \equiv_R (P \setminus A) \sqcap (Q \setminus A)$
a–13	$\vdash_R [!B?X \to P_X] \setminus A \equiv_R [!(B - A)?X \to (P_X \setminus A)]$

Axioms for renaming:

a–14	$\vdash_R \bot_{I,O}[S] \equiv_R \bot_{I[S],O[S]}$
a–15	$\vdash_R (P \sqcap Q)[S] \equiv_R (P[S]) \sqcap (Q[S])$
a–16	$\vdash_R [!B?X \to P_X][S] \equiv_R [!B[S]?X \to P_{X[S^{-1}]}[S]]$

Ordering rules:

o–1	$\vdash_R \ P \sqcap Q \sqsubseteq_R P$	**o–3**	$\dfrac{P \equiv_R Q}{P \sqsubseteq_R Q \sqsubseteq_R P}$
o–2	$\dfrac{P \sqsubseteq_R Q \sqsubseteq_R P}{P \equiv_R Q}$	**o–4**	$\dfrac{P \sqsubseteq_R Q \sqsubseteq_R R}{P \sqsubseteq_R R}$

Monotonicity rules:

m–1	$\dfrac{P_1 \sqsubseteq_R P_2 \wedge Q_1 \sqsubseteq_R Q_2}{P_1 \sqcap Q_1 \sqsubseteq_R P_2 \sqcap Q_2}$
m–2	$\dfrac{\forall X \subseteq B \bullet P_X \sqsubseteq_R Q_X}{[!B?X \to P_X] \sqsubseteq_R [!B?X \to Q_X]}$

B.5 Proof system for SRPT

The proof system for the closed terms of SRPT consists of all the rules in the previous section, with the following additions.

Axioms for recursion:

a–17	$\vdash_R \ P[(\mu x : I, O \bullet P)/x] \equiv_R \mu x : I, O \bullet P$

Least fixed point rule:

r–1	$\dfrac{\forall Q \in FIN_R(P) \bullet Q \sqsubseteq_R R}{P \sqsubseteq_R R}$

Appendix C

Algebraic Derivations

In this appendix we demonstrate the use of the algebraic laws of SCSP and SRPT. We present the first steps in the derivation of resluts in the Token ring example (Section 4.2.4), which used the language SCSP. We also derive some of the results required in the sorter example (Section 7.4.2), which used the language SRPT and timewise abstraction.

C.1 Token ring interface with data

Recall that for $X \in \{L, T, D\}$ the definition of $ID(y, X, s)$ is

$$ID(y, X, s) \triangleq (I(y, X, s) \parallel DATA) \setminus \{on!\}$$

where $I(y, X, s)$ is defined in Figure 4.3 and

$$DATA \triangleq (\prod_{d \in \delta(on)} (on!d \rightsquigarrow DATA)) \sqcap (wait(1) \rightarrow DATA)$$

We shall show that

$$ID(y, L, \langle\rangle) \equiv [\{in?x, out!y\} \rightarrow ((\textstyle\prod_{d \in \delta(on)} ID'(x, L, fr(d))) \sqcap ID'(x, L, \langle\rangle)) \triangleright \bot]$$

and

$$ID'(y, L, \langle\rangle) \equiv wait(2) \rightarrow ((\textstyle\prod_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle))$$

where $ID'(y, L, s:frs) \triangleq (I'(y, L, s:frs) \parallel DATA) \setminus \{on!\}$

Firstly

$$
\begin{array}{ll}
 & ID(y, L, \langle\rangle) \\
\equiv & \{ \text{ by definition } \} \\
 & (I(y, L, \langle\rangle) \parallel DATA) \setminus \{on!\} \\
\equiv & \{ \text{ expanding } DATA \} \\
 & (I(y, L, \langle\rangle) \parallel ((\sqcap_{d \in \delta(on)}(on!d \rightsquigarrow DATA)) \\
 & \qquad \sqcap(wait(I) \rightarrow DATA))) \setminus \{on!\} \\
\equiv & \{ \text{ by A-12 and A-8 } \} \\
 & (\sqcap_{d \in \delta(on)}(I(y, L, \langle\rangle) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\}) \\
 & \sqcap((I(y, L, \langle\rangle) \parallel (wait(I) \rightarrow DATA)) \setminus \{on!\})
\end{array}
$$

Now we recall

$$
\begin{array}{l}
I(y, L, \langle\rangle) \triangleq [\{in?x, out!y, on?d\} \rightarrow I'(x, L, fr(d)) \\
\qquad\qquad \Box \{in?x, out!y\} \rightarrow I'(x, L, \langle\rangle) \\
\qquad\qquad \rhd \perp]
\end{array}
$$

Thus

$$
\begin{array}{ll}
 & (I(y, L, \langle\rangle) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\} \\
\equiv & \{ \text{ expanding } I(y, L, \langle\rangle) \text{ and using axioms A-10 and A-13 } \} \\
 & [\{in?x, out!y\} \rightarrow (I'(x, L, fr(d)) \parallel DATA) \setminus \{on!\} \\
 & \rhd \perp] \\
\equiv & \{ \text{ by definition } \} \\
 & [\{in?x, out!y\} \rightarrow ID'(x, L, fr(d)) \rhd \perp]
\end{array}
$$

and

$$
\begin{array}{ll}
 & (I(y, L, \langle\rangle) \parallel (wait(I) \rightarrow DATA)) \setminus \{on!\} \\
\equiv & \{ \text{ expanding } I(y, L, \langle\rangle) \text{ and using axioms A-10 and A-13 } \} \\
 & [\{in?x, out!y\} \rightarrow (I'(x, L, \langle\rangle) \parallel DATA) \setminus \{on!\} \rhd \perp]
\end{array}
$$

Hence

$$
\begin{array}{ll}
 & ID(y, L, \langle\rangle) \\
\equiv & \{ \text{ substituting the above results } \} \\
 & (\sqcap_{d \in \delta(on)}[\{in?x, out!y\} \rightarrow ID'(x, L, fr(d)) \rhd \perp]) \\
 & \sqcap[\{in?x, out!y\} \rightarrow (I'(x, L, \langle\rangle) \parallel DATA) \setminus \{on!\} \rhd \perp] \\
\equiv & \{ \text{ by L-1 } \} \\
 & [\{in?x, out!y\} \rightarrow ((\sqcap_{d \in \delta(on)} ID'(x, L, fr(d))) \\
 & \qquad\qquad \sqcap(I'(x, L, \langle\rangle) \parallel DATA) \setminus \{on!\}) \\
 & \rhd \perp]
\end{array}
$$

Now let us consider

$$(I'(y, L, \langle\rangle) \parallel DATA) \setminus \{on!\}$$

We rewrite $I'(y, L, \langle\rangle)$ as follows:

$$I'(y, L, \langle\rangle) \triangleq [\{on?d\} \rightarrow P(y, fr(d)) \rhd P(y, \langle\rangle)]$$

where

$$P(y, s:frs) \triangleq (wait(1) \rightarrow I(y, L, s:frs))$$

$$P(y, \langle\rangle) \triangleq [\{on?d\} \rightarrow I(y, L, fr(d)) \rhd I(y, L, \langle\rangle)]$$

Now

$$\begin{array}{ll}
 & (P(y, s:frs) \parallel DATA) \setminus \{on!\} \\
\equiv & \{ \text{ expanding } DATA \text{ and by A–8 and A–12 } \} \\
 & (\sqcap_{d \in \delta(on)} (P(y, s:frs) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\}) \\
 & \sqcap ((P(y, s:frs) \parallel (wait(1) \rightarrow DATA)) \setminus \{on!\}) \\
\equiv & \{ \text{ expanding processes and applying axioms A–10 and A–13 } \} \\
 & (\sqcap_{d \in \delta(on)} (wait(1) \rightarrow (I(y, L, s:frs) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\})) \\
 & \sqcap (wait(1) \rightarrow (I(y, L, s:frs) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ by L-1 } \} \\
 & wait(1) \rightarrow ((\sqcap_{d \in \delta(on)} (I(y, L, s:frs) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\}) \\
 & \qquad \sqcap (I(y, L, s:frs) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ by axioms A–8 and A-12 } \} \\
 & wait(1) \rightarrow \\
 & \qquad ((I(y, L, s:frs) \parallel (\sqcap_{d \in \delta(on)} (on!d \rightsquigarrow DATA) \sqcap DATA)) \setminus \{on!\}) \\
\equiv & \{ \text{ by definition of } DATA \text{ and A–3 } \} \\
 & wait(1) \rightarrow ((I(y, L, s:frs) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ by definition of } ID \ \} \\
 & wait(1) \rightarrow ID(y, L, s:frs)
\end{array}$$

Also

$$\begin{array}{ll}
 & (P(y, \langle\rangle) \parallel DATA) \setminus \{on!\} \\
\equiv & \{ \text{ expanding } DATA \text{ and by A–8 and A–12 } \} \\
 & (\sqcap_{d \in \delta(on)} (P(y, \langle\rangle) \parallel (on!d \rightsquigarrow DATA)) \setminus \{on!\}) \\
 & \sqcap ((P(y, \langle\rangle) \parallel (wait(1) \rightarrow DATA)) \setminus \{on!\}) \\
\equiv & \{ \text{ expanding processes and applying axioms A–10 and A–13 } \} \\
 & (\sqcap_{d \in \delta(on)} (wait(1) \rightarrow (I(y, L, fr(d)) \parallel DATA) \setminus \{on!\})) \\
 & \sqcap (wait(1) \rightarrow (I(y, L, \langle\rangle) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ by L–1 } \} \\
 & wait(1) \rightarrow ((\sqcap_{d \in \delta(on)} (I(y, L, fr(d)) \parallel DATA) \setminus \{on!\}) \\
 & \qquad \sqcap (I(y, L, \langle\rangle) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ by definition of } ID \ \} \\
 & wait(1) \rightarrow ((\sqcap_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle))
\end{array}$$

Finally

$$
\begin{array}{ll}
 & (I'(y, L, \langle\rangle) \parallel DATA) \setminus \{on!\} \\
\equiv & \{ \text{ by the same working as above } \} \\
 & wait(1) \rightarrow ((\sqcap_{d \in \delta(on)}(P(y, fr(d)) \parallel DATA) \setminus \{on!\}) \\
 & \qquad \sqcap (P(y, \langle\rangle) \parallel DATA) \setminus \{on!\}) \\
\equiv & \{ \text{ substituting from earlier working } \} \\
 & wait(1) \rightarrow ((\sqcap_{d \in \delta(on)}(wait(1) \rightarrow ID(y, L, fr(d)))) \\
 & \qquad \sqcap (wait(1) \rightarrow ((\sqcap_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle)))) \\
\equiv & \{ \text{ by L-1 } \} \\
 & wait(2) \rightarrow ((\sqcap_{d \in \delta(on)} ID(y, L, fr(d))) \\
 & \qquad \sqcap ((\sqcap_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle)))) \\
\equiv & \{ \text{ idempotence of } \sqcap \ \} \\
 & wait(2) \rightarrow ((\sqcap_{d \in \delta(on)} ID(y, L, fr(d))) \sqcap ID(y, L, \langle\rangle))
\end{array}
$$

Hence we have the required results.

C.2 The sorter pipeline

In this section we present in more detail some of the steps of the derivations used in Section 7.4.2.

C.2.1 First phase of the pipeline

We recall that the overall aim was to obtain an algebraic representation of the first phase of the pipeline in a form and time frame in which it can be deduced that this phase is a pipe. So we want to derive an expansion of:

$$Slow(2, 0, \{\}, (Phase1 \parallel CK) \setminus \{ck\})$$

which only involves the set prefix and nondeterministic choice constructs.

Recall

$$Phase1 \triangleq Q[i0/d0, i1/d1, a0/c, a1/d] \parallel Q[i2/d0, i3/d1, a2/c, a3/d]$$

where $Q \triangleq (Dff[d0/d, a/q] \parallel Dff[d1/d, b/q] \parallel Comp) \setminus \{a, b\}$

and the definitions of $Comp$ and Dff are given in Section 7.4.2 (pages 139 and 140).

In order to simplify the expansions, we set

$$
\begin{aligned}
DD(x,\{\}) &\triangleq Dff_x[d0/d, a/q] \parallel Dff_x[d1/d, b/q] \\
DD(x,\{d0\}) &\triangleq Dff'_x[d0/d, a/q] \parallel Dff_x[d1/d, b/q] \\
DD(x,\{d1\}) &\triangleq Dff_x[d0/d, a/q] \parallel Dff'_x[d1/d, b/q] \\
DD(x,\{d0,d1\}) &\triangleq Dff'_x[d0/d, a/q] \parallel Dff'_x[d1/d, b/q]
\end{aligned}
$$

The first parameter of DD takes the value L or H and should be interpreted as the voltage level on the clock. The second parameter is the set of input wires with high voltage at the time of the last rising edge in the clock signal.

We also set

$$
\begin{aligned}
DDC(x,\{\}) &\triangleq (DD(x,\{\}) \parallel Comp) \setminus \{a,b\} \\
DDC'(x,y) &\triangleq (DD(x,y) \parallel Comp') \setminus \{a,b\} \\
DDC(x,\{d0,d1\}) &\triangleq (DD(x,\{d0,d1\}) \parallel Comp'') \setminus \{a,b\}
\end{aligned}
$$

where $x \in \{L,H\}$ and $y \in \{\{d0\},\{d1\}\}$.

So $Q \triangleq DDC(L,\{\})$

First we evaluate

$$
\begin{aligned}
& DD(L,\{\}) \\
\equiv\ & \{ \text{ expanding definition of } DD \ \} \\
& Dff_L[d0/d, a/q] \parallel Dff_L[d1/d, b/q] \\
\equiv\ & \{ \text{ expanding definition of } Dff \ \} \\
& ([!\{\}?X \rightarrow (Dff_L \textbf{ if } ck \notin X \textbf{ else} \\
& \quad ([!\{q\}?Y \rightarrow (Dff'_H \textbf{ if } ck \in Y \textbf{ else } Dff'_L)] \\
& \quad \textbf{if } d \in X \textbf{ else} \\
& \quad [!\{\}?Y \rightarrow (Dff_H \textbf{ if } ck \in Y \textbf{ else } Dff_L)]))][d0/d, a/q]) \\
& \parallel ([!\{\}?X \rightarrow (Dff_L \textbf{ if } ck \notin X \textbf{ else} \\
& \quad ([!\{q\}?Y \rightarrow (Dff'_H \textbf{ if } ck \in Y \textbf{ else } Dff'_L)] \\
& \quad \textbf{if } d \in X \textbf{ else} \\
& \quad [!\{\}?Y \rightarrow (Dff_H \textbf{ if } ck \in Y \textbf{ else } Dff_L)]))][d1/d, b/q]) \\
\equiv\ & \{ \text{ by a-16 } \} \\
& [!\{\}?X \rightarrow (Dff_L[d0/d, a/q] \textbf{ if } ck \notin X \textbf{ else} \\
& \quad ([!\{a\}?Y \rightarrow (Dff'_H[d0/d, a/q] \textbf{ if } ck \in Y \textbf{ else } Dff'_L[d0/d, a/q])] \\
& \quad \textbf{if } d0 \in X \textbf{ else} \\
& \quad [!\{\}?Y \rightarrow (Dff_H[d0/d, a/q] \textbf{ if } ck \in Y \textbf{ else } Dff_L[d0/d, a/q])]))] \\
& \parallel [!\{\}?X \rightarrow (Dff_L[d1/d, b/q] \textbf{ if } ck \notin X \textbf{ else} \\
& \quad ([!\{b\}?Y \rightarrow (Dff'_H[d1/d, b/q] \textbf{ if } ck \in Y \textbf{ else } Dff'_L[d1/d, b/q])] \\
& \quad \textbf{if } d1 \in X \textbf{ else} \\
& \quad [!\{\}?Y \rightarrow (Dff_H[d1/d, b/q] \textbf{ if } ck \in Y \textbf{ else } Dff_L[d1/d, b/q])]))]
\end{aligned}
$$

$\equiv$ { by a–10 and noting definition of DD }

$[!\{\}?X \rightarrow (DD(L,\{\})$ **if** $ck \notin X$ **else**
$([!\{a,b\}?Y \rightarrow (DD(H,\{d0,d1\})$ **if** $ck \in Y$ **else** $DD(L,\{d0,d1\}))]$
if $\{d0,d1\} \subseteq X$ **else**
$([!\{a\}?Y \rightarrow (DD(H,\{d0\})$ **if** $ck \in Y$ **else** $DD(L,\{d0\}))]$
if $\{d0\} \subseteq X$ **else**
$([!\{b\}?Y \rightarrow (DD(H,\{d1\})$ **if** $ck \in Y$ **else** $DD(L,\{d1\}))]$
if $\{d1\} \subseteq X$ **else**
$[!\{\}?Y \rightarrow (DD(H,\{\})$ **if** $ck \in Y$ **else** $DD(L,\{\}))]))))]$

Thus

$DDC(L,\{\})$

$\equiv$ { expanding definition of DDC }

$(DD(L,\{\}) \parallel Comp) \setminus \{a,b\}$

$\equiv$ { recalling expansions and by a–10 }

$[!\{\}?X \rightarrow ((DD(L,\{\}) \parallel Comp)$ **if** $ck \notin X$ **else**
$([!\{a,b\}?Y \rightarrow ((DD(H,\{d0,d1\}) \parallel Comp'')$ **if** $ck \in Y$ **else**
$(DD(L,\{d0,d1\}) \parallel Comp''))]$
if $\{d0,d1\} \subseteq X$ **else**
$([!\{a\}?Y \rightarrow ((DD(H,\{d0\}) \parallel Comp')$ **if** $ck \in Y$ **else**
$(DD(L,\{d0\}) \parallel Comp'))]$
if $\{d0\} \subseteq X$ **else**
$([!\{b\}?Y \rightarrow ((DD(H,\{d1\}) \parallel Comp')$ **if** $ck \in Y$ **else**
$(DD(L,\{d1\}) \parallel Comp'))]$
if $\{d1\} \subseteq X$ **else**
$[!\{\}?Y \rightarrow ((DD(H,\{\}) \parallel Comp)$ **if** $ck \in Y$ **else**
$(DD(L,\{\}) \parallel Comp))]$
$))))] \setminus \{a,b\}$

$\equiv$ { by a–13 and noting definition of DDC }

$[!\{\}?X \rightarrow (DDC(L,\{\})$ **if** $ck \notin X$ **else**
$([!\{\}?Y \rightarrow (DDC(H,\{d0,d1\})$ **if** $ck \in Y$ **else** $DDC(L,\{d0,d1\}))]$
if $\{d0,d1\} \subseteq X$ **else**
$([!\{\}?Y \rightarrow (DDC(H,\{d0\})$ **if** $ck \in Y$ **else** $DDC(L,\{d0\}))]$
if $\{d0\} \subseteq X$ **else**
$([!\{\}?Y \rightarrow (DDC(H,\{d1\})$ **if** $ck \in Y$ **else** $DDC(L,\{d1\}))]$
if $\{d1\} \subseteq X$ **else**
$[!\{\}?Y \rightarrow (DDC(H,\{\})$ **if** $ck \in Y$ **else** $DDC(L,\{\}))]$ $))))]$

$\equiv$ { rearranging terms }

$[!\{\}?X \rightarrow (DDC(L,\{\})$ **if** $ck \notin X$ **else**
$[!\{\}?Y \rightarrow (DDC(H,X-\{ck\})$ **if** $ck \in Y$ **else** $DDC(L,X-\{ck\}))]])]$

Continuing in this manner we can also show that

$$
\begin{array}{rcl}
DDC(H,\{\}) & \equiv & [!\{\}?X \rightarrow (DDC(H,\{\}) \textbf{ if } ck \in X \textbf{ else } DDC(L,\{\}))] \\
DDC(L,\{d0\}) & \equiv & [!\{c\}?X \rightarrow (DDC(L,\{d0\}) \textbf{ if } ck \notin X \textbf{ else} \\
 & & \quad [!\{c\}?Y \rightarrow (DDC(H,X-\{ck\}) \\
 & & \qquad \textbf{if } ck \in Y \textbf{ else } DDC(L,X-\{ck\}))])] \\
DDC(H,\{d0\}) & \equiv & [!\{c\}?X \rightarrow (DDC(H,\{d0\}) \\
 & & \qquad \textbf{if } ck \in X \textbf{ else } DDC(L,\{d0\}))] \\
DDC(L,\{d1\}) & \equiv & [!\{c\}?X \rightarrow (DDC(L,\{d1\}) \textbf{ if } ck \notin X \textbf{ else} \\
 & & \quad [!\{c\}?Y \rightarrow (DDC(H,X-\{ck\}) \\
 & & \qquad \textbf{if } ck \in Y \textbf{ else } DDC(L,X-\{ck\}))])] \\
DDC(H,\{d1\}) & \equiv & [!\{c\}?X \rightarrow (DDC(H,\{d1\}) \\
 & & \qquad \textbf{if } ck \in X \textbf{ else } DDC(L,\{d1\}))] \\
DDC(L,\{d0,d1\}) & \equiv & [!\{c,d\}?X \rightarrow (DDC(L,\{d0,d1\}) \textbf{ if } ck \notin X \textbf{ else} \\
 & & \quad [!\{c,d\}?Y \rightarrow (DDC(H,X-\{ck\}) \\
 & & \qquad \textbf{if } ck \in Y \textbf{ else } DDC(L,X-\{ck\}))])] \\
DDC(H,\{d0,d1\}) & \equiv & [!\{c,d\}?X \rightarrow (DDC(H,\{d0,d1\}) \\
 & & \qquad \textbf{if } ck \in X \textbf{ else } DDC(L,\{d0,d1\}))]
\end{array}
$$

Hence, by uniqueness of solutions to guarded recursive equations

$$Q \equiv S(L,\{\})$$

where S is defined in Section 7.4.2 (page 141).

We are now in a position to reduce the expression for *Phs1*, eliminating parallel composition, hiding and renaming. We recall

$$Phs1 \triangleq (Phase1 \parallel CK) \setminus \{ck\}$$

Now we can expand *Phase1* as follows:

$$
\begin{array}{ll}
 & Phase1 \\
\equiv & \{ \text{ by definition } \} \\
 & Q[i0/d0, i1/d1, a0/c, a1/d] \parallel Q[i2/d0, i3/d1, a2/c, a3/d] \\
\equiv & \{ \text{ by the equivalence deduced above } \} \\
 & S(L,\{\})[i0/d0, i1/d1, a0/c, a1/d] \parallel S(L,\{\})[i2/d0, i3/d1, a2/c, a3/d] \\
\equiv & \{ \text{ expanding the definition of } S \} \\
 & ([!\{\}?X \rightarrow (S(l,\{\}) \textbf{ if } ck \notin X \textbf{ else} \\
 & \quad [!\{\}?Y \rightarrow (S(H,X-\{ck\}) \textbf{ if } ck \in Y \textbf{ else} \\
 & \qquad S(L,X-\{ck\}))])[i0/d0, i1/d1, a0/c, a1/d]) \\
 & \parallel ([!\{\}?X \rightarrow (S(l,\{\}) \textbf{ if } ck \notin X \textbf{ else} \\
 & \quad [!\{\}?Y \rightarrow (S(H,X-\{ck\}) \textbf{ if } ck \in Y \textbf{ else} \\
 & \qquad S(L,X-\{ck\}))])[i2/d0, i3/d1, a2/c, a3/d])
\end{array}
$$

$\equiv$ { by a–16 }

$[!\{\}?X \rightarrow (S(l,\{\})[i0/d0, i1/d1, a0/c, a1/d]$ **if** $ck \notin X$ **else**

$[!\{\}?Y \rightarrow (S(H,(X-\{ck\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

if $ck \in Y$ **else**

$S(L,(X-\{ck\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d])]$

$\|[!\{\}?X \rightarrow (S(L,\{\})[i2/d0, i3/d1, a2/c, a3/d]$ **if** $ck \notin X$ **else**

$[!\{\}?Y \rightarrow (S(H,(X-\{ck\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d]$

if $ck \in Y$ **else**

$S(L,(X-\{ck\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d])]$

$\equiv$ { by a–10 }

$[!\{\}?X \rightarrow ((S(L,\{\})[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(l,\{\})[i2/d0, i3/d1, a2/c, a3/d])$

if $ck \notin X$ **else** $[!\{\}?Y \rightarrow$

$(\|S(H,(X \cap \{i0,i1\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(H,(X \cap \{i2,i3\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d])$

if $ck \in Y$ **else**

$(S(L,(X \cap \{i0,i1\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(L,(X \cap \{i2,i3\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d]))]$

Thus

$Phs1$

$\equiv$ { by definition of $Phs1$ }

$(PhastA \| CK) \setminus \{ck\}$

$\equiv$ { expanding terms }

$([!\{\}?X \rightarrow ((S(l,\{\})[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(l,\{\})[i2/d0, i3/d1, a2/c, a3/d])$

if $ck \notin X$ **else** $[!\{\}?Y \rightarrow$

$((S(H,(X \cap \{i0,i1\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(H,(X \cap \{i2,i3\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d])$

if $ck \in Y$ **else**

$(S(L,(X \cap \{i0,i1\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(L,(X \cap \{i2,i3\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d]))])]$

$\|[!\{ck\} \rightarrow [!\{\} \rightarrow CK]]) \setminus \{ck\}$

$\equiv$ { by a–10 }

$[!\{ck\}?X \rightarrow [!\{\}?Y \rightarrow$

$(S(L,(X \cap \{i0,i1\})[d0/i0, d1/i1])[i0/d0, i1/d1, a0/c, a1/d]$

$\| S(L,(X \cap \{i2,i3\})[d0/i2, d1/i3])[i2/d0, i3/d1, a2/c, a3/d]$

$\| CK)]] \setminus \{ck\}$

$\equiv$ { by a-13 }

$$
\begin{array}{l}
[!\{\}?X \rightarrow [!\{\}?Y \rightarrow \\
\quad (S(L,(X \cap \{i0,i1\})\{d0/i0,d1/i1\})[i0/d0,i1/d1,a0/c,a1/d] \\
\quad \| S(L,(X \cap \{i2,i3\})\{d0/i2,d1/i3\})[i2/d0,i3/d1,a2/c,a3/d] \\
\quad \| CK) \setminus \{ck\}]]
\end{array}
$$

Continuing in this manner we can demonstrate that

$$Phs1 \equiv Ph1(\{\})$$

where $Ph1$ is given in Figure C.1.

$$
\begin{array}{rl}
Ph1(\{\}) & \hat{=} [!\{\}?X \rightarrow [!\{\}?Y \rightarrow Ph1(X)]] \\
\left.\begin{array}{r} Ph1(\{i0\}) \\ Ph1(\{i1\}) \end{array}\right\} & \hat{=} [!\{a0\}?X \rightarrow [!\{a0\}?Y \rightarrow Ph1(X)]] \\
\left.\begin{array}{r} Ph1(\{i2\}) \\ Ph1(\{i3\}) \end{array}\right\} & \hat{=} [!\{a2\}?X \rightarrow [!\{a2\}?Y \rightarrow Ph1(X)]] \\
Ph1(\{i0,i1\}) & \hat{=} [!\{a0,a1\}?X \rightarrow [!\{a0,a1\}?Y \rightarrow Ph1(X)]] \\
\left.\begin{array}{r} Ph1(\{i0,i2\}) \\ Ph1(\{i0,i3\}) \\ Ph1(\{i1,i2\}) \\ Ph1(\{i1,i3\}) \end{array}\right\} & \hat{=} [!\{a0,a2\}?X \rightarrow [!\{a0,a2\}?Y \rightarrow Ph1(X)]] \\
Ph1(\{i2,i3\}) & \hat{=} [!\{a2,a3\}?X \rightarrow [!\{a2,a3\}?Y \rightarrow Ph1(X)]] \\
\left.\begin{array}{r} Ph1(\{i0,i1,i2\}) \\ Ph1(\{i0,i1,i3\}) \end{array}\right\} & \hat{=} [!\{a0,a1,a2\}?X \rightarrow [!\{a0,a1,a2\}?Y \rightarrow Ph1(X)]] \\
\left.\begin{array}{r} Ph1(\{i0,i2,i3\}) \\ Ph1(\{i1,i2,i3\}) \end{array}\right\} & \hat{=} [!\{a0,a2,a3\}?X \rightarrow [!\{a0,a2,a3\}?Y \rightarrow Ph1(X)]] \\
Ph1(\{i0,i1,i2,i3\}) & \hat{=} [!\{a0,a1,a2,a3\}?X \rightarrow \\
 & \qquad [!\{a0,a1,a2,a3\}?Y \rightarrow Ph1(X)]]
\end{array}
$$

Figure C.1: Expansion of the first phase of the sorter pipeline in original time frame

It remains to calculate $\mathcal{S}low(2, 0, \{\}, Phs1)$.

For all $\lambda \subseteq \{i0, i1, i2, i3\}$ we define:

$$SPh1(X) \triangleq \mathcal{S}low(2, 0, \{\}, Ph1(X))$$

Now

$\mathcal{S}low(2, 0, \{\}, Phs1)$
$\equiv$ { expanding definition of $SPh1$ }
$\mathcal{S}low(2, 0, \{\}, Ph1(\{\}))$
$\equiv$ { expanding definition of $Ph1$ }
$\mathcal{S}low(2, 0, \{\}, [!\{\}?X \rightarrow [!\{\}?Y \rightarrow Ph1(X)]])$
$\equiv$ { by a- 20 }
$[!\{\}?X \rightarrow \mathcal{S}low(2, 1, X, [!\{\}?Y \rightarrow Ph1(X)])]$
$\equiv$ { by a–20 }
$[!\{\}?X \rightarrow \mathcal{S}low(2, 0, X, Ph1(X))]$
$\equiv$ { by note on page 124 }
$[!\{\}?X \rightarrow \mathcal{S}low(2, 0, \{\}, Ph1(X))]$
$\equiv$ { by definition of $SPh1$ }
$[!\{\}?X \rightarrow SPh1(X)]$

Hence, continuing in this manner, and by the uniqueness of solutions to guarded recursive equations we have that

$$\mathcal{S}low(2, 0, \{\}, Phs1) \equiv P1(\{\})$$

where $P1$ is defined in Figure 7.6.

C.2.2 Composing pipes

Recall that we need to evaluate an expansion of:

$$\mathcal{S}low(3, 0, \{\}, (Phase1' \gg Phase2') \gg Phase3')$$

which enables us to deduce the effect of this pipe of length 1.

By Theorem 7.5

$$\mathcal{S}low(3, 0, (Phase1' \gg Phase2') \gg Phase3') \equiv \mathcal{S}low(2, 0, \{\}, (\mathcal{S}low(2, 0, \{\}, Phase1' \gg Phase2') \gg Phase3'))$$

So as a first step we should evaluate $\mathcal{S}low(2, 0, \{\}, Phase1' \gg Phase2')$

Now

$$
\begin{aligned}
& Phase1' \gg Phase2' \\
\equiv\ & \{\ \text{definition of processes and chaining}\ \} \\
& P1(\{\}) \parallel P2(\{\}) \setminus \{a0, a1, a2, a3\} \\
\equiv\ & \{\ \text{expanding processes}\ \} \\
& [!\{\}?X \rightarrow P1(X)]\|\|[!\{\}?X \rightarrow P2(X)] \setminus \{a0, a1, a2, a3\} \\
\equiv\ & \{\ \text{by a-10}\ \} \\
& [!\{\}?X \rightarrow (P1(X)\|P2(\{\}))] \setminus \{a0, a1, a2, a3\} \\
\equiv\ & \{\ \text{by a-13}\ \} \\
& [!\{\}?X \rightarrow (P1(X)\|P2(\{\})) \setminus \{a0, a1, a2, a3\}] \\
\equiv\ & \{\ \text{expanding processes}\ \} \\
& [!\{\}?X \rightarrow ([!f_1(X)?Y \rightarrow P1(Y)]\|[!\{\}?Y \rightarrow P2(Y)]) \setminus \{a0, a1, a2, a3\}] \\
\equiv\ & \{\ \text{by a-10}\ \} \\
& [!\{\}?X \rightarrow ([!\{\}?Y \rightarrow (P1(Y)\|P2(f_1(X)))]) \setminus \{a0, a1, a2, a3\}] \\
\equiv\ & \{\ \text{by a-13}\ \} \\
& [!\{\}?X \rightarrow [!\{\}?Y \rightarrow (P1(Y)\|P2(f_1(X))) \setminus \{a0, a1, a2, a3\}]]
\end{aligned}
$$

So

$$
\begin{aligned}
& Slow(2, 0, \{\}, Phase1' \gg Phase2') \\
\equiv\ & \{\ \text{by a-20 and using above expansion}\ \} \\
& [!\{\}?X \rightarrow Slow(2, 1, X, [!\{\}?Y \rightarrow (P1(Y)\|P2(f_1(X))) \setminus \{a0, a1, a2, a3\}])] \\
\equiv\ & \{\ \text{by a-20}\ \} \\
& [!\{\}?X \rightarrow Slow(2, 0, X, (P1(Y)\|P2(f_1(X))) \setminus \{a0, a1, a2, a3\})] \\
\equiv\ & \{\text{by uote on page 124 and defn. of chaining}\ \} \\
& [!\{\}?X \rightarrow Slow(2, 0, \{\}, (P1(Y) \gg P2(f_1(X))))]
\end{aligned}
$$

Continuing in this manner we can show that

$$Slow(2, 0, \{\}, Phase1' \gg Phase2') \equiv P12(\{\})$$

where $P12$ is given in Figure C.2.

Using the results above and the same approach we can show that

$$Slow(2, 0, \{\}, P12 \gg Phase3') \equiv P13(\{\})$$

where $P13(X) \triangleq [!f_{13}(X)?Y \rightarrow P13(Y)]$ for $X \subseteq \{i0, i1, i2, i3\}$

and $f_{13}(X) = \begin{cases} \{\} & \text{if } |X| = 0 \\ \{o0\} & \text{if } |X| = 1 \\ \{o0, o1\} & \text{if } |X| = 2 \\ \{o0, o1, o2\} & \text{if } |X| = 3 \\ \{o0, o1, o2, o3\} & \text{if } |X| = 4 \end{cases}$

From this the effect of $\mathcal{S}low(3, 0, \{\}, (Phase1' \gg Phase2' \gg Phase3'))$ can be deduced.

$$P12(X) \triangleq [!f_{12}(X)?Y \rightarrow P12(Y)]$$

where f_1 is defined over the domain $\mathbb{P}\{i0, i1, i2, i3\}$ as follows:

$$\left.\begin{array}{l} f_{12}(\{i0\}) \\ f_{12}(\{i1\}) \\ f_{12}(\{i2\}) \\ f_{12}(\{i3\}) \end{array}\right\} = \{b0\}$$

$$\left.\begin{array}{l} f_{12}(\{i0, i2\}) \\ f_{12}(\{i1, i2\}) \\ f_{12}(\{i0, i3\}) \\ f_{12}(\{i1, i3\}) \end{array}\right\} = \{b0, b1\}$$

$$\left.\begin{array}{l} f_{12}(\{i0, i1\}) \\ f_{12}(\{i2, i3\}) \end{array}\right\} = \{b0, b2\}$$

$$\left.\begin{array}{l} f_{12}(\{i0, i1, i2\}) \\ f_{12}(\{i0, i1, i3\}) \\ f_{12}(\{i0, i2, i3\}) \\ f_{12}(\{i1, i2, i3\}) \end{array}\right\} = \{b0, b1, b2\}$$

$$f_{12}(\{i0, i1, i2, i3\}) = \{b0, b1, b2, b3\}$$

$$f_{12}(\{\}) = \{\}$$

Figure C.2: The first two phases of the sorter pipeline

Bibliography

[BB88] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1), January 1988.

[BB90] J. C. M. Baeten and J. A. Bergstra. Real space process algebra. Technical Report P9005, University of Amsterdam, Programming Research Group, 1990.

[BB91a] J. C.M. Baeten and J. A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142-188, 1991.

[BB91b] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, September 1991.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87-152, 1992.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[BK84] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[BL92] T. Bolognesi and F. Lucidi. Timed process algebras with urgent interactions and a unique powerful binary operator. In *Real-Time: Theory in Practice*, LNCS 600. Springer-Verlag, 1992.

[BlGSS92] A. Benveniste, P. le Guernic, Y. Sorel, and M. Sorine. A denotational theory of synchronous reactive systems. *Information and Computation*, 99:192–230, 1992.

[BM79] R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, 1979.

[Bou85] G. Boudol. Notes on algebraic calculi of processes. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, volume F13 of *NATO ASI Serics*. Springer–Verlag, 1985.

[BR85] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Proceedings of the Pittsburgh Seminar on Concurrency*, LNCS 197, pages 281–305. Springer–Verlag, 1985.

[Bro83] S. D. Brookes. *A Model for Communicating Sequential Processes.* D.Phil. Thesis, Oxford University, 1983.

[BT89] A. Bronstein and C. L. Talcott. Formal verification of synchronous circuits based on string-functional semantics: The 7 Paillet circuits in Boyer-Moore. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 317–333. Springer-Verlag, 1989.

[Che91] Liang Chen. An interleaving model for real-time systems. LFCS Report ECS-LFCS-91-184, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[CLM91] E. M. Clarke, Jr., D. E. Long, and K. L. McMillan. A language for compositional specification and verification of finite state hardware controllers. *Proceedings of the IEEE*, 79(9):1283–1292, September 1991.

[CR85] J. E. Coolahan, Jr. and N. Roussopoulos. A timed Petri Net methodology for specifying real-time system requirements. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 24–31, Toronto, Italy, 1985. IEEE.

[Dan92] M. Daniels. Modelling real-time behaviour with an interval time calculus. In *Formal Techniques in Real-Time and Fault-Tolerant systems*, LNCS 571, pages 53–72. Springer-Verlag, 1992.

[Dav91] J. Davies. *Specification and Proof in Real-Time Systems.* D.Phil. Thesis, Oxford Uuiversity, 1991.

[Dil89] D. L. Dill. *Trace theory for automatic hicrarchical verification of speed-independent circuits.* MIT Press, 1989.

[DS89] J. Davies and S. Schneider. An introduction to Timed CSP. Technical Monograph PRG-75, Oxford University, Programming Research Group, 1989.

[DS92] J. Davies and S. Schneider. A brief history of Timed CSP. Technical Monograph PRG-96, Oxford University, Programming Research Group, 1992.

[FKTMO86] M. Fujita, S. Kono, H. Tanaka, and T. Moto-Oka. Aid to hierachial and structured logic design using temporal logic and prolog. *IEE Proceedings*, 133(5), 1986.

[For92] Formal Systems (Europe) Ltd. *Failures Divergence Refinement: Users manual and tutorial*, 1992.

[GMA89] G. C. Gopalakrishnan, N. S. Mani, and V. Akella. Parallel composition of lockstep synchronous processes for hardware validation: divide-and-conquer composition. In *Automatic Verification Methods for Finite State Systems*, LNCS 707. Springer-Verlag, 1989.

[Gor85] M. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, Cambridge University, Computing Laboratory, 1985.

[Gor86] M. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In *Formal Aspects of VLSI Design*. North-Holland, 1986.

[Har87] D. Harel. STATECHARTS: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[Hen88] M. Hennessy. *An Algebraic Theory of Processes*. MIT, 1988.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[Hoa86] C. A. R. Hoare. *Communicating Sequential Processes Exercises and Answers*. Prentice Hall International, 1986.

[Hoo90] J. Hooman. Compositional verification of distributed real-time systems. In H. Zedan, editor, *Real-Time Systems*, pages 1-20. North-Holland, 1990.

[HR90] M. Hennessy and T. Reagan. A temporal process algebra. Technical Report 2/90, University of Sussex, Computer Science, 1990.

[HSZFH92] C. Ho-Stuart, H. S. M. Zedan, M. Fang, and C. M. Holt. PARTY: A process algebra with real-time from York. Technical Report YCS 177, University of York, Department of Computer Science, 1992.

[IEE85] *802.5: Token Ring Access Method*, New York, 1985. IEEE.

[Jac92] D. M. Jackson. *Logical Verification of Reactive Software Systems.* D.Phil Thesis, Oxford University, 1992.

[Jef91a] A. Jeffrey. Discrete Timed CSP. PMG Memo 78, Chalmers University, Programming Methodology Group, 1991.

[Jef91b] A. Jeffrey. Abstract timed observation and process algebra. In *CONCUR '91*, LNCS 527, pages 332–345. Springer-Verlag, 1991.

[JIH89] M. B. Josephs, C. A. R. Hoare, and He Jifeng. A theory of asynchronous processes. Technical Report TR–7–89, Oxford University Computing Laboratory, 1989.

[Jos92] M. B. Josephs. Receptive process theory. *Acta Informatica*, 29(1):17–31, 1992.

[JS90] G. Jones and M. Sheeran. Circuit design in Ruby. In *Formal methods for VLSI design*, pages 13–70. North-Holland, 1990.

[KdR85] R. Koymans and W. P. de Roever. Examples of a real-time temporal logic specification. In *The Analysis of Concurrent Systems*, LNCS 207, pages 231–252. Springer-Verlag, 1985.

[KP92] Y. Kesten and A. Pnueli. Timed and hybrid Statecharts and their textual representation. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 571, pages 591–620. Springer-Verlag, 1992.

[MF76] P. M. Merlin and D. J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communications*, COM–24(9), September 1976.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil86] G. J. Milne. Towards vrerifiably correct vlsi design. In *Formal Aspects of VLSI Design.* North-Holland, 1986.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice–Hall International, 1989.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *CONCUR'90 Theories of Concurrency: Unification and Extention*, LNCS 458, pages 401–415. Springer–Verlag, 1990.

[MT91] F. Moller and C. Tofts. Relating processes with repect to speed. LFCS Report ECS–LFCS–91–143, Laboratory of Foundations of Computer Science, University of Edinburgh, 1991.

[NS90] X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and application. Technical Report RT–C26, LGI - IMAG, Grenoble, France, December 1990. (Revised version).

[OMdFE91] Y. Ortega-Mallén and D. de Frutos-Escrig. A complete proof system for timed observations. In *TAPSOFT'91*, LNCS 493, pages 412–440. Springer-Verlag, 1991.

[Pet77] J. L. Peterson. Petri nets. *Computing Surveys*, 9(3):223–252, 1977.

[PH88] A. Pnueli and E. Harel. Applications of temporal logic to the specification of real time systems. In *Proceedings of a Symposium on Formal techniques in Real-Time and Fault-Tolerant systems*, LNCS 331, pages 84–98. Springer-Verlag, 1988.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept, Århus University, Denmark, 1981.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. Providence, R.I., 1977.

[QAF90] J. Quemada, A. Azcorra, and D. Frutos. TIC: A timed calculus for LOTOS. In S. T. Vuong, editor, *Formal Discription Techniques, II, Proceedings of FORTE'89*. North-Holland, 1990.

[QF87] J. Quemada and A. Fernandez. Introduction of quantitative relative time into lotos. In *Protocol Specification, Testing and Verification VII*. North-Holland, 1987.

[Rei85] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer–Verlag, 1985.

[Ros88a] A. W. Roscoe. An alternative order for the failures model. Technical report, Oxford University, Programming Research Group, 1988. In [Ros88b].

[Ros88b] A. W. Roscoe. Two papers on CSP. Technical Monograph PRG–67, Oxford University, Programming Research Group, 1988.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for communicating sequential processes. In *Proceedings of ICALP'86*, LNCS 226, pages 314–323. Springer-Verlag, 1986.

[RR87] G. M. Reed and A. W. Roscoe. Metric spaces as models for real-time concurrency. In *Proceedings of Third Workshop on the Mathematical Foundations of Programming*, LNCS 298, pages 331–343. Springer-Verlag, 1987.

[Sch90] S. A. Schneider. *Correctness and Communication of Real-Time Systems.* D.Phil. thesis, Oxford University, 1990.

[Sch91] S. A. Schneider. The watchdog timer in Timed CSP, June 1991. BRA project 3096-SPEC Deliverable.

[She86] M. Sheeran. Design and verification of regular synchronous circuits. *IEE Proceedings*, 133(5), 1986.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT, 1977.

[Tan89] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall International, second edition, 1989.

[Tar55] A. Tarski. A lattice–theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Thi86] P. S. Thiagarajan. Elementary net systems. In *Petri Nets: Central Models and Their Properties*, LNCS 254, pages 26–59. Springer–Verlag, 1986.

[Too93] M. Tooley. *Electronic Circuits Handbook.* Butterworth-Heinmann Ltd, second edition, 1993.

[Yi91] Wang Yi. CCS + Time = an interleaving model for real-time systems. In *ICALP'91 Proceedings*, LNCS 510, pages 217–228. Springer-Verlag, 1991.

Glossary

Syntax

$\bot$	chaos	7, 76
$\sqcap$	non-deterministic choice	8, 76
$[X \subseteq A \rightharpoonup P_X]$	set prefix (SCSP)	8
$\Box$	choice in set prefix	10
$\rhd$	default in set prefix	10
$[!B?X \rightharpoonup P_X]$	output prefix (SRPT)	77
$\|\|$	parallel composition	10, 78
$\setminus$	hiding	11, 78
$P[S]$	renaming	11, 79
$\mu\, x : A \bullet P$	recursion (SCSP)	12
$\mu\, x : I, O \bullet P$	recursion (SRPT)	80
$(x_i \mathrel{\widehat{=}} P_i)_j$ with A	mutual recursion	13
$wait(n) \rightarrow P$	waiting	14
$STOP$	deadlock	14, 80
$a \rightsquigarrow P$	event prefix	16
$c!e$	output term	56
$c?x$	input term	56
$\delta(c)$	data set on channel	55
$chan(P)$	channels	55
$out(P)$	output channels	55
$in(P)$	input channels	55
$ev(P)$	non-communication events	55
$\gg$	chaining	135

SCSP	synchronous language terms	7
SCSP^0	non-recursive SCSP terms	35
$\text{SCSP}^!$	closed SCSP^0 terms	42
SRPT	synchronous receptive language terms	76
SRPT^0	non-recursive SRPT terms	92
$\text{SRPT}^!$	closed SRPT^0 terms	110

Semantics

Σ	universal alphabet	29
A	alphabet	29
$\overline{A}$	refusal alphabet	29
$\tilde{A}$	observation alphabet	29
ST_A	all traces with alphabet A (SCSP)	29
$RT_{I,O}$	all traces with alphabets I & O (SRPT)	86
$\cup$	union	29, 30
$\cap$	intersection	29
$-$	subtraction	29
$saturated_A(s)$	trace predicate	30
$\underline{\text{in}}$	trace membership	30
$feasible(B)$	set predicate	30
$\widehat{T}$	maximal behaviours	87
$T \downarrow S$	restriction	87
SM	model for SCSP	32
SM^A	model restricted to alphabet A	32
$SM_{\tilde{T}}$	trace projection of model	32
RM	model for SRPT	89
$\sqsubseteq$	non-determinism order	32
$\leqslant$	information order	89
Var	variables	33
$BIND$	domain of bindings (SCSP)	33
$BIND_R$	domain of bindings (SRPT)	92
ρ	variable binding (SCSP)	34
σ	variable binding (SRPT)	92
$\mathcal{M}$	semantic mapping for SCSP	33
T	trace projection of $\mathcal{M}$	35
α	alphabet projection of $\mathcal{M}$	35
$\mathcal{M}_\mathcal{R}$	semantic mapping for SRPT	92
$T_\mathcal{R}$	trace projection of $\mathcal{M}_\mathcal{R}$	93
ι	input alphabet projection of $\mathcal{M}_\mathcal{R}$	93
o	output alphabet projection of $\mathcal{M}_\mathcal{R}$	93

Proof Systems

	SCSP			SRPT	
$\sqsubseteq$	non-determinism order	42	$\sqsubseteq_R$	non-determinism order	110
$\equiv$	equivalence	42	$\equiv_R$	equivalence	110
$\vdash$	theorem	43	$\vdash_R$	theorem	111
$\prec$	syntactic approximation	50	$\prec_R$	syntactic approximation	117
$FIN(P)$	approximations of P	50	$FIN_R(P)$	approximations of P	117
A–	axiom		a–	axiom	
L–	law		l–	law	

Embeddings

Φ	embedding of RM in SM	100	ψ	image of trace	100
Θ	embedding of SRPTF in SCSP	106	ψ_I	image of maximal trace	100
η	embedding of $BIND_R$ in $BIND$	107	γ	output saturation	100
ϕ	trace projection of Φ	100			

Timewise Abstraction and Pipes

$Slow$	timewise abstraction operator	123	$\mathcal{E}_P$	effect of pipe	134
$choose$	trace contraction	131	ℓ_P	length of pipe	134
$\otimes$	trace multiplication	132	$\sim_p$	pipe equivalence	135

Mathematical Symbols

$\mathbb{N}$	set of natural numbers	$\lvert\_\rvert$	length of trace
$\mathbf{R}$	set of real numbers	$\sqcup$	least upper bound
$\mathbf{P}$	powerset operator	$\sqcap$	greatest lower bound
$\mathbb{F}$	set of all finite subsets	$\underline{\text{fix}}$	fixed point operator
$\{\}$	empty set		
A^*	finite traces	$\widehat{=}$	defined as
$\langle \ldots \rangle$	trace	$\square$	end of proof
$\langle\rangle$	empty trace	$\blacksquare$	end of theorem
$\frown$	concatenation	$\Diamond$	end of definition