Communicating Process Architectures 2004
Ian East, Jeremy Martin, Peter Welch, David Duce, and Mark Green (Eds.)
IOS Press, 2004

Triples

A.E. Lawrence
Department of Computer Science, Loughborough University, Leicestershire LE11 3TU UK.

Abstract. The most abstract form of acceptance semantics for a variant of *CSPP* is outlined. It encompasses processes which may involve priority, but covers a much wider class of systems including real time behaviour. It shares many of the features of the standard Failures-Divergences treatment: thus it is only a Complete Partial Order when the alphabet of events is finite.

1 Introduction

CSPP is a close relative of the process algebra CSP: [1], [2], [3]. It was designed originally to capture priority in the context of hardware compilation using a **occam**-like language. Its development has been informed by the work of the WoTUG community, and explorations of various sorts have been presented regularly in this conference series: [4], [5] [6], [7], [8], [9], [10], [11] and [12]. The intuition and core ideas have not changed and are transparent.

The **occam** community does not need any convincing that CSP is extremely simple and elegant: it well understands that this theoretical underpinning is responsible for the simplicity and power of **occam**. *CSPP* aspires to have the same simplicity and elegance, and to achieve that in the same way: by being properly defined by a mathematical theory. One motivation for its development was to provide the same sort of theoretical underpinning for future **occam**-like languages for co-design.

As with standard CSP, the routine use of the language is quite simple. But setting up a rigorous mathematical theory which covers all the obscure corners is a substantial task. Yet without that theory, we cannot be certain that there is no serious unrecognised trouble waiting to cause disaster.

The mathematical theories underlying process algebras are generally of three sorts: algebraic, operational and denotational. The characteristic flavour of CSP derives from the fact that the first full semantic models were denotational rather than operational: algebraic semantics also played an important role in the development as is evident in [1]. The denotational theories are still regarded as canonical, even though operational semantics have subsequently been developed. [2] gives a comprehensive account of many theories and their interplay.

Denotational semantic theories generally take the form of a mapping from syntax into a mathematical description of behaviour: behaviour which, in principle, is observable. The sorts of observation are not always carefully examined, but in CSP that is usually taken seriously, and can also be regarded as one of the roots of its power and simplicity. The companion paper, [13], is an examination of the observational model underlying *CSPP*.

The scrutiny of the observational basis of *CSPP* has been fruitful: the scope of the language has been extended far beyond systems that exhibit priority or neutrality. The language can now describe almost any system that can be understood to communicate what events it is prepared to perform in any given situation as described in [13]. One might question the relevance of this power in the context of the original design aims for *CSPP*: a foundation for concrete language design. The answer is that the aims have been widened: CSP is probably used mainly to describe and explore the properties of systems which have not themselves

been designed with CSP. *CSPP* may eventually be able to play the same role for a wider class of behaviour.

The explicit examination of the observational model also helped elucidate the various denotational approaches that have been explored for *CSPP*. It shows how much detail should be recorded in order to achieve a particular level of abstraction in the resulting dialect. Some readers will see the analogy with, say, the traces and failures models for standard CSP.

This paper presents the most abstract version of *CSPP* based on these developments. The main components of the underlying theoretical model are included.

1.1 The basic ideas

Readers familiar with *CSPP* may wish to omit this introductory material. There are three elementary processes in CSP and so also in *CSPP*. *Stop* is the process which does nothing at all, not even terminate. It is usually an error such as deadlock, but as will be abundantly apparent below, it appears very frequently in examples because it so simple.

Skip is the second of the elementary processes and represents termination. In the semantics of CSP, *Skip* is treated as a process which autonomously performs a special event '✓'. But in the semantics of *CSPP*, ✓ is treated as a 'token' rather than as a first class event: it is never 'performed' by a process, instead it is regarded as a signal to the external world. In *CSPP*, it is rather appropriate to say that '*Skip* does nothing, but does it successfully'.

The last process is written as div in CSP and in all except the most recent presentations of *CSPP*. We have recently taken to calling it *Spin* because there are technical connotations of div as the most undefined process which are inappropriate in *CSPP*. But whatever it is called, it represents livelock, a process that is in an uncontrollable sequence of internal actions, usually a loop. In the companion paper, [13], it is explained that in the semantics of *CSPP*, but not in CSP, we introduce a second 'token', which we write as ✗, which we pretend can be 'seen' by an external observer in an ideal infinite experiment.

Stop does nothing at all, not even terminate.
Skip terminates: hands control on to successor.
Spin does nothing at all externally: but it is active internally. Called div in CSP.

Prefixing provides a simple way to build more complex processes from *Stop*, *Skip* and *Spin*. So

- $a \rightarrow$ *Stop* is a process that performs the event *a* before stopping;
- $a \rightarrow$ *Skip* performs the event *a* before terminating; and
- $a \rightarrow$ *Spin* does *a* before typically looping internally for ever.

Another simple way to define a process using prefixing is to write $P = a \rightarrow P$. It is easy to see that this defines a process that will engage in an endless sequence of *a* events. It is also pretty clear that $Q = a \rightarrow Q$ must define the same process. But can we be sure? We may wonder exactly how we might establish that. What about $P = P$? This does not impose any constraint on *P* except the implied assumption that *P* is a process: it is evident that *every* process is a solution. These recursive definitions are at two extremes of a spectrum: one clearly defines a unique process, and the other is satisfied by any process. One may worry about intermediate cases, especially when the recursive equations get complicated. How do we know when there are solutions? Is there some special solution that we should pick when

there are several? We can't answer such questions properly without some sort of underlying theory: the more technical later parts of this paper describe one approach.

Note that our examples of recursion above could all be written in the form $P = f(P)$ where f is a function from processes to processes. In the context of $\mathcal{CSPP}$, we might write its type as $f : \mathcal{CSPP} \to \mathcal{CSPP}$. So that $P = a \to P$ matches $f(P) = a \to P$. Solutions to $f(P) = P$ are *fixed points*.

There is a standard mathematical notation for a fixed point of a function f: it is μf or $\mu P \bullet f(P)$. CSP and $\mathcal{CSPP}$ use the same notation. Thus the solution of $P = a \to P$ is written $\mu P \bullet a \to P$. Perhaps the most important task in building a theory for any sort of CSP is establishing that fixed points exist with the right properties. Almost all non trivial uses of CSP involve recursion.

Abstraction in the sense of being able to ignore irrelevant or internal detail is a crucial tool for humans. CSP and its derivatives have the powerful notion of *hiding*, examined at length in [12]. The notation is $P \setminus H$ representing a process P where events in the set H are hidden: events from H performed by P become internal, invisible to, and thus uncontrollable by, the outside world. It should be no surprise then that

$$(\mu P \bullet a \to P) \setminus \{a\} = Spin \ ,$$

which shows how these operations interlock.

There are only 4 basic operations of CSP left. The first is a generalisation of prefixing. $(a \to P_a) \,\Box\, (b \to P_b)$ is a process which is prepared either to perform the event a and then behave like P_a or to perform b and then behave as P_b. This is where the nature of events as joint actions between partners must be covered. A process P is understood as interacting with its *environment*, primarily by engaging in events which require the cooperation of both parties. Thus $a \to Stop$ cannot perform the event a until such time as the environment cooperates. This single concept covers a typical outermost passive observer who is willing to observe any event as well as parallel partners who may block certain events. The idea of blocking, that is synchronised, communication enters into the semantics of the parallel operators in a crucial way. The real point however is to capture the meaning of a process by external interaction with an agent as in [13].

The choice between a and b in $Q = (a \to P_a) \Box (b \to P_b)$ is controlled by the environment in the sense that if it chooses one or the other that fixes the subsequent the behaviour of Q. Of course Q blocks the environment if that only offers another event c initially. For these reasons $\Box$ is called external choice. The next operation arises from hiding and external choice and demonstrates more interlocking:

$$((a \to c \to Stop) \,\Box\, (b \to d \to Stop)) \setminus \{a, b\} = (c \to Stop) \sqcap (d \to Stop) \ .$$

$\sqcap$ is called internal choice. The environment of the process above has no control over whether the process is prepared to perform c or d: that was determined by whether the event a or b happened internally. In general, $P_1 \sqcap P_2$ may behave like either P_1 or like P_2, but the environment has no influence over the choice.

The remaining ways of composing processes are to place them in parallel, or run them in sequence. $P_1 \underset{E}{\|} P_2$ is the parallel composition where any events in the set E must be jointly performed by P_1 and by P_2: any other events are interleaved. $P_1 \fatsemi P_2$ is the process which behaves like P_1 until such time, if ever, that it terminates. If P_1 does terminate, subsequently $P_1 \fatsemi P_2$ behaves like P_2. Thus $(a \to Skip) \fatsemi (b \to Stop) = a \to b \to Stop$.

$a \to P$ performs a when the environment is willing, and then behaves like P.

$\mu P \bullet f(P)$ covers recursion.

$P \setminus H$ has the events in H hidden, internalised.

$P_1 \Box P_2$ is external choice: controlled by the environment.

$P_1 \sqcap P_2$ is internal choice.

$P_1 \underset{E}{\|} P_2$ is parallel composition with synchronisation set E.

$P_1 \fatsemi P_2$ is sequential composition.

1.2 CSPP and acceptances

Can $N = (a \to Stop) \Box (b \to Stop)$ favour a rather than b? Ordinary CSP abstracts away from such issues. The observations on which its semantics are based do not collect sufficient information to answer the question. Both Fidge's approach in [14] and Lowe's in [15] use order relations which can determine that a has greater priority than b.

CSPP is based on a more fundamental observational approach called acceptances. It examines environmental offers and their consequences as described in [13]. But the idea is extremely simple: if we want to know whether N prefers a rather than b, we offer the process both simultaneously, and see which it chooses. The offer is thus $\{a, b\}$ and if the implementation of N is biased in favour of a, it will reply with $\{a\}$. Consequently *CSPP* has a prioritised version of external choice and

$$AB = (a \to Stop) \overleftarrow{\Box} (b \to Stop)$$

is the process that always gives priority to a. The semantics is captured in the response to offers as $\{a, b\} \rightsquigarrow \{a\}$, $\{a\} \rightsquigarrow \{a\}$, $\{b\} \rightsquigarrow \{b\}$, and $\emptyset \rightsquigarrow \emptyset$ which we can summarise as $X \rightsquigarrow \{a\} \blacktriangleleft a \in X \blacktriangleright X \cap \{b\}$. There is really only this one simple idea in *CSPP* and its defining acceptance semantics. Everything else is just following through the consequences.

Notice that AB is a possible implementation of N in that it is natural to think of N as defined by $\{a\} \rightsquigarrow \{a\}$, $\{b\} \rightsquigarrow \{b\}$ and $\emptyset \rightsquigarrow \emptyset$. The response to $\{a, b\}$ is left partially open: we know that at least one or other of a and b is possible, and AB is one way of satisfying that requirement. Thus AB is better defined than N and its behaviour we suppose is one of those of N. That is *AB refines N* which is written $N \sqsubseteq AB$. N might be regarded as a specification and AB a possible implementation.

We must now come clean and say that there is more than one way of interpreting N in *CSPP*. But to understand that, we need to introduce neutral or *compliant* processes. As first pointed out by Bill Roscoe, most CSP practitioners who do not use **occam** think of N as a neutral process. So far we have only produced a biased version of $\Box$. This is where *CSPP* begins to show that it has a far wider scope than just priority. What is the response of a neutral version of N to the offer $\{a, b\}$? The answer is $\{a, b\}$ again: it is happy to do either. So *CSPP* also has $\overleftrightarrow{\Box}$ which can be used to write

$$S = (a \to Stop) \overleftrightarrow{\Box} (b \to Stop)$$

with $\{a, b\} \rightsquigarrow \{a, b\}$, $\{a\} \rightsquigarrow \{a\}$, $\{b\} \rightsquigarrow \{b\}$, and $\emptyset \rightsquigarrow \emptyset$. As noted in [13], it is *incoherent* to have a theory of priority which does not also cover compliance. This is because priority is

only of relevance when an environment offers more than one choice of event simultaneously. But what is it that can offer several events? In a theory that only includes biased processes, the answer is nothing. In $\mathcal{CSPP}$, the simple answer is, of course, a compliant process. Since the theory extends beyond simple compliance and simple priority, a more correct answer is a process which is at least partially compliant.

The idea of compliance carries with it the notion of *environmental* or *external* nondeterminism. If an environment offers a choice of $\{a, b\}$, it is nondeterministic about which is to be performed. Just as a compliant process is nondeterministic about which event the environment selects from its response. $\mathcal{CSPP}$ thus distinguishes *internal nondeterminism* which arises mainly from $\sqcap$ and *external nondeterminism* which is necessary in order for priority to be meaningful. Indeed, priority is precisely a means of resolving external nondeterminism.

These ideas percolate through many other operators. Although we have not introduced it above, standard CSP has a more general form of prefixing in which $e : E \to P(e)$ is a process which is prepared to do any of the events in the set E and then behave like the corresponding process $P(e)$ matching the particular event e selected. This is really a form of external choice, so there is a variety of possibilities for preference and lack of preference among the members of E. A simple and useful case is when all the members of E are available compliantly. That is written $e : \overleftrightarrow{E} \to P(e)$.

We take a simple example, $Q = e : \overleftrightarrow{E} \to Stop$, to illustrate how we define the semantics precisely. As in [13], Σ represents a universe of all events that we may wish to consider. We assume here that $E \subseteq \Sigma$, that is, E does not contain the tokens $\checkmark$ or $\times$. In $\mathcal{CSPP}$ it is sometimes convenient to permit those tokens to appear in a prefix set. Before Q has performed any events, the only trace is $\langle\rangle$. Then an offer $X \subseteq \Sigma$ evokes the response $X \cap E$ which is compliant when more than one member of E is in X. We write that as $\langle\rangle : X \rightsquigarrow X \cap E$. The only traces that can be performed thereafter are of the sort $\langle e\rangle$ and then the process refuses everything: $\langle e\rangle : X \rightsquigarrow \emptyset$. We specify all this precisely as the set

$$\begin{aligned} \mathcal{B}(e : \overleftrightarrow{E} \to Stop) = \quad & \{(\langle\rangle, X, X \cap E) \,|\, X \subseteq \Sigma\} \\ & \cup \\ & \{(\langle e\rangle, X, \emptyset) \,|\, e \in E \wedge X \subseteq \Sigma\} \ . \end{aligned} \tag{1}$$

$\mathcal{B}$ is the mapping from syntax to a set of behaviours, here triples: this is a denotational semantics. It completely specifies the meaning of $e : \overleftrightarrow{E} \to Stop$. All the definitions later in this paper follow this style.

The other place where priority has a major effect is in the definition of the parallel operations. In $P_1 \underset{E}{\|} P_2$, P_1 and P_2 may themselves express priority, but we are more concerned here with the parallel constructor itself. Synchronised events in E are joint actions of P_1 and P_2 and their selection involves the preferences of the partners. But external nondeterminism arises when both P_1 and P_2 offer unsynchronised events: the situation is exactly the same as for external choice. A parallel operator might always favour P_1: so $\mathcal{CSPP}$ includes $\underset{E}{\overleftarrow{\|}}$. One can think of $\underset{E}{\overleftarrow{\|}}$ as behaving like $\overleftarrow{\Box}$ on unsynchronised interleaved events, but on every event rather just the initials. The compliant version, $\underset{E}{\overleftrightarrow{\|}}$, is likely to match the intuition of most CSP practitioners. But notice that because CSP abstracts away from preferences, one should really always make the most nondeterministic identification. Thus $\underset{E}{\|}$ in $\mathcal{CSPP}$ can have any sort of bias or lack thereof and behave quite differently in that respect from event to event. This is in the spirit of CSP as an abstraction, but in practical applications of $\mathcal{CSPP}$ one nearly always chooses the simpler versions like $\underset{E}{\overleftrightarrow{\|}}$. It should be mentioned that Lowe in [15] uses

a very restricted set of parallel operators with a very different semantics from $\mathcal{CSPP}$ on the synchronised events.

To summarise, $P_1 \overset{\leftrightarrow}{\underset{E}{\|}} P_2$ is simple and the most used parallel operator in $\mathcal{CSPP}$. $P_1 \overset{\leftarrow}{\underset{E}{\|}} P_2$ is occasionally useful. $P_1 \underset{E}{\|} P_2$ is extremely chaotic in the sense that it permits any sort of resolution or lack of resolution of external nondeterminism. When soft priority, see below in section 3, is in force, $\underset{E}{\|}$ can be identified with the corresponding CSP operator.

1.3 Internal and external nondeterminism

The major insight in $\mathcal{CSPP}$ and acceptances is in distinguishing external from internal nondeterminism. We have seen that external nondeterminism is captured in multiple events present in either an offer or a response (in the pairs $X \rightsquigarrow Y$).

Internal nondeterminism is captured in the usual way: by recording multiple behaviours. Thus $Ex = (a \to Stop) \overset{\leftrightarrow}{\Box} (b \to Stop)$ and $In = (a \to Stop) \sqcap ((b \to Stop)$ are represented by the sets

$$\mathcal{B}(Ex) = \begin{array}{c} \{(\langle\rangle, X, X \cap \{a,b\}) \,|X \subseteq \Sigma\} \\ \cup \\ \{(\langle x\rangle, X, \emptyset) \,|x \in \{a,b\} \wedge X \subseteq \Sigma\} \end{array}$$

and

$$\mathcal{B}(In) = \begin{array}{c} \{(\langle\rangle, X, X \cap \{a\}) \,|X \subseteq \Sigma\} \\ \cup \\ \{(\langle\rangle, X, X \cap \{b\}) \,|X \subseteq \Sigma\} \\ \cup \\ \{(\langle x\rangle, X, \emptyset) \,|x \in \{a,b\} \wedge X \subseteq \Sigma\} \end{array} = \mathcal{B}(a \to Stop) \cup \mathcal{B}(b \to Stop) \ .$$

In general, internal nondeterminism is represented by the union of behaviours:
$\mathcal{B}(P_1 \sqcap P_2) = \mathcal{B}(P_1) \cup \mathcal{B}(P_2)$.

1.4 Fixed points: the heart of a denotational semantics.

We have seen that CSP is elegant and sparse with very few operators. $\mathcal{CSPP}$ aims for the same qualities, but the inclusion of priority means that some variant decorated versions of the standard operators are needed. In both cases, recursion is the only way to produce non trivial programs: all the other syntax only gives us finite length traces.

Accordingly recursion is at the heart of both theories. It is essential that $\mu P \bullet f(P)$ is well defined, and any restrictions on the function f which may be used are identified. Let $P = a \to b \to P$. Successively unwinding this recursion as $P = a \to b \to a \to b \to P$, $P = a \to b \to a \to b \to a \to b \to P, \ldots$ can be thought of a sequence of more precise specifications for P. $P = a \to b \to P$ might potentially be satisfied by any process that at least starts off like $a \to b \to \ldots$. Next we admit the subset of those processes that start with $a \to b \to a \to b \to \ldots$. But these are successive *refinements*. Each unwinding is getting *nearer* to the final fixed point (if there is one). We are 'converging' towards a solution. We can think of this as a succession of successive approximations $A_0 \sqsubseteq A_1 \sqsubseteq A_2 \ldots$. So it is no surprise to discover that we use the refinement partial order as a way to establish that recursions are properly defined. Indeed this is usually the major task in setting up any denotational semantics.

In ordinary convergence, there must be limit points available to which a sequence can converge: that is the space is complete. In partial order theory, the corresponding property of

the space is that it be a Complete Partial Order (CPO). Refinement is a partial order, and we prove in section 8.15 that the variant of $\mathcal{CSPP}$ below is a CPO under a certain condition. It then follows from standard theorems that functions that are monotone in the refinement order have a least fixed point. We show below that all the $\mathcal{CSPP}$ operators are indeed monotone. If a function is in addition continuous (a partial order analogue of ordinary continuity), then the fixed point of a function can be calculated in a simple way which is essentially the same as we did above with the sequence $A_0, A_1, A_2, \ldots$. Background can be found in [16] and [2] among other places.

1.5 Differences from earlier versions of $\mathcal{CSPP}$

Earlier versions of the semantics allowed ✓ and ✗ to appear in a response mixed in with ordinary events. So this appeared to offer an environment the possibility of making a choice that might in effect steer a process either towards or away from termination or livelock. Both tokens represent situations that are normally regarded as being outside environmental control, so in the current versions, these tokens may only appear as singletons. In effect, external nondeterminism in these tokens is converted into internal nondeterminism by this change. It only affects some rather obscure situations.

As a result of the examination of the observational status documented in [13], the axioms have been significantly weakened, and the scope of the language very considerably extended. There are now no preconceived notions of how a 'good' process should behave: there is an exception in section 8.16.4. The language is now based simply on observing what a system, however bizarre, does. We only require that responses are reliable. There is some cost in extra complexity of some of the semantic definitions, at least in the most abstract version below.

There is a little freedom left in two areas: how much detail should be recorded in observations which determines the level of abstraction of the resulting dialect; and whether priority conflict results in deadlock or is resolved nondeterministically.

For this paper, both choices have been made on the grounds of simplicity of the underlying semantics. That implies maximum abstraction and hard priority: conflict results in deadlock. The first choice leads to a dialect which is arguably in the spirit of the Failures models for standard CSP, but is a little too abstract for the author's taste. The second is in the spirit of **occam**, but complicates the congruence with standard CSP.

2 An abstract variant of $\mathcal{CSPP}$

The companion paper, [13], examines the sorts of observations which can be made of processes that can usefully participate in systems that employ priority. Any process that can be regarded as capable of reacting to an environmental offer with a declaration of which events can be accepted, and then reliably and jointly performing one of the accepted events is included. This is a far wider class then just those that employ a straightforward notion of priority.

This version of the language includes

$$\begin{aligned} P \quad ::= \quad & \mathit{Stop} \mid \mathit{Skip} \mid \mathit{Spin} \mid a \to P \mid e : E \to P(e) \mid e : (\alpha) \to P(e) \mid P \fatsemi P \\ & \mid P \sqcap P \mid P \overleftarrow{\Box} P \mid P \overleftrightarrow{\Box} P \mid P \underset{E}{\overleftarrow{\parallel}} P \mid P \underset{E}{\overleftrightarrow{\parallel}} P \mid P \setminus H \mid \mu P \bullet f(P) \mid P[\![\mathsf{R}]\!] \,. \end{aligned}$$

$e : (\alpha) \to P(e)$ is a relational form of prefixing, and provides a syntax for capturing some very irregular processes that are not described by either a priority or absence of a priority: α is a relation. There is no $\top$ process in this version of $\mathcal{CSPP}$: it is useful in less abstract

versions of the language. *Spin* is usually called div in standard CSP: it is not the bottom of the refinement order here, so the name has been changed to avoid confusion.

The theory below uses *hard* priority for simplicity. The price to be paid is that there is no unique identification of standard external choice $\Box$ which accordingly does not appear in the syntax above. There are two maximal, but incompatible, identifications for $\Box$:

$$P \Box Q = P \overleftrightarrow{\Box} Q \sqcap P \overleftarrow{\Box} Q \quad \text{or} \quad P \Box Q = P \overleftrightarrow{\Box} Q \sqcap P \overrightarrow{\Box} Q \,.$$

Both, of course, are compatible with **occam**. Similarly, there is no unique identification for $\|$. However a $\underset{E}{\|}$ is defined below but it should not normally be identified with the corresponding standard operator. In most circumstances, the standard operator would be identified with the compliant version $\underset{E}{\overleftrightarrow{\|}}$.

The semantics below is all straightforward but because some very irregular processes are admitted few assumptions can be made. It becomes necessary to spell out some conditions in detail. Hiding is by far the most difficult and subtle operator, and the possibility of irregular behaviour needs careful consideration. The existence of least fixed points to define recursion is established for a finite alphabet by showing that $\mathcal{CSPP}$ is a Complete Partial Order (CPO) under refinement. All operators are monotone, and most distribute over $\sqcap$. There is also a metric so the usual Unique Fixed Point (UFP) theorems hold.

The purpose of this paper is to demonstrate that there is a very abstract version of $\mathcal{CSPP}$ with a semantics closely analogous to the standard Failures-Divergences model for CSP.

3 Hard and Soft priority.

Consider

$$\Big((a \to \mathit{Stop}) \overleftarrow{\Box} (b \to \mathit{Stop})\Big) \| \Big((a \to \mathit{Stop}) \overrightarrow{\Box} (b \to \mathit{Stop})\Big) \,.$$

What happens when the environment offers $\{a, b\}$? Soft priority somehow resolves the conflict: some versions of $\mathcal{CSPP}$ allow the system to 'search' for the largest sub-offers on which the two processes can agree. When there is more than one such sub-offer, there is a non deterministic choice between the options.

However, hard priority is simpler, and arguably more in the spirit of **occam**. In this case, if the two processes cannot agree, the result is deadlock. A difficulty is that $\overleftarrow{\Box}$ and $\overrightarrow{\Box}$ cannot simultaneously refine $\Box$ since $((a \to \mathit{Stop}) \Box (b \to \mathit{Stop})) \| ((a \to \mathit{Stop}) \Box (b \to \mathit{Stop}))$ cannot deadlock.

4 Alphabets and traces

There is a global alphabet Σ of ordinary events: this will be large enough to include all the visible events of any process that we need to describe. Later we will need to restrict it to be finite. We add the pseudo events ✓ and ✗ which only occur as singleton responses.

Traces are simply sequences, empty or finite, of events drawn from Σ. $\langle\rangle$ is the empty sequence. The set of all finite traces drawn from Σ is written as Σ^*. Because we need $\langle\rangle$, it is convenient to label the elements of the other traces from 1: such a trace t has type $t : \{1, 2, \ldots, n\} \to \Sigma$ and we can write $t = \langle t(1), t(2), \ldots, t(n)\rangle$. In this context, it is sometimes useful to implicitly identify $\langle\rangle$ with the empty function $\emptyset$: this is used below, for example in definition 4 on page 166.

$t \frown s$ represents the concatenation of two sequences, so $\langle a, b\rangle \frown \langle c, d\rangle = \langle\rangle \frown \langle a, b, c, d\rangle = \langle a, b, c, d\rangle = \langle abcd\rangle$: we may omit commas when there is no ambiguity.

We also make use of infinite traces which are of the sort $s : \mathbb{N}_1 \to \Sigma$, but only to express certain properties of *finite* traces.

5 Acceptances

As described in the companion paper, we specify the meaning of a process by describing its responses after it has performed some trace of events. Traces are members of Σ^*. Given such a trace s, we record the response Y to an offer X. We sometimes write that as $X \rightsquigarrow Y$ or $s : X \rightsquigarrow Y$ as an equivalent way of expressing the triple (s, X, Y). The offer X is some subset of the alphabet Σ: that is $X \subseteq \Sigma$. And a response Y is a subset of Σ or $\{\checkmark\}$ or $\{\text{✗}\}$.

Definition 1

- *When $X \subseteq \Sigma$, $X^{\checkmark}$ denotes $X \cup \{\checkmark\}$ and $X^{\checkmark\text{✗}}$ denotes $X \cup \{\checkmark, \text{✗}\}$.*
- *The lone sets of X are* $\overset{\checkmark\text{✗}}{\mathbb{L}}(X) \equiv \{\{\checkmark\} \mid \checkmark \in X\} \cup \{\{\text{✗}\} \mid \text{✗} \in X\}$.
- $\overset{\checkmark\text{✗}}{\mathbb{S}}(X) \equiv \{\emptyset\} \blacktriangleleft X = \emptyset \blacktriangleright \overset{\checkmark\text{✗}}{\mathbb{L}}(X) \cup \{X \cap \Sigma \mid X \cap \Sigma \neq \emptyset\}$
- $\overset{\checkmark\text{✗}}{\mathbb{P}}\, X$ *denotes* $\mathbb{P}(X \cap \Sigma) \cup \overset{\checkmark\text{✗}}{\mathbb{L}}(X)$.
- $\overset{\checkmark\text{✗}}{\mathbb{M}}(X)$ *denotes* $\{\emptyset\} \blacktriangleleft X = \emptyset \blacktriangleright \left(\overset{\checkmark\text{✗}}{\mathbb{P}}(X) - \{\emptyset\}\right)$.

Acceptances have type $\Sigma^* \times \mathbb{P}\,\Sigma \times \overset{\checkmark\text{✗}}{\mathbb{P}}\,\Sigma^{\checkmark\text{✗}}$, which we occasionally identify with

$$\Sigma^* \times \left(\mathbb{P}\,\Sigma \times \overset{\checkmark\text{✗}}{\mathbb{P}}\,\Sigma^{\checkmark\text{✗}}\right) . \tag{2}$$

More specifically, acceptances are members of

$$\mathcal{A} = \{(s, X, Y) \mid s \in \Sigma^* \wedge X \subseteq \Sigma \wedge Y \in \overset{\checkmark\text{✗}}{\mathbb{P}}(X^{\checkmark\text{✗}})\} \tag{3}$$

Thus we have a description which is a set $\mathcal{B}\,P$ of such acceptances $\mathcal{B}(P) \subseteq \mathcal{A}$. $[\![P]\!]$ is the usual notation for the semantic function describing the behaviour of the process P, but $\mathcal{B}\,P$ is more intuitive here and avoids confusion with the renaming of equation (23) on page 178.

The set of traces of the process is

$$traces(P) = \{s \mid \exists X, Y \bullet (s, X, Y) \in \mathcal{B}\,P\} \quad \text{or} \quad traces(P) = \operatorname{dom} \mathcal{B}\,P$$

when the type of $\mathcal{B}\,P$ is taken from (2).

We identify a process directly with its acceptances where the context warrants.

5.1 Acceptances determine traces

There are often simple patterns for triples (s, X, Y) representing a process P for a general s. That is, when $s \in \mathit{traces}(P)$ then the pattern $(s, X, Y) \in \mathcal{B}(P)$. It is sometimes intricate, error prone and laborious to spell out explicitly which s is a trace of P. But that information is already implicit in the acceptances, determined by the events in Y.

Thus, take $B \subseteq \mathcal{A}$ and define

$$\begin{array}{lcl} B_0 & = & \{(\langle\rangle, X, Y) \in B\} \\ B_{n+1} & = & \{(s^\frown\langle e\rangle, X, Y) \in B \mid \exists X', Y' \bullet (s, X', Y') \in B_n \wedge e \in Y'\} \quad \text{for } n \in \mathbb{N}. \end{array} \tag{4}$$

The we define the inductive core, $\lfloor B \rfloor$ by

$$\lfloor B \rfloor \equiv \cup\{B_n \mid n \in \mathbb{N}\} \,. \tag{5}$$

Since all traces are finite, when $\mathcal{B}(P) \subseteq B$, then $\mathcal{B}(P) = \lfloor B \rfloor$. In this context we can say that acceptances determine the traces.

6 Some definitions

$\#s$ is the length of s: $\#\langle\rangle = 0$ and $\#(t^\frown\langle e\rangle) = \#t + 1$. We extend this notation to acceptances: $\#A = \max\{\#s \mid s \in \mathit{traces}(A)\}$ which is well defined when the lengths are bounded. Otherwise $\#A = \infty$.

$s \downharpoonright n$ just truncates a trace to a length no more than n whether s is finite or infinite:

Definition 2 $s \downharpoonright 0 = \langle\rangle$ *and* $s \downharpoonright n = \mathbf{n} \lhd s$ *where* $\mathbf{n} = \{i \mid 1 \le i \le n\}$.

Here $\lhd$ is domain restriction: if $f : X \to Y$ is a function then $D \lhd f : X \cap D \to Y$ is the restriction of $f_{\mid D \cap X}$ with domain $X \cap D$. So $\#(s \downharpoonright n) \le n$.

If S is a set of traces $S \downharpoonright n$ contains only the truncated versions:

Definition 3 $S \downharpoonright n = \{s \downharpoonright n \mid s \in S\}$.

We extend prefixing to allow comparison of a finite with an infinite trace:

Definition 4 *Let* $s \in \Sigma^*$ *and* $t \in \Sigma^\omega$. *Then*

$$(s \le t) \;\Leftrightarrow\; (s = \langle\rangle \;\vee\; \exists n \in \mathbb{N} \bullet s = \mathbf{n} \lhd t)$$

$\lhd$ above is domain restriction as before: the leading n elements of t match a non-null s above.

Definition 5 $s \setminus H$ *is a trace composed of those elements not in the set* H*:*

$$\langle\rangle \setminus H = \langle\rangle \qquad s^\frown\langle x\rangle \setminus H = s \setminus H \blacktriangleleft x \in H \blacktriangleright (s \setminus H)^\frown x$$

We can extend trace hiding to infinite traces.

Definition 6 *If* $w \in \Sigma^\omega$ *then we write* $w \setminus H = s$ *when*

$$\forall n \in \mathbb{N} \bullet (w \downharpoonright n) \setminus H \le s \quad \textit{and} \quad \forall n \in \mathbb{N} \bullet \exists m \in \mathbb{N} \bullet s \downharpoonright n = (w \downharpoonright m) \setminus H \,.$$

Definition 7 *The down-set* $\downarrow s = \{t \mid t \le s\}$ *is a standard notion from partial order theory.*

If $A : \mathbb{P}\left(\Sigma^* \times \mathbb{P}\Sigma \times \overset{\checkmark\!\!\times}{\mathbb{P}}\Sigma^{\checkmark\!\!\times}\right)$ is a set of acceptances then $A \Downarrow n$ just represents the acceptances no longer than n:

Definition 8

$$A \Downarrow n = \{(s, X, Y) \in A \mid \#s \leqslant n\} \,. \tag{6}$$

The notation $R(\!|X|\!)$ means the relational image of a set X. So:

$$R(\!|X|\!) = \{y \mid x \in X \ \wedge \ x\, R\, y\} \,.$$

Definition 9 *Let $(S, \leqslant)$ be a partial order. $\mathbf{m}_{\leqslant} : \mathbb{P}\, S \to \mathbb{P}\, S$ is the function that produces the set of maximal elements:*

$$\mathbf{m}_{\leqslant}(\emptyset) = \emptyset$$
$$\mathbf{m}_{\leqslant}(X) = \{x \in X \mid \forall\, y \in X \bullet x \not< y\} \quad (X \neq \emptyset) \,.$$

Usually the order is implied and we write $\mathbf{m} : \mathbb{P}\, S \to \mathbb{P}\, S$.

Definition 10 *An* atomic *process is one represented by a minimal non empty set of acceptances: any smaller non empty set cannot represent a process. Such processes have a unique response to any offer: they are* internally deterministic.

7 Axioms

The axioms here were briefly described in the companion paper and shown to encompass a very large class of processes. For a set $\mathcal{B}\, P \subseteq \mathcal{A}$ to describe a process, it must conform to these simple constraints.

Every process performs events only after it has started:

H1: $\quad \langle\rangle \in \mathit{traces}(P)$

There is at least one acceptance for every possible offer:

H2: $\quad \forall\, s \in \mathit{traces}\,(P) \bullet \forall\, X \subseteq \Sigma \bullet \exists\, Y \in \overset{\checkmark\!\!\!\times}{\mathbb{P}}(X^{\checkmark\!\!\!\times}) \bullet (s, X, Y) \in \mathcal{B}\,(P)$

The traces of a process are prefix closed, and match the responses:

H3: $\quad \forall\, s \in \Sigma^{*} \bullet \forall\, x \in \Sigma \bullet \ s^{\frown}\langle x\rangle \in \mathit{traces}(P) \ \Leftrightarrow \ \exists (s, X, Y) \in \mathcal{B}\,(P) \bullet x \in Y \cap \Sigma$

One may regard $\mathcal{B}\, P \subseteq \mathcal{A}$ and properties like $\checkmark \ \neq \ \times$ and $\Sigma \cap \{\checkmark, \times\} = \emptyset$ as implicit axioms.

8 Semantics

The main purpose of this paper is to present the details of the semantics of the principal operators of *CSPP*.

8.1 Stop, Skip and Spin

Stop, *Skip* and *Spin* are all similar:

$$\begin{array}{lcl} \mathcal{B}\, \mathit{Stop} & = & \{(\langle\rangle, X, \emptyset) \mid X \subseteq \Sigma\} \\ \mathcal{B}\, \mathit{Skip} & = & \{(\langle\rangle, X, \{\checkmark\}) \mid X \subseteq \Sigma\} \\ \mathcal{B}\, \mathit{Spin} & = & \{(\langle\rangle, X, \{\times\}) \mid X \subseteq \Sigma\} \end{array} \tag{7}$$

8.2 $\perp$

$\perp$ is the most unpredictable of all processes: it can behave in any fashion at all.

$$\mathcal{B}\perp = \mathcal{A} \tag{8}$$

See equation 3 on page 165 for the definition of $\mathcal{A}$.

8.3 Prefix choice

Consider $e : E \to P(e)$ with $E \subseteq \Sigma^{\checkmark\!\!\times}$. In general there are many possible initial acceptances

$$\langle\rangle : X \rightsquigarrow \emptyset \blacktriangleleft X^{\checkmark\!\!\times} \cap E = \emptyset \blacktriangleright U$$

where $U \in \overset{\checkmark\!\!\times}{\mathbb{P}}\left(X^{\checkmark\!\!\times} \cap E\right)$ is not empty.

$$\begin{aligned}\mathcal{B}(e : E \to P(e)) = \quad & \left\{(\langle\rangle, X, Y) \,\middle|\, X \subseteq \Sigma \wedge Y \in \overset{\checkmark\!\!\times}{\mathbb{M}}\left(X^{\checkmark\!\!\times} \cap E\right)\right\} \\ & \cup \\ & \{(\langle e\rangle^\frown s, X, Y) \,|\, e \in E \cap \Sigma \wedge (s, X, Y) \in \mathcal{B}(P(e))\}\end{aligned} \tag{9}$$

$\overset{\checkmark\!\!\times}{\mathbb{M}}(X)$ was defined in definition 1 on page 165: the acceptances include all ways of assigning or refraining from assigning priority among the events of E. Subsequent behaviour, if any, depends on the initial event e and matches one of those in $\mathcal{B}P(e)$. Clearly, $P(e)$ is defined on $E \cap \Sigma$, at least.

Example 1 *Usually E in $e : E \to P(e)$ is a set of real events. More general cases include*

1. $x : \{\checkmark\} \to Stop = x : \{\checkmark\} \to P = Skip$ *for any* P.
2. $x : \{\times\} \to Stop = x : \{\times\} \to P = Spin$ *for any* P.
3. $x : \{\checkmark, \times\} \to P = Skip \sqcap Spin$ *for any* P.
4. $x : \{a, \checkmark\} \to Stop = (a \to Stop) \overset{\leftarrow}{\Box} Skip \;\sqcap\; Skip$.

When subsequent behaviour does not depend on the choice of event as in the examples above, it is natural to omit "$x :$" as in $\{a, b\} \to \{b, c\} \to Stop$.

8.4 Relational Prefix choice

It does not take much experience using *CSPP* syntax to see the utility of notations like $e : (S, [\{a, b\} > \{c\} > \{d\}]) \to P(e)$ to stand for $((a \to P(a)) \overset{\leftrightarrow}{\Box} (b \to P(b))) \overset{\leftarrow}{\Box} (c \to P(c)) \overset{\leftarrow}{\Box} (d \to P(d))$. More generally $e : (S, \leqslant) \to P(e)$ is a process which initially gives priority to events according to the partial order $\leqslant$ and then behaves like $P(e)$. One might expect that we can always find such a partial order to describe the initial acceptances, but this is not true.

Example 2

$$((a \to Stop) \overset{\leftarrow}{\Box} (b \to Stop)) \;\overset{\leftrightarrow}{\Box}\; ((c \to Stop) \overset{\leftarrow}{\Box} (b \to Stop) \overset{\leftarrow}{\Box} (a \to Stop))$$

has initial acceptances that do not match any partial order:
$\{a, b, c\} \rightsquigarrow \{a, c\}, \{b, c\} \rightsquigarrow \{b, c\}, \{a, c\} \rightsquigarrow \{a, c\}, \{a, b\} \rightsquigarrow \{a, b\}, \ldots$. *In such an order,* $\{a, b, c\} \rightsquigarrow \{a, c\}$ *would give* $b \leqslant a$, *but that does not match* $\{a, b\} \rightsquigarrow \{a, b\}$ *which would require that a and b are incomparable.*

This suggests that a more general form of prefixing may be desirable. The obvious, simple and natural approach is to directly specify an initial acceptance relation, although an acceptance function would be adequate. Relations used in practice are nearly always functions: such prefixes can be called atomic.

$$\mathcal{B}(e : (\alpha) \to P(e)) = \\ \{(\langle\rangle, X, Y) \in \mathcal{A} \mid X\, \alpha\, Y\} \cup \{(\langle e\rangle^\frown s, X, Y) \mid \exists (X', Y') \in \alpha \bullet e \in Y' \cap \Sigma \wedge (s, X, Y) \in \mathcal{B}(P(e))\} \tag{10}$$

where $\alpha : \mathbb{P}\left(\mathbb{P}\,\Sigma \times \overset{\checkmark\!\!\times}{\mathbb{P}}(\Sigma^{\checkmark\!\!\times})\right)$ is an acceptance relation with $\operatorname{dom}\alpha = \mathbb{P}\,\Sigma$ matching the axioms and respecting $\mathcal{A}$. $P(e)$ is understood to be defined for all the events that can be accepted by α.

The most common use of this notation is atomic. It is when the initial acceptances match a partial order relation $(S, \leqslant)$ on events in which every nonempty subset of S has a maximal element and ✓ or ✗ may be present only as bottom elements. In such cases

$$\mathcal{B}(e : (S, \leqslant) \to P(e)) = \\ \left\{(\langle\rangle, X, \mathbf{m}(S \cap X^{\checkmark\!\!\times})) \mid X \subseteq \Sigma\right\} \\ \cup \\ \left\{(\langle e\rangle^\frown s, X, Y) \mid \exists\, Z \subseteq \Sigma \bullet e \in \mathbf{m}(S \cap Z^{\checkmark\!\!\times}) \cap \Sigma \wedge (s, X, Y) \in \mathcal{B}(P(e))\right\} \tag{11}$$

$\mathbf{m} : \mathbb{P}\,\Sigma^{\checkmark\!\!\times} \to \mathbb{P}\,\Sigma^{\checkmark\!\!\times}$ is the function which selects the maximal elements of a set: see definition 9 on page 167.

Many orders of interest can be expressed in terms of *layers* as in $[\{p\} > \{q, r\} > \{s\}]$: the elements of the component sets are strictly ordered only with respect to members of other sets. This notation is used in some of the examples below:

Example 3 $x : (\{a, b\}, =) \to Stop = a \to Stop \overset{\leftrightarrow}{\Box} b \to Stop.$

Example 4 $x : ([\{a\} > \{b\}]) \to Stop = a \to Stop \overset{\leftarrow}{\Box} b \to Stop.$

Example 5
$x : ([\{a, b\} > \{c\} > \{d\}]) \to Stop = (a \to Stop \overset{\leftrightarrow}{\Box} b \to Stop) \overset{\leftarrow}{\Box} (c \to Stop \overset{\leftarrow}{\Box} d \to Stop).$

Example 6 $x : ([\{a, b\} > \{\checkmark\}]) \to Stop = (a \to Stop \overset{\leftrightarrow}{\Box} b \to Stop) \overset{\leftarrow}{\Box} Skip.$

All the examples above are cases when the '$x :$' might have been omitted for brevity: x was not explicitly bound in the bodies, which means that $P(e)$ is a constant function.

8.5 Compliant prefixing

$e : \overset{\leftrightarrow}{E} \to P(e)$, is defined as $e : (E, =) \to P(e)$.

8.6 Internal choice

$$\mathcal{B}(P_1 \sqcap P_2) = \mathcal{B}P_1 \cup \mathcal{B}P_2 \tag{12}$$

We extend the definition to suitable non empty sets of processes $\mathcal{P}$, writing

$$\mathcal{B}\left(\bigsqcap \mathcal{P}\right) = \bigcup \{\mathcal{B}P \mid P \in \mathcal{P}\}\,. \tag{13}$$

To avoid foundational issues, we need to impose a suitable limit on the size of the sets admitted by (13) to ensure the behaviours form a set rather than a proper class. It is then immediately clear that $\sqcap \mathcal{P}$ satisfies the axioms of section 7:

Lemma 1 *$\mathcal{B}(\bigcup \mathcal{P})$ represents a process when $\mathcal{P}$ is a suitable non empty set of processes.*

The following lemmas are not hard to prove:

Lemma 2 *Let $\{P_i : \Sigma \to \mathcal{CSPP}\}$ be an indexed nonempty set of process functions. Then $e : E \to \bigsqcap\{P_i(e) \mid i \in I\} = \bigsqcap\{e : E \to P_i(e) \mid i \in I\}$*

Lemma 3 *Let $\{P_i : \Sigma \to \mathcal{CSPP}\}$ be an indexed nonempty set of process functions. Then $e : (\alpha) \to \bigsqcap\{P_i(e) \mid i \in I\} = \bigsqcap\{e : (\alpha) \to P_i(e) \mid i \in I\}$*

Lemma 4 *Let $\{P_i : \Sigma \to \mathcal{CSPP}\}$ be an indexed nonempty set of process functions. Then $e : (S, \leqslant) \to \bigsqcap\{P_i(e) \mid i \in I\} = \bigsqcap\{e : (S, \leqslant) \to P_i(e) \mid i \in I\}$*

8.7 Compliant external choice

$$\mathcal{B}(P_1 \overleftrightarrow{\Box} P_2) = \left\{ (\langle\rangle, X, Y) \;\middle|\; \begin{array}{c} \exists\, Y_1, Y_2 \in \overset{\checkmark\text{✗}}{\mathbb{P}}(\Sigma^{\checkmark\text{✗}}) \bullet \\ (\langle\rangle, X, Y_1) \in \mathcal{B}P_1 \wedge (\langle\rangle, X, Y_2) \in \mathcal{B}P_2 \wedge Y \in \overset{\checkmark\text{✗}}{\mathbb{S}}(Y_1 \cup Y_2) \end{array} \right\} \cup \left\{ (\langle x\rangle^\frown s, X, Y) \,\middle|\, (\langle x\rangle^\frown s, X, Y) \in \mathcal{B}(P_1) \;\vee\; (\langle x\rangle^\frown s, X, Y) \in \mathcal{B}(P_2) \right\} \tag{14}$$

$\overset{\checkmark\text{✗}}{\mathbb{S}}(X)$ is defined in definition 1 on page 165. It ensures that ✓ and ✗ can be returned only as singletons if they are present.

Example 7 $(a \to P_1) \overleftrightarrow{\Box} (a \to P_2) = (a \to P_1) \sqcap (a \to P_2)$.

Example 8 $Skip \overleftrightarrow{\Box} (a \to P) = Skip \;\sqcap\; (a \to P) \overleftarrow{\Box} Skip$.

Although $Skip \overleftrightarrow{\Box} (a \to P)$ looks as if it ought to be compliant on $\langle\rangle$, an environment cannot choose ✓, so ✓ cannot be offered as a legitimate *choice*, only as a singleton. ■[8]

It is clear from the definition that $\overleftrightarrow{\Box}$ is associative:

$$P_1 \overleftrightarrow{\Box} \left(P_2 \overleftrightarrow{\Box} P_3\right) = \left(P_1 \overleftrightarrow{\Box} P_2\right) \overleftrightarrow{\Box} P_3$$

¿From standard CSP, we might expect that $P_1 \sqcap \left(P_2 \overleftrightarrow{\Box} P_3\right) = (P_1 \sqcap P_2) \overleftrightarrow{\Box} (P_1 \sqcap P_3)$ but this cannot be true in the presence of priority. The reason becomes clear when we regard the processes as specifications. $(P_1 \sqcap P_2) \overleftrightarrow{\Box} (P_1 \sqcap P_3)$ has a refinement of $P_1 \overleftrightarrow{\Box} P_3$ which is not an implementation of $P_1 \sqcap \left(P_2 \overleftrightarrow{\Box} P_3\right)$. The correct result is

$$P_1 \sqcap \left(P_2 \overleftrightarrow{\Box} P_3\right) \sqsupseteq (P_1 \sqcap P_2) \overleftrightarrow{\Box} (P_1 \sqcap P_3) \tag{15}$$

It is again not hard to establish:

Lemma 5 *Let* $\{P_i : \Sigma \to \mathcal{CSPP}\}$ *be an indexed nonempty set of processes. Then* $Q \overleftrightarrow{\Box} \bigsqcap\{P_i \mid i \in I\} = \bigsqcap\{Q \overleftrightarrow{\Box} P_i \mid i \in I\}$

8.8 Prioritised external choice

$$\begin{array}{l}\mathcal{B}(P_1 \overleftarrow{\Box} P_2) = \\ \qquad \left\{(\langle\rangle, X, Y) \;\middle|\; \begin{array}{c}\exists\, Y_1, Y_2 \subseteq \Sigma^{\checkmark\!\!\!\times} \bullet (\langle\rangle, X, Y_1) \in \mathcal{B}\,P_1 \wedge (\langle\rangle, X, Y_2) \in \mathcal{B}\,P_2 \\ \wedge \\ Y = (Y_2 \lhd Y_1 = \emptyset \rhd Y_1)\end{array}\right\} \\ \qquad\qquad \cup \\ \left\{(\langle e\rangle^\frown s, X, Y) \;\middle|\; \begin{array}{c}(\langle e\rangle^\frown s, X, Y) \in \mathcal{B}(P_1) \\ \vee \\ \exists\, X' \subseteq \Sigma \bullet (\langle\rangle, X', \emptyset) \in \mathcal{B}(P_1) \wedge \exists\, Y_2 \subseteq \Sigma^{\checkmark\!\!\!\times} \bullet \\ (\langle\rangle, X', Y_2) \in \mathcal{B}(P_2) \wedge e \in Y_2 \cap \Sigma \wedge (\langle e\rangle^\frown s, X, Y) \in \mathcal{B}(P_2)\end{array}\right\}\end{array} \tag{16}$$

Equation (16) simply says that the process always behaves like P_1 unless P_1 refuses in which case it behaves like P_2. It allows P_1 to perform any event, terminate or livelock if it is capable of so doing. When P_1 is active in any sense, it is let loose:

$$\langle\rangle : X \rightsquigarrow U \quad \text{if} \quad P_1 :: \langle\rangle : X \rightsquigarrow U \quad \text{apart from } U = \emptyset$$

P_2 is only allowed to be active when P_1 is not. If P_1 shows any sign of life, even pathological life, it executes. Even if P_2 can perform ✓ or signal ✗, it is ignored until all events from P_1 are positively blocked.

Example 9 $(a \to \mathit{Stop}) \overleftarrow{\Box} (a \to b \to \mathit{Stop}) = a \to \mathit{Stop}$

In general, $(a \to P_1) \overleftarrow{\Box} (a \to P_2) = a \to P_1$. ■[9]

Example 10 *SKIP guards.*

Skip $\overleftarrow{\Box}$ $(a \to \mathit{Stop})$ *only has acceptances of the sort:*

$$\langle\rangle : X \rightsquigarrow \{\checkmark\}$$

because Skip never refuses, so $\mathit{Skip} \overleftarrow{\Box} (a \to \mathit{Stop}) = \mathit{Skip}$. *And*

```
PRI ALT
  B & SKIP
    P
  a?
    Q
```

is $(Skip \fatsemi P \blacktriangleleft B \blacktriangleright Stop) \overleftarrow{\Box} (a \to Q) = P \blacktriangleleft B \blacktriangleright (a \to Q)$ *as we would wish. This gives a denotational semantics for, and so defines, SKIP guards. However,*

```
PRI ALT
  a?
    P
  SKIP
    Q
```

is not *normally implemented with the semantics of* $\overleftarrow{\Box}$*. The reason is that when a*? *is checked for availability, the process which outputs a is not necessarily involved. This is quite different from the semantics of equation* (16) *which requires* P_1 *to positively refuse before* P_2 *is allowed to execute.*

Example 11 *Skip and Spin are left multiplicative zeroes of* $\overleftarrow{\Box}$ *while Stop is a unit:*

$$Skip \overleftarrow{\Box} P = Skip \qquad Spin \overleftarrow{\Box} P = Spin \qquad Stop \overleftarrow{\Box} P = P \qquad P \overleftarrow{\Box} Stop = P$$

Like its compliant counterpart, $\overleftarrow{\Box}$ is associative:

$$P_1 \overleftarrow{\Box} \left(P_2 \overleftarrow{\Box} P_3\right) = \left(P_1 \overleftarrow{\Box} P_2\right) \overleftarrow{\Box} P_3$$

Notice however that

$$P_1 \sqcap \left(P_2 \overleftarrow{\Box} P_3\right) \sqsupseteq (P_1 \sqcap P_2) \overleftarrow{\Box} (P_1 \sqcap P_3) \tag{17}$$

because $P_1 \overleftarrow{\Box} P_3$ and $P_2 \overleftarrow{\Box} P_1$ are refinements of the right hand side, but not of the left.

Lemma 6 *Let* $\{P_i : \Sigma \to \mathcal{CSPP}\}$ *be an indexed nonempty set of processes. Then* $Q \overleftarrow{\Box} \bigsqcap\{P_i \mid i \in I\} = \bigsqcap\{Q \overleftarrow{\Box} P_i \mid i \in I\}$ *and* $\bigsqcap\{P_i \mid i \in I\} \overleftarrow{\Box} Q = \bigsqcap\{P_i \overleftarrow{\Box} Q \mid i \in I\}$.

8.9 Parallels

8.9.1 Interleaving

When $P_1 \underset{E}{\|} P_2$ executes and performs a trace s, then the parallel processes have performed matching traces (s_1, s_2). If $E = \Sigma$, then all events are synchronised and performed jointly by P_1 and P_2. In that case $s_1 = s_2 = s$.

The other extreme is when $E = \emptyset$, so no events are synchronised. That is just standard interleaving: $P_1 \underset{\emptyset}{\|} P_2 = P_1 \,|||\, P_2$. True concurrency is not part of the present theory.

In general a trace s can arise from more than one pair of traces (s_1, s_2). We call this way of picking events from the s_1 and s_2 to form s generalised interleaving and write $s \in s_1 \underset{E}{\|} s_2$. Clearly $\langle\rangle \underset{E}{\|} \langle\rangle = \{\langle\rangle\}$.

The following abbreviation captures this:

Definition 11 *Let $s \in \Sigma^*$. Then*

$$\mathsf{Interleaves}(P_1, P_2, s, E) \equiv$$
$$\left(\left\{ f : \downarrow s \to \Sigma^{*2} \;\middle|\; \begin{array}{c} f(\langle\rangle) = (\langle\rangle, \langle\rangle) \\ \wedge \\ \forall\, t^\frown\langle x\rangle \in \downarrow s \bullet \exists\, t_1 \in traces(P_1) \bullet \exists\, t_2 \in traces(P_2) \bullet f(t) = (t_1, t_2) \\ \wedge \\ \left(\begin{array}{l} x \in E \Rightarrow \\ \left(\begin{array}{c} \exists\, X, Y_1, Y_2 \bullet \\ (t_1, X, Y_1) \in \mathcal{B}(P_1) \wedge (t_2, X, Y_2) \in \mathcal{B}(P_2) \wedge x \in Y_1 \cap Y_2 \\ \wedge \\ f\,(t^\frown\langle x\rangle) = (t_1{}^\frown\langle x\rangle, t_2{}^\frown\langle x\rangle) \end{array} \right) \\ \wedge\ \ x \notin E \Rightarrow \\ \left(\begin{array}{c} f\,(t^\frown\langle x\rangle) = (t_1{}^\frown\langle x\rangle, t_2) \wedge t_1{}^\frown\langle x\rangle \in traces(P_1) \\ \vee \\ f\,(t^\frown\langle x\rangle) = (t_1, t_2{}^\frown\langle x\rangle) \wedge t_2{}^\frown\langle x\rangle \in traces(P_2) \end{array} \right) \end{array} \right) \end{array} \right\} \right)$$

The projections of $f \in \mathsf{Interleaves}(P_1, P_2, s, E)$ are written as f_1 and f_2 so $f(t) = (f_1(t), f_2(t))$.

Definition 11 has to include a number of details to ensure that irregular processes are handled properly.

8.9.2 General parallel

$$\mathcal{B}(P_1 \underset{E}{\|} P_2) = \left\{ (s, X, Y) \;\middle|\; \begin{array}{c} \exists f \in \mathsf{Interleaves}(P_1, P_2, s, E) \bullet \exists\, Y_1, Y_2 \in \overset{\surd\times}{\mathbb{P}}\, X^{\surd\times} \bullet \\ (f_1(s), X, Y_1) \in \mathcal{B}(P_1) \wedge (f_2(s), X, Y_2) \in \mathcal{B}(P_2) \\ \wedge \\ Y \in \overset{\surd\times}{\mathbb{M}}\Big((Y_1 \cap Y_2 \cap E^\surd) \ \cup\ ((Y_1 \cup Y_2) - E^\surd) \Big) \end{array} \right\} \tag{18}$$

$\overset{\surd\times}{\mathbb{M}}(X)$ was defined on page 165: see definition 1. In equation (18), Y can be any available nonempty subset or available singleton token.

The difficulty with hard priority here is just the same as that which arises with $\Box$: there can be 'unexpected' deadlocks. So $(P_1 \,|||\, P_2) \,|||\, (P_1 \,|||\, P_2)$ can behave like $(P_1 \overleftarrow{|||} P_2) \overleftrightarrow{|||} (P_1 \overrightarrow{|||} P_2)$ at any point and create a deadlock which would not arise in standard CSP. As a consequence, $P_1 \underset{E}{\|} P_2$ should not be identified with the corresponding operator of standard CSP. In most circumstances, $P_1 \overleftrightarrow{\underset{E}{\|}} P_2$ will be the right identification, although just as for $\Box$

$$P_1 \overset{CSP}{\underset{E}{\|}} P_2 = P_1 \overleftrightarrow{\underset{E}{\|}} P_2 \ \sqcap\ P_1 \overleftarrow{\underset{E}{\|}} P_2$$

and

$$P_1 \overset{CSP}{\underset{E}{\|}} P_2 = P_1 \overleftrightarrow{\underset{E}{\|}} P_2 \ \sqcap\ P_1 \overrightarrow{\underset{E}{\|}} P_2$$

are more abstract possibilities as well as variants that are not consistent at each step.

Lemma 7 *Equation* (18) *defines a process.*

Proof. Suppose $(f_1(s), X, Y_1) \in \mathcal{B}(P_1) \wedge (f_2(s), X, Y_2) \in \mathcal{B}(P_2)$. Then acceptances of the form $(f_i(s), X', Y_2')$ exist for all other $X' \subseteq \Sigma$, so all (s, X', Y') triples are present.

The set of acceptances must also satisfy **H3**. Take $s^\frown\langle x\rangle \in traces\left(P_1 \underset{E}{\|} P_2\right)$. That means that $\exists (s^\frown\langle x\rangle, X, Y) \in \mathcal{B}\left(P_1 \underset{E}{\|} P_2\right)$. Suppose $x \in E$. Then there is an $f \in \mathsf{Interleaves}(P_1, P_2, s^\frown\langle x\rangle, E)$ with $(f_1(s^\frown\langle x\rangle), X, Y_1) \in \mathcal{B}(P_1)$ and $(f_2(s^\frown\langle x\rangle), X, Y_2) \in \mathcal{B}(P_2)$. ¿From definition 11, there are some sets X', Y_1' and Y_2' for which $(f_1(s), X', Y_1') \in \mathcal{B}(P_1)$ and $(f_2(s), X', Y_2') \in \mathcal{B}(P_2)$ with $x \in Y_1' \cap Y_2'$. This shows that there is a Y with $(s, X', Y) \in \mathcal{B}\left(P_1 \underset{E}{\|} P_2\right)$ and $x \in Y$ so the forward implication of **H3** is satisfied when $x \in E$.

When $x \notin E$ the result is even easier to establish because only one parallel partner is involved.

The converse implication of **H3** is obviously satisfied. ⊣

Example 12 *Take P_1, P_2 and Q to be processes with acceptances which include*

$Q :: \langle\rangle :$	$\{a,b\} \rightsquigarrow \{a\}$	$P_1 :: \langle\rangle :$	$\{a,b\} \rightsquigarrow \{a\}$	$P_2 :: \langle\rangle :$	$\{a,b\} \rightsquigarrow \{b\}$
	$\{a,c\} \rightsquigarrow \{c\}$		$\{a,c\} \rightsquigarrow \{a\}$		$\{a,c\} \rightsquigarrow \{a\}$
	$\{b,c\} \rightsquigarrow \{c\}$		$\{b,c\} \rightsquigarrow \emptyset$		$\{b,c\} \rightsquigarrow \{b\}$
	$\{a\} \rightsquigarrow \emptyset$		$\{a\} \rightsquigarrow \{a\}$		$\{a\} \rightsquigarrow \{a\}$
	$\{b\} \rightsquigarrow \{b\}$		$\{b\} \rightsquigarrow \emptyset$		$\{b\} \rightsquigarrow \{b\}$
	$\{c\} \rightsquigarrow \{c\}$		$\{c\} \rightsquigarrow \emptyset$		$\{c\} \rightsquigarrow \emptyset$
$Q :: \langle a\rangle :$	$X \rightsquigarrow X \cap \{a,b,c\}$	$P_1 :: \langle a\rangle :$	$X \rightsquigarrow \emptyset$	$P_2 :: \langle a\rangle :$	$X \rightsquigarrow X \cap \{b\}$
$Q :: \langle b\rangle :$	$X \rightsquigarrow X \cap \{a,b,c\}$			$P_2 :: \langle b\rangle :$	$X \rightsquigarrow \emptyset$
$Q :: \langle c\rangle :$	$X \rightsquigarrow X \cap \{a,b,c\}$				

Then $Q \parallel P_1 :: \langle\rangle : \{a,b\} \rightsquigarrow \{a\}$, so it has a trace $\langle a\rangle$ after which it stops. $Q \parallel P_2 :: \langle\rangle : \{b\} \rightsquigarrow \{b\}$ provides the only nonempty trace, $\langle b\rangle$, after which it stops. In particular, this process cannot perform the trace $\langle ab\rangle$. However $Q \parallel (P_1 \sqcap P_2)$ can *perform the trace $\langle ab\rangle$. This shows that*

$$Q \parallel (P_1 \sqcap P_2) \neq Q \parallel P_1 \;\sqcap\; Q \parallel P_2 \,.$$

The various responses in equation (18) allow, for example, a scheduler to make arbitrary decisions about which events to select in preference to others, perhaps for reasons of efficiency. $P_1 \underset{E}{\overset{\leftrightarrow}{\|}} P_2$ does not permit that freedom:

$$\mathcal{B}(P_1 \underset{E}{\overset{\leftrightarrow}{\|}} P_2) = \left\{ (s,X,Y) \;\middle|\; \begin{array}{c} \exists f \in \mathsf{Interleaves}(P_1,P_2,s,E) \bullet \exists\, Y_1, Y_2 \in \overset{\surd\!\times}{\mathbb{P}}\, X^{\surd\!\times} \bullet \\ (f_1(s),X,Y_1) \in \mathcal{B}(P_1) \wedge (f_2(s),X,Y_2) \in \mathcal{B}(P_2) \\ \wedge \\ Y \in \overset{\surd\!\times}{\mathbb{S}}\left((Y_1 \cap Y_2 \cap E^\surd) \;\cup\; ((Y_1 \cup Y_2) - E^\surd)\right) \end{array} \right\} \tag{19}$$

$\overset{\surd\!\times}{\mathbb{S}}(X)$ was defined on page 165.

8.9.3 Derived Parallels

As usual there are derived versions.

$$P_X \|_Y Q \;=\; P' \underset{X\cap Y}{\|} Q'$$

where $P' = P \underset{\Sigma}{\|} \mathrm{Run}_X$, $Q' = Q \underset{\Sigma}{\|} \mathrm{Run}_Y$ and $\mathrm{Run}_E = e : E \to \mathrm{Run}_E$. And of course interleaving: $P \,|||\, Q \;=\; P \underset{\emptyset}{\|} Q$ and fully synchronised parallel: $\| = \underset{\Sigma}{\|}$.

8.10 Prioritised parallel

For the form of parallel composition that always favours one partner, we have to refine the notion of interleaving to match because triples like (s, X, Y) only record a very limited history in the trace component s.

Definition 12 *Let $s \in \Sigma^*$. Then*

$$\overleftarrow{\mathsf{Interleaves}}(P_1, P_2, s, E) \equiv$$

$$\left\{ f : \downarrow s \to \Sigma^{*2} \;\middle|\; \begin{array}{c} f(\langle\rangle) = (\langle\rangle, \langle\rangle) \\ \wedge \\ \forall\, t^\frown\langle x\rangle \in \downarrow s \bullet \exists\, t_1 \in traces(P_1) \bullet \exists\, t_2 \in traces(P_2) \bullet f(t) = (t_1, t_2) \\ \wedge \\ \left(\begin{array}{l} x \in E \Rightarrow \\ \left(\begin{array}{c} \exists\, X, Y_1, Y_2 \bullet \\ (t_1, X, Y_1) \in \mathcal{B}(P_1) \wedge (t_2, X, Y_2) \in \mathcal{B}(P_2) \wedge x \in Y_1 \cap Y_2 \\ \wedge \\ f(t^\frown\langle x\rangle) = (t_1{}^\frown\langle x\rangle, t_2{}^\frown\langle x\rangle) \end{array} \right) \\ \wedge \\ x \notin E \Rightarrow \\ \left(\begin{array}{c} f(t^\frown\langle x\rangle) = (t_1{}^\frown\langle x\rangle, t_2) \wedge t_1{}^\frown\langle x\rangle \in traces(P_1) \\ \vee \\ \left(\begin{array}{c} f(t^\frown\langle x\rangle) = (t_1, t_2{}^\frown\langle x\rangle) \\ \wedge \\ \exists\, X, Y_2 \bullet \\ (t_1, X, \emptyset) \in \mathcal{B}(P_1) \wedge (t_2, X, Y_2) \in \mathcal{B}(P_2) \wedge x \in Y_2 \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right\}$$

The projections of $f \in \overleftarrow{\mathsf{Interleaves}}(P_1, P_2, s, E)$ are again written as f_1 and f_2 so $f(t) = (f_1(t), f_2(t))$.

$$\mathcal{B}(P_1 \underset{E}{\overleftarrow{\parallel}} P_2) = \left\{ (s, X, Y) \;\middle|\; \begin{array}{c} \exists f \in \overleftarrow{\mathsf{Interleaves}}(s, E) \bullet \exists\, Y_1, Y_2 \in \overset{\surd\!\!\!\!\times}{\mathbb{P}}\, X^{\surd\!\!\!\!\times} \bullet \\ (f_1(s), X, Y_1) \in \mathcal{B}(P_1) \wedge (f_2(s), X, Y_2) \in \mathcal{B}(P_2) \\ \wedge \\ Y \in \overset{\surd\!\!\!\!\times}{\mathbb{S}} \left((Y_1 \cap Y_2 \cap E^\surd) \;\cup\; \left(\left(Y_2 \blacktriangleleft (Y_1 - E^\surd) = \emptyset \blacktriangleright Y_1 \right) - E^\surd \right) \right) \end{array} \right\} \tag{20}$$

A less explicit way to define $P_1 \underset{E}{\overleftarrow{\parallel}} P_2$ would just use the ordinary Interleaves and the $\lfloor B \rfloor$ notation of section 5.1 on page 166 to eliminate the excess traces.

8.11 Sequential Composition

$$\mathcal{B}(P_1 \fatsemi P_2) =$$

$$\begin{array}{c} \{(s, X, Y) \in \mathcal{B}(P_1) \,|\, Y \neq \{\surd\}\} \\ \cup \\ \{(s, X, Y) \,|\, (s, X, \{\surd\}) \in \mathcal{B}(P_1) \wedge (\langle\rangle, X, Y) \in \mathcal{B}(P_2)\} \\ \cup \\ \left\{ (t_1{}^\frown\langle x\rangle{}^\frown t_2, X, Y) \;\middle|\; \begin{array}{c} \exists\, X', Y' \bullet (t_1, X', \{\surd\}) \in \mathcal{B}(P_1) \wedge (\langle\rangle, X', Y') \in \mathcal{B}(P_2) \wedge x \in Y' \\ \wedge \\ (\langle x\rangle{}^\frown t_2, X, Y) \in \mathcal{B}(P_2) \end{array} \right\} \end{array} \tag{21}$$

The reference to two triples from $\mathcal{B}(P_2)$ in the last set in equation (21) means that distribution over $\sqcap$ is unlikely to hold in all cases, and this proves to be the case.

Example 13 *Take $\Sigma = \{a, b\}$ and let Q be the rather artificial process that terminates on the offer $\{a, b\}$ and refuses all other offers. That is $Q :: \langle\rangle : \{a, b\} \rightsquigarrow \{\checkmark\}$ and $Q :: \langle\rangle : X \rightsquigarrow \emptyset$ for all other X. Write $P_1 = (a \to a \to Stop) \overleftarrow{\Box} (b \to a \to Stop)$ and $P_2 = (b \to b \to Stop) \overleftarrow{\Box} (a \to b \to Stop)$. Notice that $Q \fatsemi P_1$ has traces $\{\langle\rangle, \langle a\rangle, \langle aa\rangle\}$ and $Q \fatsemi P_2$ has traces $\{\langle\rangle, \langle b\rangle, \langle bb\rangle\}$. However $Q \fatsemi (P_1 \sqcap P_2)$ has a trace $\langle ab\rangle$ so this shows that $Q \fatsemi P_1 \ \sqcap \ Q \fatsemi P_2 \neq Q \fatsemi (P_1 \sqcap P_2)$.*

Distribution across $\bigsqcap$ does work in the other direction:

Lemma 8 *Let $\{P_i : \Sigma \to \mathcal{CSPP}\}$ be an indexed nonempty set of processes. Then $\bigsqcap\{P_i \mid i \in I\} \fatsemi Q = \bigsqcap\{P_i \fatsemi Q \mid i \in I\}$.*

The failure to distribute over $\sqcap$ in every case arises from the multiple references to triples from $\mathcal{B}(P_2)$ in equation (21). In effect, the present semantics is too fine grained to guarantee the distribution. The failure is repaired in a more coarse grained semantics in which behaviours representing the histories of experiments are recorded. This yields a slightly less abstract version of $\mathcal{CSPP}$, but is not examined further here. The same phenomenon occurs in hiding below.

Notice that we cover processes like $\mu p \bullet (Skip \sqcap a \to p) \fatsemi Q$. And also that it is trivial to check that $Skip \fatsemi P = P \fatsemi Skip = P$.

8.12 Hiding

Conceptually, $P \setminus H$ is a process with the internal behaviour of P, yet with external 'visible' behaviour which excludes any 'internal' events from the set H. The internal events of H are no longer subject to direct environmental control. Often there are several possibilities for the internal dynamics: the process $P \setminus H$ models them by nondeterminism.

The most natural form of hiding offers the internal hidden events in a compliant way. The version below also allows external hesitant offers.

$$
\begin{aligned}
&\mathcal{B}(P \setminus H) = \\
&\qquad \left\{ (s \setminus H, X, \{\checkmark\}) \;\middle|\; \begin{array}{c} \mathsf{Accessible}(P, s, H) \\ \wedge \\ \exists\, w \in H^{\omega} \bullet \forall\, n \in \mathbb{N} \bullet \\ \exists\, Y' \bullet w(n+1) \in Y' \wedge (s^{\frown}(w \upharpoonright n), X \cup H, Y') \in \mathcal{B}(P) \end{array} \right\} \\
&\qquad\qquad \cup \\
&\left\{ (s \setminus H, X, Y \setminus H) \;\middle|\; Y \in \overset{\checkmark}{\mathbb{S}} \left(\bigcup \left\{ Y' \;\middle|\; \begin{array}{c} \mathsf{Accessible}(P, s, H) \\ \wedge \\ \exists\, u \bullet s \leqslant u \wedge s \setminus H = u \setminus H \\ \wedge \\ (u, X \cup H, Y') \in \mathcal{B}(P) \\ \wedge \\ \forall\, n \in [\#s, \#u) \bullet \exists\, Y'' \bullet \\ (u \upharpoonright n, X \cup H, Y'') \in \mathcal{B}(P) \wedge u(n+1) \in Y'' \end{array} \right\} \right) \right. \\
&\qquad\qquad \left. \begin{array}{c} \wedge \\ (Y \setminus H = \emptyset) \Rightarrow \mathsf{Stable}(P, s, X, H) \end{array} \right\}
\end{aligned}
\tag{22}
$$

where

$$\mathsf{Accessible}(P,s,H) \equiv \forall\, n \in [0,\#s) \bullet \exists\, X', Y' \bullet (s \restriction n, X' \cup H, Y') \in \mathcal{B}(P) \wedge s(n+1) \in Y'$$

means that there is some way that P can perform the trace s when the events of H are always included compliantly in any offer.

$$\begin{aligned}\mathsf{Stable}(P,s,X,H) \equiv \exists\, u_0 \bullet s \leqslant u_0 \wedge s \setminus H = u_0 \setminus H \wedge (u_0, X \cup H, \emptyset) \in \mathcal{B}(P) \wedge \\ \forall\, n \in [\#s, \#u_0) \bullet \exists\, Y \bullet (u_0 \restriction n, X \cup H, Y) \in \mathcal{B}(P) \wedge u_0(n+1) \in Y\end{aligned}$$

is used to ensure that external refusals are genuine rather than as a side effect of hiding internal events.

The first set in equation (22) captures livelock. The second set references more than one triple in $\mathcal{B}(P)$ and this is why it does not distribute over $\sqcap$. It is however manifestly monotone.

Example 14 *Let*

$$\begin{aligned}P_1 &= \overleftrightarrow{\{a,h\}} \rightarrow \overleftrightarrow{\{b,h\}} \rightarrow \overleftrightarrow{\{c,h\}} \rightarrow \mathit{Stop}\\ P_2 &= \overleftrightarrow{\{e,h\}} \rightarrow \overleftrightarrow{\{d,h\}} \rightarrow \overleftrightarrow{\{f,h\}} \rightarrow \mathit{Stop}\\ &\textit{and}\\ P_3 &= \overleftrightarrow{\{a,h\}} \rightarrow \overleftrightarrow{\{d,h\}} \rightarrow \overleftrightarrow{\{c,h\}} \rightarrow \mathit{Stop}\end{aligned}$$

Then

$$\begin{aligned}&P_1 \setminus \{h\} : \{a,b,c,d,e,f\} \rightsquigarrow \{a,b,c\}\\ &P_2 \setminus \{h\} : \{a,b,c,d,e,f\} \rightsquigarrow \{d,e,f\}\\ &\textit{and}\\ &P_3 \setminus \{h\} : \{a,b,c,d,e,f\} \rightsquigarrow \{a,c,d\}\end{aligned}$$

Since $P_1 \sqcap P_2$ can behave like P_3, it follows that $(P_1 \sqcap P_2) \setminus \{h\} \neq P_1 \setminus \{h\} \ \sqcap \ P_2 \setminus \{h\}$.

Lemma 9 *Hiding is monotone: $P \sqsubseteq Q \Rightarrow P \setminus H \sqsubseteq Q \setminus H$.*

Proof. Obvious by inspection of equations (12) and (22). ⊣

It does not appear that there can be any sensible definition of hiding using the present fine grained semantics which can distribute over $\sqcap$. Sliding requires that we access more than one unit of information to determine responses, and this is incompatible with the distribution. It is interesting to notice how the problem is neatly side-stepped in standard CSP semantics based on Failures by employing 'inverted logic'.

8.13 Renaming

Renaming needs careful definition because one-many renaming introduces additional non determinism. If R is the renaming relation we write $s\ \mathsf{R}\ s'$ to mean that the trace s' is a pointwise renamed version of s. And it is useful to extend it to $\Sigma^{\checkmark\!\!\times}$:

Definition 13 $\mathsf{R}^{\checkmark\!\!\times} = \mathsf{R} \cup \{\checkmark \mapsto \checkmark, \times \mapsto \times\}$.

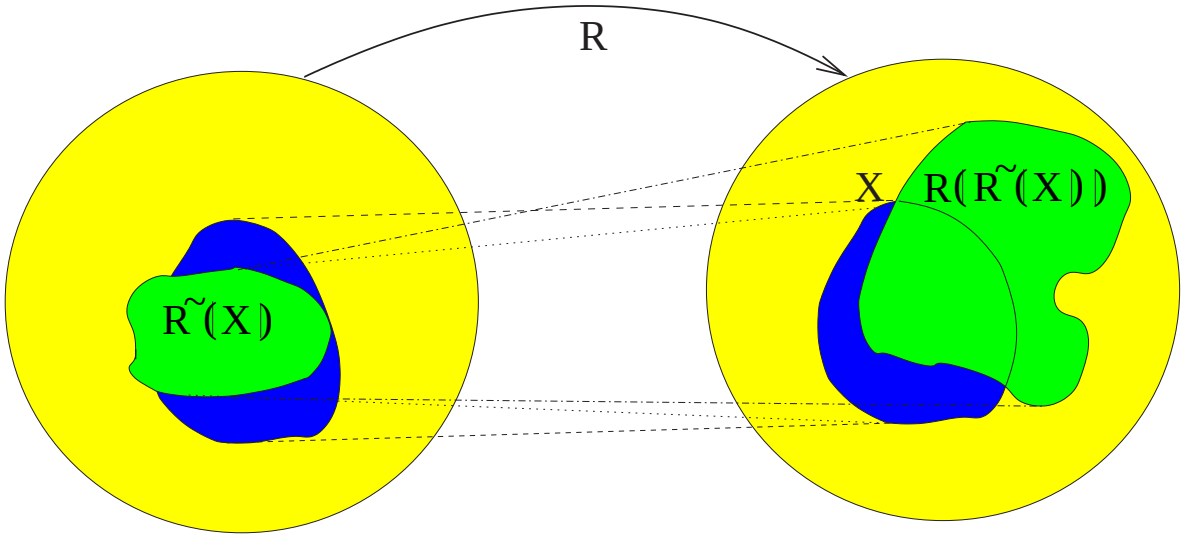

Figure 1: Renaming with R

If P is a process, $P[\![\mathsf{R}]\!]$ is the renamed process.

$$\mathcal{B}\,P[\![\mathsf{R}]\!] = \left\{ (s,X,Y) \;\middle|\; \begin{array}{c} \exists\, t \bullet t\ \mathsf{R}\ s \wedge \exists\, Y' \bullet (t, \mathsf{R}^{\sim}\ (\!|X|\!), Y') \in \mathcal{B}\,(P) \wedge Y = \mathsf{R}^{\checkmark\times}\ (\!|Y'|\!) \cap X^{\checkmark\times} \\ \wedge \\ \forall\, n \in [0, \#s) \bullet \exists\, t \bullet \exists\, X', Y' \bullet \\ t\ \mathsf{R}\ (s \upharpoonright n) \wedge (t, \mathsf{R}^{\sim}\ (\!|X'|\!), Y') \in \mathcal{B}\,(P) \wedge s(n+1) \in \mathsf{R}^{\checkmark\times}\ (\!|Y'|\!) \cap X'^{\checkmark\times} \end{array} \right\} \quad (23)$$

$\mathsf{R}^{\sim}$ is the reverse relation: the notation is similar to that used in Z. And $\mathsf{R}\ (\!|X|\!) = \{y \mid x\ \mathsf{R}\ y\}$ is the relational image of the set X. The second predicate in equation (23) ensures that **H3** is obeyed: it is needed because the axioms admit some very irregular processes.

Renaming is clearly monotone.

Lemma 10 *Renaming is monotone:* $P \sqsubseteq Q \Rightarrow P[\![\mathsf{R}]\!] \sqsubseteq Q[\![\mathsf{R}]\!]$.

8.14 Interruption: $P_1 \underset{(i,r)}{\triangle} P_2$

A variant of the usual sort of interrupt operator is defined here with one eye on the evolution of the Honeysuckle language: see [17], [18] and [19].

Let P_1 and P_2 be two processes and i and r two distinguished events. i will represent an interruption, and r can be regarded as a 'return from interrupt'. The intention here is that P_2 is something like a standard interrupt service routine. Denote the alphabets of P_1 and P_2 by αP_1 and αP_2 in the sense that these are the sets of events that the processes can perform. We require that these alphabets be disjoint, and that i is in neither of them. However $r \in \alpha P_2$ since we want P_2 to be able to complete its processing by executing r.

Two standard notations that have not been needed above are $s \downarrow a$ which is the number of occurrences of the event a in the trace s; and $s \upharpoonright E$ which is the subsequence of s consisting of members of the set E. So $s \downarrow i = s \downarrow r$ will determine whether the trace s contains matching pairs of i and r, and so whether we are 'background processing' in P_1 or in the interrupt service routine P_2. And $s \upharpoonright (\alpha P_1)$ will be the portion of s executed by P_1 and similarly for P_2.

Then

$$\mathcal{B}\left(P_1 \underset{(i,r)}{\triangle} P_2\right) = \left\lfloor \left\{ (s,X,Y) \;\middle|\; \begin{array}{c} s \in (\alpha P_1 \cup \alpha P_2 \cup \{i\})^* \\ \wedge \\ (Y = \{i\} \blacktriangleleft i \in X \blacktriangleright (s \restriction (\alpha P_1), X, Y) \in \mathcal{B}(P_1)) \\ \blacktriangleleft s \downarrow i = s \downarrow r \blacktriangleright \\ (s \restriction (\alpha P_2), X, Y) \in \mathcal{B}(P_2) \end{array} \right\} \right\rfloor \tag{24}$$

Recall that $\lfloor B \rfloor$ was defined in section 5.1 on page 166. It avoids the need to be explicit about the set of traces: only the traces that can be 'accepted', starting from $\langle\rangle$, are included.

8.15 Refinement

Refinement is as usual

$$P_1 \sqsupseteq P_2 \;\Leftrightarrow\; P_2 = P_2 \sqcap P_1 \tag{25}$$

which simply maps onto set inclusion on the acceptances:

$$P_1 \sqsupseteq P_2 \;\Leftrightarrow\; \mathcal{B}P_1 \subseteq \mathcal{B}P_2 \tag{26}$$

Proof. Assume $P_1 \sqsupseteq P_2$ or $P_2 = P_2 \sqcap P_1$ from (25) above. Then $\mathcal{B}P_2 = \mathcal{B}P_2 \cup \mathcal{B}P_1$ which gives $\mathcal{B}(P_1) \subseteq \mathcal{B}(P_2)$ immediately. The converse is equally obvious. ⊣

8.15.1 ⊥

The most nondeterministic process which has all possible behaviours is evidently below any other process in this order: it is the least element $\bot$ of $\sqsupseteq$.

There is no top process in this version based on triples, although it appears in versions based on behaviours.

8.15.2 Meets and joins

Let S be a nonempty set of processes. $\bigsqcap S$ is the obvious candidate to be the meet. *Proof.* First, $\bigsqcap S$ is a lower bound of S. If $P \in S$, then $\mathcal{B}P \subseteq \mathcal{B}(\sqcap S)$ is an immediate consequence of the definition. But this is $P \sqsupseteq \sqcap S$ so $\sqcap S$ is a lower bound of S.

Second, it is clear that $\sqcap S$ is the supremum of all lower bounds, for any lower bound must contain $\cup\{\mathcal{B}P \mid P \in S\}$. ⊣

If S has a join, then it must match the intersection of the behaviours: $\bigcap\{\mathcal{B}P \mid P \in S\}$. Unfortunately, this can fail to define a process, as in the pair $(a \rightarrow \mathit{Stop})$ and $(b \rightarrow \mathit{Stop})$. With $\Sigma = \{a,b,c\}$, $\mathcal{B}(a \rightarrow \mathit{Stop}) \cap \mathcal{B}(b \rightarrow \mathit{Stop}) = \{\langle\rangle : \{c\} \rightsquigarrow \emptyset, \langle\rangle : \emptyset \rightsquigarrow \emptyset\}$ which is not a process: **H2** is violated.

Theorem 1 $(\mathcal{CSPP}, \sqsubseteq)$ *is a Complete Partial Order when* Σ *is finite.*

Proof. Let $\mathcal{D}$ be a directed set of processes. It is necessary to show that

$$\mathcal{B}(U) = \bigcap\{\mathcal{B}(D) \mid D \in \mathcal{D}\}$$

represents a process. When Σ is finite, so also are $\mathbb{P}\Sigma$ and $\overset{\checkmark\times}{\mathbb{P}}\Sigma^{\checkmark\times}$. So the number of choices of (s,X,Y) for a fixed s is finite. This observation is used to establish that U satisfies the axioms **H1**, **H2** and **H3** of section 7.

For **H1** suppose that there is no instance of $(\langle\rangle, X, Y) \in \mathcal{B}(U)$. There is only a finite number of such possibilities, so it is possible to construct a finite subset of $\mathcal{D}$ with an upper

bound which has no instance of $(\langle\rangle, X, Y)$ for some X and Y. This is a contradiction in that the bound is itself a process satisfying **H1**. Thus U satisfies **H1**.

U must satisfy **H2** by a very similar argument noting that there are only a finite number of choices for Y in (s, X, Y) when s and X are fixed.

Suppose $s^\frown\langle x\rangle \in traces(U)$: we need to show that there is some common $(s, X, Y) \in \mathcal{B}(U)$ with $x \in Y \cap \Sigma$. If we assume the contrary and note the finite number of choices for (X, Y), once again we can construct a finite subset of $\mathcal{D}$ with an upper bound which will fail **H3**. Thus

$$s^\frown\langle x\rangle \in traces(U) \;\Rightarrow\; \exists (s, X, Y) \in \mathcal{B}(U) \bullet x \in Y \cap \Sigma \; .$$

The converse is obviously satisfied, so U satisfies **H3**.

⊣

8.16 Recursion

8.16.1 Fixed points from the refinement partial order.

$\mu p \bullet f(p)$ denotes a fixed point of the function f. This is often the least fixed point with respect to the refinement order in standard untimed CSP. Theorem 1 establishes $\mathcal{CSPP}$ as a CPO. Standard theorems now ensure that every monotone function f has a least fixed point, and so $\mu p \bullet f(p)$ is well defined. All the ordinary operators of $\mathcal{CSPP}$ are monotone with respect to the refinement order $\sqsupseteq$: in most cases this follows from the fact that they distribute over $\sqcap$.

If f is *continuous*, that is for every directed set $\mathcal{D}$ of processes, $\bigsqcup f(\!|\mathcal{D}|\!)$ exists and is the same as the image of the join of $\mathcal{D}$: $\bigsqcup f(\!|\mathcal{D}|\!) = f(\sqcup\mathcal{D})$, then another theorem gives a more useful formula:

$$\mu p \bullet f(p) = \bigsqcup \{f^n(\bot) \mid n \in \mathbb{N}_0\} \; . \tag{27}$$

Often there is a unique fixed point, but the results above do not help directly in identifying such cases. A metric exists which is useful in such cases.

8.16.2 Monotone properties

All of the standard operators are monotone in $\sqsubseteq$: this was shown as each was introduced above. In most cases, the result follows from the the fact that the operators distribute over $\sqcap$, but that is a stronger property. Hiding in particular does not so distribute, but is nevertheless monotone.

Lemma A.1.8 on page 484 of [2] shows that $\mu : (\mathcal{CSPP} \xrightarrow{m} \mathcal{CSPP}) \to \mathcal{CSPP}$ is monotone. $\mathcal{CSPP} \xrightarrow{m} \mathcal{CSPP}$ denotes the space of monotone functions on $\mathcal{CSPP}$ processes. The monotone relation on $\mathcal{CSPP} \xrightarrow{m} CSPP$ is defined pointwise:

$$f \sqsupseteq f' \Leftrightarrow \forall P \in \mathcal{CSPP} \bullet f(P) \sqsupseteq f'(P) \; .$$

Hence if $h(Q, P)$ is monotone in both arguments, then $\lambda Q \bullet h(P, Q) \in (\mathcal{CSPP} \xrightarrow{m} CSPP)$ so $\lambda P \bullet \mu Q \bullet h(Q, P)$ is also monotone. And similar arguments apply for other forms of which $\lambda P \bullet \mu Q \bullet f_P(Q)$ is perhaps the most general.

In the case of mutual recursion, that is a recursion involving a sequence of functions $\langle f_i \mid i \in I\rangle$ where I is an indexing set, then the standard treatment in [2] applies. So mutual recursion is also monotone and the fixed points are well defined. In applications, I is nearly always a subset of $\mathbb{N}$.

All the elementary functions of $\mathcal{CSPP}$ have now been shown to be monotone. Since the composition of monotone functions is itself monotone, this means that all the compound functions in $\mathcal{CSPP}$ have least fixed points.

Example 15 *$\mu P \bullet Skip \fatsemi P = \bot$ because $P = Skip \fatsemi P$ for every process P.*

8.16.3 Continuity

Once a function is known to be monotone, a least fixed point is guaranteed. If in addition, it is continuous in the partial order, then the constructive equation (27) on page 180 can be used.

When, as in $\mathcal{CSPP}$, we have a metric and the function is contracting, then we have the stronger result that iterating from *any* starting point, not just from $\bot$, will converge into a unique fixed point. Given the power of the metric approach, it is not often that *continuity* with respect to refinement is particularly useful.

It is easy to check that $f(\!|\mathcal{D}|\!)$ is directed when f is monotone and $\mathcal{D}$ is itself directed. So $\bigsqcup f(\!|\mathcal{D}|\!)$ always exists for any $f : \mathcal{CSPP} \to \mathcal{CSPP}$ and directed set $\mathcal{D}$. And by definition, $f(\bigsqcup \mathcal{D}) = \bigsqcup f(\!|\mathcal{D}|\!)$ for f continuous. If f is monotone, $f(\bigsqcup \mathcal{D}) \sqsupseteq \bigsqcup f(\!|\mathcal{D}|\!)$, because otherwise $f(\bigsqcup \mathcal{D}) \sqsubset \bigsqcup f(\!|\mathcal{D}|\!)$, which would entail some $D \in \mathcal{D}$ for which $f(\bigsqcup \mathcal{D}) \sqsubset f(D)$ which is a contradiction for monotone f.

So it is only necessary to prove the reverse containment $\mathcal{B} f(\bigsqcup \mathcal{D}) \supseteq \mathcal{B} \bigsqcup f(\!|\mathcal{D}|\!)$ to establish continuity.

Lemma 11 *$e : E \to \_ = P \mapsto e : E \to P$ is continuous.*

Proof. We need to show

$$e : E \to \bigsqcup \mathcal{D} = \bigsqcup (e : E \to \mathcal{D}) \ ,$$

for each directed set $\mathcal{D}$. As we have noted above, it is only necessary to show that $(s, X, Y) \in \bigcap\{\mathcal{B}(e : E \to D) \mid D \in \mathcal{D}\} \Rightarrow (s, X, Y) \in \mathcal{B}(e : E \to \bigsqcup \mathcal{D})$.

¿From equation (9) on page 168, it is clear that both sets of behaviours coincide when $s = \langle\rangle$, so we need only consider the case $(\langle e\rangle^\frown s, X, Y) \in \bigcap\{\mathcal{B}(e : E \to D) \mid D \in \mathcal{D}\}$. This means that (s, X, Y) is common to every member $D \in \mathcal{D}$. But that gives $(\langle e\rangle^\frown s, X, Y) \in \mathcal{B}(e : E \to \bigsqcup \mathcal{D})$ which establishes the continuity. ⊣

Lemma 12 *$P \mapsto P \overset{\leftrightarrow}{\Box} Q = Q \overset{\leftrightarrow}{\Box} P$ is continuous when Σ is finite.*

Proof. Let $(\langle\rangle, X, Y) \in \bigcap \left\{ \mathcal{B}\left(D \overset{\leftrightarrow}{\Box} Q\right) \middle| D \in \mathcal{D} \right\}$. ¿From equation (14) on page 170, $\exists (\langle\rangle, X, Y_1) \in \mathcal{B}(D)$ and $\exists (\langle\rangle, X, Y_2) \in \mathcal{B}(Q)$ with $Y \in \overset{\checkmark\!\!\!\times}{\mathbb{P}}(Y_1 \cup Y_2)$.

Consider first the case when $Y \subseteq \Sigma$ so $Y = Y_1 \cup Y_2$. Since Y is fixed and Σ is finite, there is only a finite number of pairs (Y_1, Y_2) with $Y = Y_1 \cup Y_2$, so this is certainly true when the possibilities for Y_2 are restricted to those available from initials in Q.

Now suppose $(\langle\rangle, X, Y) \notin \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$. Then there must be a finite subset of $\mathcal{D}$ with a join which does not contain $(\langle\rangle, X, Y)$, for we can choose a member of $\mathcal{D}$ which excludes each possible $(\langle\rangle, X, Y_1)$ in turn. Since $\mathcal{D}$ is directed, the join, which contains none of the possible $(\langle\rangle, X, Y_1)$ triples, is a member of $\mathcal{D}$. But that contradicts $(\langle\rangle, X, Y) \in \bigcap \left\{ \mathcal{B}\left(D \overset{\leftrightarrow}{\Box} Q\right) | D \in \mathcal{D} \right\}$, so there is some common $(\langle\rangle, X, Y_1)$ triple, and $(\langle\rangle, X, Y) \in \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$.

Next suppose $(\langle\rangle, X, \{\checkmark\}) \in \bigcap\left\{\mathcal{B}\left(D \overset{\leftrightarrow}{\Box} Q\right) | D \in \mathcal{D}\right\}$. If $(\langle\rangle, X, \{\checkmark\}) \in \mathcal{B}(Q)$ then $(\langle\rangle, X, \{\checkmark\}) \in \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$ follows immediately. Otherwise $(\langle\rangle, X, \{\checkmark\}) \in \mathcal{B}(D)$ for each $D \in \mathcal{D}$ and $(\langle\rangle, X, \{\checkmark\}) \in \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$ again. The same is true for triples of the sort $(\langle\rangle, X, \{\text{✗}\})$.

Otherwise, when $(\langle x\rangle^\frown s, X, Y) \in \bigcap\left\{\mathcal{B}\left(D \overset{\leftrightarrow}{\Box} Q\right) \middle| D \in \mathcal{D}\right\}$ then either $(\langle x\rangle^\frown s, X, Y) \in \mathcal{B}(Q)$ in which case $(\langle x\rangle^\frown s, X, Y) \in \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$ follows immediately, or $(\langle x\rangle^\frown s, X, Y) \in \mathcal{B}(D)$ for every D. Again, $(\langle x\rangle^\frown s, X, Y) \in \mathcal{B}\left(\bigsqcup \mathcal{D} \overset{\leftrightarrow}{\Box} Q\right)$ is an immediate consequence. ⊣ ♦[12]
We omit the proofs of the following lemmas in order to keep this paper within reasonable bounds.

Lemma 13 *$P \mapsto P \overset{\leftarrow}{\Box} Q$ is continuous when Σ is finite.*

Lemma 14 *$P \mapsto Q \overset{\leftarrow}{\Box} P$ is continuous when Σ is finite.*

Lemma 15 *$P \mapsto P \underset{E}{\|} Q$ is continuous when Σ is finite.*

Lemma 16 *$P \mapsto P \overset{\leftrightarrow}{\underset{E}{\|}} Q$ is continuous when Σ is finite.*

Lemma 17 *$P \mapsto P \overset{\leftarrow}{\underset{E}{\|}} Q$ and $P \mapsto Q \overset{\leftarrow}{\underset{E}{\|}} P$ are both continuous when Σ is finite.*

Lemma 18 *$P \mapsto P \fatsemi Q$ and $P \mapsto Q \fatsemi P$ are both continuous.*

Hiding is not in general continuous:

Example 16 *Let $P_m = h^m \rightarrow Stop$ be the process that performs m copies of h before stopping. Write $D_n = \bigsqcap\{P_m \mid m \geqslant n\}$ and note that $D_n \setminus \{h\} = Stop \sqcap Spin$. $\mathcal{D} = \{D_n \mid n \in \mathbb{N}\}$ is a directed set with $\bigsqcup \mathcal{D} = \mu P \bullet h \rightarrow P$. Since $\bigsqcup \mathcal{D} \setminus \{h\} = Spin$ and $\bigsqcup\{D_n \setminus \{h\}\} = Stop \sqcap Spin$, hiding is not continuous.*

Lemma 19 *$P \mapsto P[\![R]\!]$ is continuous when Σ is finite.*

8.16.4 A Stronger Order

The refinement order works well for finite Σ but there is another stronger order:

Definition 14 $P \succcurlyeq Q \;\equiv\; \mathcal{B}(P) \subseteq \mathcal{B}(Q) \;\wedge\; clear(Q) \subseteq clear(P)$
where $clear(P) = \{(s, X, Y) \in \mathcal{B}(P) \mid (s, X, \{\text{✗}\}) \notin \mathcal{B}(P)\}$.

Throughout this paper, we have emphasised that the model is based on pure observation without any notion of what a 'reasonable' process should do. But now we have a reason to restrict our consideration to processes that are not too bizarre: only then does $\succcurlyeq$ become a Complete Partial Order for arbitrary Σ. The additional axiom is light weight: if an event that can be performed is offered on its own, then it may be accepted.

H4: $\quad (s, X, Y) \in \mathcal{B}(P) \;\Rightarrow\; \forall x \in Y \cap \Sigma \bullet (s, \{x\}, \{x\}) \in \mathcal{B}(P)$

$\succcurlyeq$ has the same bottom element as refinement and all the ordinary *CSPP* operators are $\succcurlyeq$-monotone. This order is stronger in that it relates fewer processes: this is exactly the reason why it is a Complete Partial Order when refinement is not. The directed sets that are problematical for refinement need not be considered because the members are not related by $\succcurlyeq$. But for sets that are directed under both orders, the joins are identical: thus $\succcurlyeq$ yields *exactly the same fixed points* as does refinement for recursions. $\succcurlyeq$ is an analogue of Roscoe's alternative order for the Failures-Divergences model described in [20]. It is interesting to discover that $\succcurlyeq$ does not depend upon identifying livelock with the bottom element of the order. More details of $\succcurlyeq$ will be given elsewhere.

8.16.5 A Metric

Another approach to fixed points is via a metric. The big advantage is that it yields *unique* fixed points: these are invaluable in constructing certain proofs. The disadvantage is that it is of little help when there are several fixed points. So the metric and partial order treatments are complementary.

The usual restriction space method is extended here for $\mathcal{CSPP}$. A metric is

Definition 15 $d(P_1, P_2) = \inf(\{3^{-n} \mid n \in \mathbb{N} \wedge \mathcal{B}(P_1) \Downarrow n = \mathcal{B}(P_2) \Downarrow n\} \cup \{3\})$.

Restriction metrics are generally defined using powers of 2. We depart from tradition and use 3 for a technical reason that there is no room to consider here. We merely note that this metric is complete, that prefixing and contracting recursions are contracting, and that the other standard operators apart from hiding are in general non expanding.

9 Conclusions

It has been shown that $\mathcal{CSPP}$ can be given a simple acceptance semantics which closely mirrors the properties of the standard Failures-Divergences model, while incorporating priority as well as more irregular behaviour. The version of $\mathcal{CSPP}$ involved is more abstract than those that arise from observation in which records of the histories of experiments are recorded.

I would like to thank Bill Roscoe for a variety of comments which have helped shape $\mathcal{CSPP}$, for Jeremy Martin for his initial support, to the referees for spotting a slip and to the CSP and WoTUG communities in general for their interest and input.

References

[1] C.A.R Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[2] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[3] Steve Schneider. *Concurrent and Real-time Systems*. John Wiley & Sons, Ltd., 2000.

[4] A.E. Lawrence. Extending CSP. In P. H. Welch & A. P. Bakkers, editor, *Proceedings of WoTUG 21: Architectures, Languages and Patterns*, volume 52 of *Concurrent Systems Engineering*, pages 111–131, Amsterdam, April 1998. WoTUG, IOS Press.

[5] A. E. Lawrence. CSPP and event priority. In *Communicating Process Architectures – 2001*, Concurrent Systems Engineering, pages 67–92, Amsterdam, Sept 2001. IOS Press.

[6] A. E. Lawrence. Infinite traces, Acceptances and CSPP. In *Communicating Process Architectures – 2001*, Concurrent Systems Engineering, pages 93–102, Amsterdam, Sept 2001. IOS Press.

[7] A. E. Lawrence. Successes and Failures: Extending CSP. In *Communicating Process Architectures – 2001*, Concurrent Systems Engineering, pages 49–65, Amsterdam, Sept 2001. IOS Press.

[8] A.E. Lawrence. HCSP: Extending CSP for codesign and shared memory. In *Proceedings of WoTUG 21: Architectures, Languages and Patterns*, pages 133–156. WoTUG, 1998.

[9] A.E. Lawrence. Extending CSP - even further. Communicating Process Architectures–2000, 2000. WoTUG.

[10] A. E. Lawrence. Acceptances, Behaviours and infinite activity in CSPP. In *Communicating Process Architectures – 2002*, Concurrent Systems Engineering, pages 17–38, Amsterdam, Sept 2002. IOS Press.

[11] A. E. Lawrence. HCSP, imperative state and true concurrency. In *Communicating Process Architectures – 2002*, Concurrent Systems Engineering, pages 39–55, Amsterdam, Sept 2002. IOS Press.

[12] A. E. Lawrence. Overtures and hesitant offers: hiding in *CSPP*. In *Communicating Process Architectures – 2003*, volume 61 of *Concurrent Systems Engineering*, pages 97–105, Amsterdam, Sept 2003. IOS Press.

[13] A. E. Lawrence. Observing processes. In *Communicating Process Architectures – 2004*, volume 62 of *Concurrent Systems Engineering*, pages 147–156, Amsterdam, Sept 2004. IOS Press.

[14] C.J. Fidge. A formal definition of priority in CSP. *ACM Transactions on Programming Languages and Systems*, 15(4):681–705, September 1993.

[15] Gavin Lowe. *Probabilities and Priorities in Timed CSP*. D. Phil thesis, Oxford, 1993.

[16] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002.

[17] Ian R. East. Towards a semantics for prioritised alternation. In East and Martin et al., editors, *Communicating Process Architectures 2004*, volume 62 of *Concurrent Systems Engineering*, pages 253–263. IOS Press, 2004.

[18] Ian R. East. Programming prioritized alternation. In H. R. Arabnia, editor, *Parallel and Distributed Processing: Techniques and Applications 2002*, pages 531–537, Las Vegas, Nevada, USA, 2002. CSREA Press.

[19] Ian R. East. The Honeysuckle programming language: An overview. *IEE Software*, 150(2):95–107, 2003.

[20] A.W. Roscoe. An alternative order for the failures model. In *Two Papers on CSP* [27].

[21] Andrew Butterfield and Jim Woodcock. Semantics of prialt in Handel-C. In *Communicating Process Architectures – 2002*, Concurrent Systems Engineering, pages 1–16, Amsterdam, Sept 2002. IOS Press.

[22] Jeremy Malcolm Randolph Martin. *The Design and Construction of Deadlock–Free Concurrent Systems*. PhD thesis, University of Buckingham, 1996.

[23] A.W. Roscoe, editor. *A Classical Mind*. Prentice Hall Series in Computer Science. Prentice Hall, 1994. Essays in Honour of C.A.R. Hoare.

[24] Gavin Lowe. Prioritized and probabilistic models of Timed CSP. Technical Report PRG-TR-24-91, OUCL, 1991.

[25] Gavin Lowe. Prioritized and probabilistic models of timed CSP. *Theoretical Computer Science*, 138(1), 1994. Special Issue on Mathematical Foundations of Programming Semantics conference.

[26] A.W. Roscoe. Unbounded nondeterminism in CSP. In *Two Papers on CSP* [27].

[27] Oxford University Computing Laboratory. *Two Papers on CSP*, number PRG-67 in PRG Technical Monographs, July 1988.