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

Observing Processes

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

Abstract. This paper discusses the sorts of observations of processes that are appropriate to capture priority. The standard denotational semantics for CSP are based around observations of traces and refusals. Choosing to record a little more detail allows extensions of CSP which describe some very general processes including those that include priority. A minimal set of observations yields a language and semantics remarkably close to the standard Failures-Divergences model of CSP which is described in a companion paper [1]. A slightly richer set of observations yields a somewhat less abstract language.

1 Introduction

The characterisation of processes by their externally observable behaviour is fundamental in CSP. It is a general principle that processes that cannot be distinguished by observation ought to be identified. An attempt to extend the language should entail a precise elucidation of the nature of observations. The amount of detail in observation determines the level of abstraction.

Once an observational model is established, the way is opened for a process algebra defined by a denotational semantics. The operators of the process algebra are defined compositionally by describing the observations or behaviours in terms of those of their components. This is a mapping from syntax to a representation of behaviour. The accompanying paper [1] is an example for $\mathcal{CSPP}$, a CSP like language which includes priority.

The sort of observation envisaged, and the amount of detail included, determines many of the characteristics of the resulting theory. Glabbeek has demonstrated this clearly in a series of papers: [2], [3] and [4]. He produces a taxonomy of various semantics, classified by the sort of observations admitted.

This issue is particularly acute in extending CSP because the semantics for the standard theory uses observations that collect insufficient information to capture priority. Extending the observational model is crucial in adding priority to CSP. In Fidge's approach [5], the standard description of the behaviour of a process was augmented with a *preferences* function. *preferences*(P) is a set of partial orders among events and captures the extra information about priority. However, there is no explicit discussion in [5] about the observational status of preferences. Can processes with different *preferences* always be distinguished? What observations are needed to fully characterize such processes? In Lowe's thesis, [6], a timed and prioritised CSP is presented. The behaviours have the form $(\tau, \sqsubseteq, s)$ where $\sqsubseteq$ is also essentially a partial order on events, at least at any instant. In Lowe's approach, $\sqsubseteq$ carries much of the semantic information: it determines which events can be performed by a process at any instant as well as the extra information needed to capture priority. However, the status of observation is not entirely clear.

$\mathcal{CSPP}$ is an attempt to produce an initially untimed simple and intuitive extension of CSP particularly for use in hardware compilation. It is based upon a denotational approach called acceptances which is fundamentally observational, although a thorough examination of its observational status was not conducted initially. Since the semantics is a mapping to observations, the nature of the observations must be clearly established. Then the operators can be defined in terms of the observational behaviours. This paper addresses the first stage: the companion paper [1] shows one version of the second stage.

Following this programme has been very fruitful: examining the observations appropriate for $\mathcal{CSPP}$ throws light on the approaches by Fidge and Lowe above. More importantly, it became evident that $\mathcal{CSPP}$ can describe a *far* wider class of processes that those that can be described by priority. It also allowed a simplification and clarification of the axioms.

This paper is intended to be largely self-contained, although many readers will be familiar with $\mathcal{CSPP}$ from previous papers in this conference series. Some previous exposure to CSP is assumed.

2 Observations

In CSP and most process algebras, the primary notions are those of an event and an environment. Events are observable: that already implies that they are communications with the environment. In CSP there is usually a stronger intuition that an event requires the active participation of the environment. The idea of internal events also arises, but the very term signals that they are not directly observable.

Processes as abstract models of systems are to be characterised by their behaviour: the circumstances in which they perform, or perhaps fail to perform, events needs careful examination. Such behaviour is often formulated in terms of ideal experiments. A process is somehow started, events performed perhaps as the result of some actions by the observer, and the results recorded. Just what is recorded, or what the observer is supposed to be able to distinguish, determines, among other things, the level of abstraction. A thorny issue is that of livelock. How can this be recognised? The halting problem shows that a real observer could never determine in all cases whether a system was in an infinite loop. Nevertheless, we need to represent livelock in decidable cases, so we envisage ideal 'infinite experiments' which can detect this situation.

One of the simplest sort of observation is that of a trace. The only thing that an observer records is the events that a process performs, and the sequence in which they happen. In the context of CSP, the environment must have been willing to perform the recorded events. Indeed, if the trace records are to be useful to describe the process, then all possible offers must be available. If

$$P = (a \to a \to Stop) \,\Box\, (b \to b \to Stop)$$

was not offered the possibility of performing b events, then we would have an incomplete record. Here we have

$$traces\,(P) = \{\langle\rangle, \langle a\rangle, \langle aa\rangle, \langle b\rangle, \langle bb\rangle\}\,.$$

Notice that the empty trace $\langle\rangle$ is included, although one might wonder whether that qualifies as an 'observation' here. Clearly the experimeter has performed at least two experiments: there was one while a copy of the process performed the maximal trace $\langle aa\rangle$ and another for $\langle bb\rangle$. Was this done with the 'same' process? Presumably there were experiments in which other events, c perhaps, were offered. Otherwise, it might be that the process being observed was actually

$$Q = (a \to a \to Stop) \,\Box\, (b \to b \to Stop) \,\Box\, (c \to Stop)\,.$$

So the circumstances of the experiments need clarification. What events are offered? To which copies of the process are offers made? Can a process, especially a quantum process, always be duplicated? When and how may a process silently undergo internal transformations? Is it possible to 'freeze' internal activity? How long are events offered before it is concluded that the process cannot perform an event? In the untimed theories, there is an assumption that if a process can perform an event then it will be accepted eventually. The need to offer an event and maintain that offer until it is either accepted or rejected becomes an important issue in implementing an untimed process. Can an offer be made and then withdrawn and another made?

This paper examines the sorts of observations that are suitable for capturing the behaviour of a class of processes including those which involve the use of priority.

3 Failures

The standard denotational semantics for CSP is based on Failures. These, like traces, are observations of events. The traces are recorded as before, but now the recording is enriched by noting which offers were made and rejected. This is enough to distinguish

$$P = (a \rightarrow Stop) \,\Box\, (b \rightarrow Stop) \quad \text{from} \quad Q = (a \rightarrow Stop) \sqcap (b \rightarrow Stop)$$

which share the traces $\{\langle\rangle, \langle a\rangle, \langle b\rangle\}$. A Failure is a pair (s, R) where s is a possible trace of the process in question and R is a refusal set. That is, it is an offer that the process *may* refuse after it has performed s. So Q can refuse the offer $\{a\}$ initially, that is, on the trace $\langle\rangle$: recording the empty trace is now necessary. But P cannot refuse $\{a\}$.

Notice that the information recorded is sparse: only the traces, and *only* the offers that are refused after a particular trace. This is just enough information to distinguish P from Q above, and accounts for the very abstract nature of CSP. This leads to results like

$$(a \rightarrow Stop) \,\Box\, (b \rightarrow Stop) \;\sqcap\; (a \rightarrow Stop) \;\sqcap\; (b \rightarrow Stop) = (a \rightarrow Stop) \;\sqcap\; (b \rightarrow Stop) \quad (1)$$

in standard CSP. The presence of the external choice on the left hand side cannot be unambiguously detected in the presence of the internal choices by such observations.

The standard model for CSP does also record when a process can livelock, or diverge. We admit ideal infinite experiments as a sort of 'observation' to permit that.

4 Priority

Priority is a way of expressing a preference when more than one course of action is available. In the context of process algebra, this involves a selection when more than one event is possible.

Any general theory of shared events which includes priority must address the issue of contention. The decision of whether an event may be performed in general requires arbitration or negotiation: information must be exchanged before an event happens. There must be

1. a declaration;
2. arbitration or negotiation;
3. and a possible action.

A full theory cannot abstract entirely from these matters. In particular, useful systems will normally connect compliant processes to those that employ priority. A compliant process is one which is neutral with respect to available events: it will conform to the preference expressed by a partner. More importantly, compliant processes offer a choice of events to the environment. And since an environment is no more than a way of capturing the behaviour of parallel partners, a theory without compliant processes is deficient. If an enviroment can offer a choice, that must come about because there are processes which can do the same. Clearly the observations must be sufficient to distinguish and characterise compliant processes as well as those which express priority. In this sense, observations of a process have to include some elements of the declaration phase.

The observations of acceptances below seem to be as sparse as possible while fulfilling these requirements. Actually, the declaration phase can be inferred for some classes of processes. The experiments to be performed are similar to those described earlier, but now more details are recorded. The events actually performed are still recorded as traces, but now the offers are also recorded in every case, not just those that are refused. But there is more: in addition, the response is also recorded. That is the records take the form of triples (s, X, Y) where X is an environmental offer made after the process has performed the trace s, and Y is the response. It is most natural to assume that Y is directly observable, and this a very reasonable view of both hardware and software systems that employ priority. However, this is not strictly necessary for one can conduct additional experiments and observe which events can be performed after s and the offer X provided a suitable replication facility is available. This is necessary because nondeterministic processes may have more than one response in the sense that (s, X, Y_1) and (s, X, Y_2) may both be possible, and it must be possible to reliably 'freeze and replicate' if these two responses are to be distinguished properly. The reason why this is not an unreasonable expectation will be clearer in a moment. But it is far simpler and entirely realistic to assume that the responses Y_1 and Y_2 are directly observable which we assume below.

An important point is that the response from a process is reliable. A process that declares to the environment which events it is prepared to perform and then does not honour the commitment is of no use. Processes can be very nondeterministic, and resolve that nondeterminism partially or fully at any time *except* between declaring a response and performing one of the declared events. Processes that do not do that have no place in a system employing priority. In this sense a process must freeze its internal activity between responding to an offer and either performing an event or responding to a further offer. But if an offer is repeated, the process need not give the same response.

Our standard example is $AB = (a \rightarrow Stop) \overleftarrow{\Box} (b \rightarrow Stop)$ which is represented by the triples

$$\begin{aligned}\{(\langle\rangle, \{a,b\}, \{a\}), (\langle\rangle, \{a\}, \{a\}), (\langle\rangle, \{b\}, \{b\}), (\langle\rangle, \emptyset, \emptyset),&\\ (\langle a\rangle, \{a,b\}, \emptyset), (\langle a\rangle, \{a\}, \emptyset), (\langle a\rangle, \{b\}, \emptyset), (\langle a\rangle, \emptyset, \emptyset),&\\ (\langle b\rangle, \{a,b\}, \emptyset), (\langle b\rangle, \{a\}, \emptyset), (\langle b\rangle, \{b\}, \emptyset), (\langle b\rangle, \emptyset, \emptyset)\}\,,&\end{aligned}$$

when the only events are a and b. An alternative notation for the triple (s, X, Y) is $s : X \rightsquigarrow Y$, so the triples above can be summarised as

$$\begin{aligned}\langle\rangle : \quad & X \rightsquigarrow \{a\} \blacktriangleleft a \in X \blacktriangleright X \cap \{a,b\}\\ \langle a\rangle : \quad & X \rightsquigarrow \emptyset\\ \langle b\rangle : \quad & X \rightsquigarrow \emptyset\,.\end{aligned}$$

AB gives priority to $(a \rightarrow Stop)$, so if it is given an offer X which includes both a and b, it will respond with just $\{a\}$. It will only accept b when a is not also offered.

Readers not familiar with $\mathcal{CSPP}$ may find helpful the contrast between the example of the compliant variant $C = (a \to Stop) \overset{\leftrightarrow}{\Box} (b \to Stop)$ and

$$\begin{array}{lccl} \langle\rangle : & X & \rightsquigarrow & X \cap \{a, b\} \\ \langle a\rangle : & X & \rightsquigarrow & \emptyset \\ \langle b\rangle : & X & \rightsquigarrow & \emptyset \end{array}$$

When offered both a and b, it replies with the compliant $\{a, b\}$.

To illustrate the exchange of information, which in general may involve arbitration or contention, consider C and AB running in parallel, synchronising on all events: $AB \parallel C$ or explicitly

$$\Big((a \to Stop) \overset{\leftarrow}{\Box} (b \to Stop)\Big) \parallel \Big((a \to Stop) \overset{\leftrightarrow}{\Box} (b \to Stop)\Big) \ . \tag{2}$$

If the environment offers $\{a, b\}$ the responses from the two components are $\{a, b\}$ and $\{a\}$, so a occurs. In fact the process in equation (2) is just AB again. In contrast,

$$\Big((a \to Stop) \overset{\leftarrow}{\Box} (b \to Stop)\Big) \parallel \Big((a \to Stop) \overset{\rightarrow}{\Box} (b \to Stop)\Big) \ . \tag{3}$$

the two components in equation (3) answer with $\{a\}$ and $\{b\}$ and there is a disagreement. So we have *contention* and may need *arbitration*. Our observational model does not determine the outcome of this priority conflict: that is a matter of the semantics of the operators, primarily of $\parallel$. One answer is given in the second paper [1]: it is deadlock. But there are other versions of $\mathcal{CSPP}$ in which the result is a nondeterministic choice.

In passing, note that Lowe has very different forms of parallel operators. His parallel operators are always biased, even on shared – synchronised – events: this arises from his desire to remove all except one form of nondeterminism in order to simplify the transition to a probabilistic theory.

In both equations (2) and (3), it is clear that information must be exchanged between the components and the environment to determine which events, if any, can be agreed by the processes. Implementations may involve a scheduler which needs to discover which processes are ready to do what, or arbitration hardware may be involved.

It is in this sense that we contend that there is a need for some abstract form of declaration and arbitration or negotiation in general before events can be selected.

5 Experiments

Ideal experiments are performed on a process: it is made an offer and eventually produces a response which is normally a set of events but may be instead an indication of termination or (perhaps after an infinite wait) of livelock. The environment may now either change the offer, or select an event from the response to jointly perform. If the environment takes the former course and makes another offer, the first offer is termed *hesitant*. Hesitant offers model situations where an environment negotiates with a process to find a mutually agreed event. We noted earlier that the process must be stable when it produces a response in the sense that it is prepared to perform any of the events in that response. But it is allowed to undergo internal transitions when a new offer is made: hiding can produce processes which are naturally understood in this way. Consequently a single copy of a process which has performed a trace s can be nondeterministic about the response to a given offer.

These ideal experiments include observations of infinite duration, and detection of livelock may require such. But some processes with knowledge of their internal construction can immediately declare livelock in appropriate circumstances.

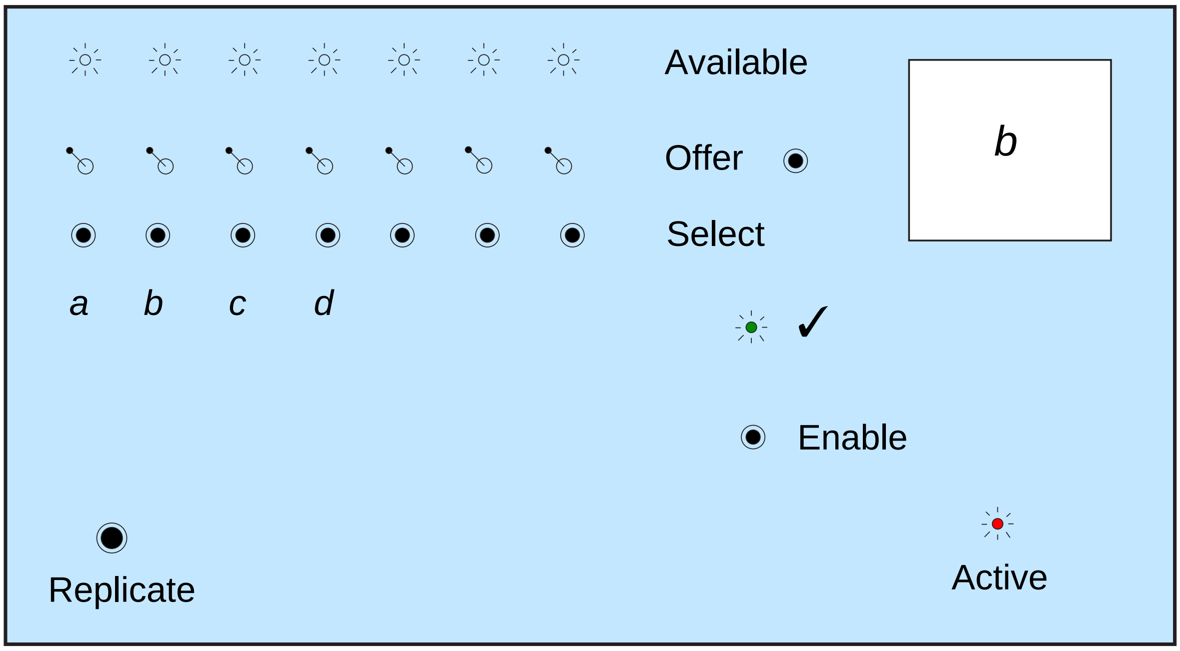

Figure 1: A machine which displays an acceptance matching an offer.

6 A Machine

In [4], [7] and particularly [2] R.J.Glabbeek describes and classifies a series of ideal machines to capture observations of processes. Figure 1 shows a machine in this spirit for observing $\mathcal{CSPP}$ processes. The machine has three rows: the first labelled *Available*, of lights; the second labelled *Offer*, of switches; and the third labelled *Select*, of buttons. Each column matches an event. The top row of lights displays the events which can be performed in the context of the second row switch settings. Valid processes will only display a light matching a switch which is on. The top row is continuously updated to match the setting of the second row switches while the process is waiting for the environment to initiate an event. That is the function of the third row of buttons. Pressing a single button matching an illuminated *Available* light results in that event happening. The display in this machine is largely redundant, but it shows each event as it occurs as in Glabbeek's machines. The machine has a green light marked $\checkmark$. This illuminates when the process inside the machine is prepared to terminate. No other light can be on at the same time as the $\checkmark$ light. Pressing the *Enable* button at such a moment will allow termination and end the experiment: $\checkmark$ will be shown on the display. Otherwise further offers can be explored. There is also a red light to indicate internal activity: in Glabbeek's machines, this light is green. The *Replicate* button, present in many of Glabbeek's machines, produces a cascade of copies of the machine and its contents in the current state.

It is a misuse of the machine to press a *Select* button when the associated light is off: if the button is so pressed, nothing will happen. The machine might be designed to lock buttons at these times to eliminate this eventuality. If the *Replicate* button is pressed while the active light is off, then the machine and its contents in its current state is copied.

It is clear that the switch setting corresponds to X and the state of the lights to Y in (s, X, Y). The machine is used as follows.

1. When a process starts, the active light is on. Eventually, a response is produced on the event lights including the $\checkmark$ light. The active light turns off at this point and enables a

refusal to be recognised. Livelock is recognised in an ideal infinite experiment by the active light staying on.

2. The switches may now be updated in which case the active light turns on at least briefly, and a new response displayed.

3. When a response is displayed and the *Active* light is off, a *Select* button matching an illuminated event light may be pressed: the event then happens. If the ✓ light is on, then the *Enable* button may be pressed to terminate the experiment. In an ideal infinite experiment, the *Enable* button may be pressed after livelock has been detected: this will also complete the experiment after another infinite time. Some processes with knowledge of their internal construction can predict livelock, and an extra light could be added to the machine to permit this to be passed to the environment as a normal response in a similar way to the ✓ light.

The recordings made in such experiments are just the (s, X, Y) trace–offer–response triples. The number or order in which the offers are made at a particular s in an experiment is not recorded.

If it is known that a system to be modelled does not involve hesitant offers, then a restricted set of observations can be made. After an event has occurred, the new offer must be set up instantly on the switches. Otherwise an additional *sample* button can be added to introduce each offer. Clearly the set of acceptances collected in such observations will be a subset of those when hesitant offers are included.

7 A Minimal Record: Triples

Recording only the triples as above provides the maximum abstraction with the minimum assumptions about the nature of the processes being observed. Consider the processes

$$\begin{aligned} P_1 &= \overleftrightarrow{\{a, u\}} \rightarrow \overleftrightarrow{\{b, v\}} \rightarrow Stop \\ P_2 &= \{[\{a\} > \{u\}]\} \rightarrow \{[\{b\} > \{v\}]\} \rightarrow Stop \\ P_3 &= \overleftrightarrow{\{a, u\}} \rightarrow \{[\{b\} > \{v\}]\} \rightarrow Stop\,. \end{aligned} \tag{4}$$

The notation $\overleftrightarrow{E} \rightarrow P$ is an abbreviation for the compliant prefixing $x : \overleftrightarrow{E} \rightarrow P$ which is a process which is prepared to perform any of the events from the set E and thereafter behave like the process P: its initial triples have the form $(\langle\rangle, X, X \cap E)$. If it is offered several events in E, it accepts all of them. So P_1 is a process which is always compliant: initially it is prepared to perform either a or u according to the wishes of its environment. On the next step it will perform either b or v before stopping.

The notation $\{[\{a\} > \{u\}]\} \rightarrow P$ is an abbreviation for the biased prefixing $x : \{[\{a\} > \{u\}]\} \rightarrow P$ which starts with a preference for a rather than u, so its initial triples have the form $(\langle\rangle, X, \{a\} \blacktriangleleft a \in X \blacktriangleright (X \cap \{u\}))$.

Thus P_2 has the same traces and possible events as P_1, but on the first step it gives priority to a and on the second priority to b. P_3 is like P_1 on the first step, and then behaves like P_2. Now consider $P_1 \sqcap P_2$. If our only observations are of triples, then $P_1 \sqcap P_2$ can behave like P_3. So

$$P_1 \sqcap P_2 = P_1 \sqcap P_2 \sqcap P_3\,.$$

The reason that these processes cannot be distinguished is that the only 'history' in our triples are the traces, and P_1, P_2 and P_3 share these. Should $P_1 \sqcap P_2$ be allowed to behave like

P_3? We may have processes which defer the resolution of internal indeterminism so that the behaviour after each event is only selected at that point, perhaps according to the outcome of a radio-active decay. Then one might argue that a specification $P_1 \sqcap P_2$ should permit a behaviour matching P_3. A less abstract view in which a richer set of records is included is described below. That distinguishes $P_1 \sqcap P_2$ from $P_1 \sqcap P_2 \sqcap P_3$ while still permitting the extreme form of late resolution of nondeterminism to be properly described.

The very abstract approach using just triples is in the spirit of the Failures-Divergences model of standard CSP, has most properties in common, but includes priority. In fact, it goes far beyond processes that can be described with priority or compliance. It describes almost any process that can respond to an offer, produce a reliable response and perform events.

The version of $\mathcal{CSPP}$ based on triples constrains processes in the least possible way. The set of triples representing a process P is written as $\mathcal{B}(P)$. The triples have type

$$\mathcal{A} = \{(s, X, Y) \mid s \in \Sigma^* \wedge X \subseteq \Sigma \wedge (Y \subseteq X \vee Y = \{\checkmark\} \vee Y = \{\text{✗}\})\} \,. \qquad (5)$$

Σ is the set of all possible events, excluding the ‘tokens’ $\checkmark$ and ✗ which are used to represent termination and livelock. Σ^* is the standard notation for the set of all finite traces, including $\langle\rangle$, drawn from Σ. $\mathcal{B}(P) \subseteq \mathcal{A}$ ensures that the response Y to an offer X is either a subset of the events offered in Y, or is an indication of termination or livelock. The axioms are intended to be minimal:

1. $\langle\rangle \in traces(P)$.

2. There is a triple $(s, X, Y) \in \mathcal{B}(P)$ for each offer X.

3. Traces and responses match properly:
   $s^\frown\langle x\rangle \in traces(P) \Leftrightarrow \exists (s, X, Y) \in \mathcal{B}(P) \bullet x \in Y \cap \Sigma$.

The requirement $\mathcal{B}(P) \subseteq \mathcal{A}$ might be regarded as an additional axiom.

8 Recording History: Behaviours

Recording only the triples (s, X, Y) gives a very abstract form of $\mathcal{CSPP}$ which turns out to closely mirror the standard Failure-Divergence semantics of standard CSP. However, there are some slightly disturbing features of the model: in particular, several of the operators including some forms of parallel and sequential composition and hiding fail to distribute over $\sqcap$. This can be understood as a side effect of recording only the traces that lead up to a particular offer and response. Recording a little more information, namely the history of an experiment as the set of responses along a trace turns out to have several advantages. We collect *behaviours* for each experiment. An experiment follows a process along some trace. At each point in the evolution a record of the responses is made. That is, we have a relation, a set of pairs (X, Y) at each point along some trace s. This means that a behaviour b, the record of the evolution along some trace s, is a function $b : {\downarrow}s \to \left(\mathbb{P}\Sigma \leftrightarrow \overset{\checkmark\text{✗}}{\mathbb{P}}\,\Sigma^{\checkmark\text{✗}}\right)$. The downset ${\downarrow}s = \{t \mid t \leqslant s\}$ is the set of all prefixes of s: this is a standard notation from the theory of partial orders [8]. $\mathbb{P}S$ is the power set: $\mathbb{P}S = \{U \subseteq S\}$, and $\overset{\checkmark\text{✗}}{\mathbb{P}}\,S$ adds the sets $\{\checkmark\}$ and $\{\text{✗}\}$ so $\overset{\checkmark\text{✗}}{\mathbb{P}}\,S = \mathbb{P}S \cup \{\{\checkmark\}, \{\text{✗}\}\}$. The set of of binary relations between sets S and T is written $S \leftrightarrow T$ so it is an alias for $\mathbb{P}(S \times T)$, [9]. Thus $\mathbb{P}\Sigma \leftrightarrow \overset{\checkmark\text{✗}}{\mathbb{P}}\,\Sigma^{\checkmark\text{✗}}$ is the set of relations consisting of pairs (X, Y) with $X \subseteq \Sigma$ and Y which is either also a subset of Σ, in our case $Y \subseteq X$, or may be one of the singleton sets $\{\checkmark\}$ and $\{\text{✗}\}$. Hence $b : {\downarrow}s \to \left(\mathbb{P}\Sigma \leftrightarrow \overset{\checkmark\text{✗}}{\mathbb{P}}\,\Sigma^{\checkmark\text{✗}}\right)$ records the history of the observations of an experiment which ended after the $\{(X, Y)\}$ offer-response relation on s was recorded. A process is characterised as a set of such behaviours.

Once again, the sort of process to be described is not artificially constrained. The axioms are now

1. $t \in \downarrow s \;\Rightarrow\; \downarrow t \lhd b \in \mathcal{B}(P)$.

2. $s^\frown\langle x\rangle \in traces(b) \Rightarrow \exists(X, Y) \in bs \bullet x \in Y$
   and
   $s \in traces(b) \wedge \exists(X, Y) \in bs \bullet x \in Y \Rightarrow$
   $\exists\, b' \in \mathcal{B}(P) \bullet traces\,(b') = \downarrow(s^\frown\langle x\rangle) \wedge b < b'$.

The first axiom requires that we keep records of each prefix of s as a separate experiment. $\lhd$ is domain restriction. The second axiom requires that responses and traces match properly. If the experiment ends when a real event could have been accepted, then there is another behaviour b' which describes the extension of the experiment. As before, the type constraint on $b \in \mathcal{B}(P)$ may be regarded as an extra implicit axiom.

Recording the history of observations in this way now allows us to distinguish $P_1 \sqcap P_2$ from $P_1 \sqcap P_2 \sqcap P_3$ where the processes are given by equation (4). Extreme processes that do not resolve internal nondeterminism until the last moment can still be described, but now we can tell when they are present.

9 Internal and External Nondeterminism

It is evident that the response Y to an offer X is a declaration by a process of which events it is prepared to perform. Whenever the response contains more than one event, this is an invitation to the environment to make a choice. This is an example of *external* nondeterminism. It is an essential feature of useful systems employing priority: it must be possible to arrange synchronisation of events when one partner has a preference.

The tokens ✓ and ✗ have a special status here. It makes no sense to offer the environment a choice between these pseudo-events and the ordinary 'real' events in Σ. Thus in all our models and experiments, these tokens may only appear as a 'response' as the singletons $\{\checkmark\}$ and $\{✗\}$.

Internal nondeterminism is manifested when there is more that one response possible to a given offer X. The standard examples are

$$E = (a \rightarrow Stop) \overset{\leftrightarrow}{\Box} (b \rightarrow Stop)$$
$$I = (a \rightarrow Stop) \sqcap (b \rightarrow Stop)$$

The compliant E manifests external, and I internal, nondeterminism. Internal nondeterminism is more extreme in that it is outside environmental control.

10 Conclusions

The status of observations which can capture very general processes including those which can express priority have been examined. The minimal axioms that naturally arise for a couple of levels of abstraction have been described. Versions of $\mathcal{CSPP}$ based on such observations can describe a very wide class of real time systems: some rather irregular processes which cannot be described by any conventional form of priority are included. The companion paper [1], outlines the most abstract of these which bears a remarkable congruence with the standard Failures-Divergence model of CSP.

References

[1] A. E. Lawrence. Triples. In *Communicating Process Architectures – 2004*, volume 62 of *Concurrent Systems Engineering*, pages 157–184, Amsterdam, Sept 2004. IOS Press.

[2] R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, Proceedings *CONCUR'93*, 4*th* International Conference on *Concurrency Theory*, Hildesheim, Germany, August 1993, volume 715 of LNCS, pages 66–81. Springer-Verlag, 1993.

[3] R.J. van Glabbeek. What is branching time semantics and why to use it? In M. Nielsen, editor, *The Concurrency Column*, pages 190–198. *Bulletin of the EATCS* 53, 1994. Also available as Report STAN-CS-93-1486, Stanford University, 1993, at `http://theory.stanford.edu/branching/`, and in G. Paun, G. Rozenberg & A. Salomaa, editors: *Current Trends in Theoretical Computer Science; Entering the 21st Century*, World Scientific, 2001.

[4] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001. Available at `http://boole.stanford.edu/pub/spectrum1.ps.gz`.

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

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

[7] R.J. van Glabbeek. What is branching time semantics and why to use it? In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science; Entering the 21st Century*. World Scientific, 2001. Also available as Report STAN-CS-93-1486, Stanford University, 1993, at `http://theory.stanford.edu/branching/`, and in M. Nielsen, editor: *The Concurrency Column*, *Bulletin of the EATCS* 53, 1994, pp. 190–198.

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

[9] M. J. Spivey. The Z notation: A reference manual. http://spivey.oriel.ox.ac.uk/˜mike/zrm/.

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

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