Communicating Process Architectures – 2003
Jan F. Broenink and Gerald H. Hilderink (Eds.)
IOS Press, 2003

Overtures and hesitant offers: hiding in $\mathcal{CSPP}$

A.E. LAWRENCE
Department of Computer Science, Loughborough University, Leicestershire, LE11 3TU UK
A.E.Lawrence@lboro.ac.uk

Abstract. Hiding is an important and characteristic part of CSP. Defining it in the presence of priority for $\mathcal{CSPP}$ needs care. The ideas of overtures and hesitant offers introduced here arise naturally in the context of Acceptances. They provide clear insight into the behaviour of hidden processes. And particularly illuminate the origin of the nondeterminism which frequently arises from hiding.

Keywords: CSP; CSPP; Denotational semantics; formal methods; concurrency; parallel systems; occam; hardware compilation; priority; hiding.

1 Introduction

Hiding is the basic abstraction mechanism in CSP[1, 2]. A child buys an ice cream:

$$Buy = pay \to record \to ice\_cream \to Stop$$

Buy is the process in which the money is first handed over, then the transaction is recorded, and finally the ice cream is handed over before the shop shuts. The child is unlikely to be very interested in the event *record*: the fact that the shop needs to keep track of the stock and finance is of no immediate concern. This is irrelevant detail: so the child is likely to dismiss the *record* event as irrelevant. The child's view will be

$$Buy \setminus \{record\} = pay \to ice\_cream \to Stop \quad .$$

The notation is $P \setminus H$ where P is a process and H is the set of events to be hidden. The explicit set notation is sometimes omitted when there is no ambiguity, so $Buy \setminus \{record\}$ might be written as $Buy \setminus record$.

In **occam**, we might have something like

```
CHAN OF money payment:
CHAN OF goods    hand:

--  data events hidden here
CHAN OF transaction data:
PAR
  database(data)
  ...
  SEQ
    payment ? pay
    ...
    data ! details
    ...
    hand ! ice.cream
```

So hiding can be used to model the effect of scope in **occam**. In π-calculus[3, 4], there is a notion of *restriction* which uses the keyword **new**: this is the nearest that π-calculus comes to hiding. And it matches the conventional idea of scope rather closely. In that respect it is a little like **occam** in that hiding does not appear explicitly, but is implicit in the idea of scope. But when we need to prove properties using CSP, hiding is very important: the proof of the correcteness of the implementation of JCSP[5, 6] and CTJ channels in [7] uses hiding repeatedly. Indeed it is absolutely fundamental in showing the equivalence between real channels and the JCSP and CTJ implementations with the internal details hidden. A particularly simple example, very like the child buying an ice cream, appears on page 291 of [7]:

$$(write.a!x \to a!x \to ack.a \to PROCESS) \setminus \{write.a, ack.a\} = a!x \to PROCESS$$

CSP seems to be the only process algebra with such an explicit and comprehensive form of hiding. That in turn guarantees that CSP must also include explicit nondeterminism. A simple example is

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

$(a \to c \to Stop) \,\Box\, (b \to d \to Stop)$ is a process which is driven by its environment – its parallel partners – into performing *either* the event a *or* the event b initially. It depends on which is offered first: for the moment we only consider offers of single events. If a and b are hidden, they become 'internal': that is unconstrained by the environment. If we are concerned with software, it may depend on the details of a particular implementation which event becomes available first. CSP is intended to cover all such possibilities: it leaves open exactly what happens internally in such cases. Thus

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

where $\sqcap$ is *internal choice*, the primary operator capturing the fullest form of nondeterminism. The point of this example is that the presence of hiding in the language inevitably requires a matching operator $\sqcap$, even if we did not need it for anything else. That in turn ensures that refinement is an inherent part of CSP: we will return to that below.

A pseudo-occam model of $((a \to c \to Stop) \,\Box\, (b \to d \to Stop)) \setminus \{a, b\}$ using booleans is

```
CHAN OF BOOL a,b :
PAR
  PAR
    a ! TRUE
    b ! TRUE
  BOOL x,y :
  ALT
    a ? x
      c ! x
    b ? y
      d ! y
```

Here we model the internal environment which makes the events a and b available in an unconstrained way by

```
PAR
  a ! TRUE
  b ! TRUE
```

which can be implemented in either order on a sequential machine. Without knowledge of the implementation, and perhaps even with such knowledge, we cannot predict whether the occam fragment will communicate on the `c` or on the `d` channel. Of course, this `PAR` will not terminate in the fragment above: only one of the communications will happen, but this does not matter for the purposes of illustration. Notice that even if `ALT` was replaced by `PRI ALT` it still would not determine whether `c` or `d` was used. But things would be different if in addition `PAR` was changed to `PRIPAR`: examining the effects of priority in hiding is one of the aims of this paper.

A conceptual picture of $P \setminus H$ is of the process P placed inside a box with walls opaque to the events of H. And inside the box, the events of H are entirely free to occur: the only constraints are those imposed by P itself. External events like c and d above can 'pass through' the walls and involve interaction with the outside world: they are not directly influenced by the box. Any external events from H which might be offered are simply ignored: they cannot penetrate the box, and their internal counterparts cannot be seen outside.

The 'bubble diagrams' that most **occam** programmers draw when designing their systems use hiding. It is fundamental to compositional design in which only external behaviour need be considered when putting processes together. Figure 1 is a bubble diagram for an occam

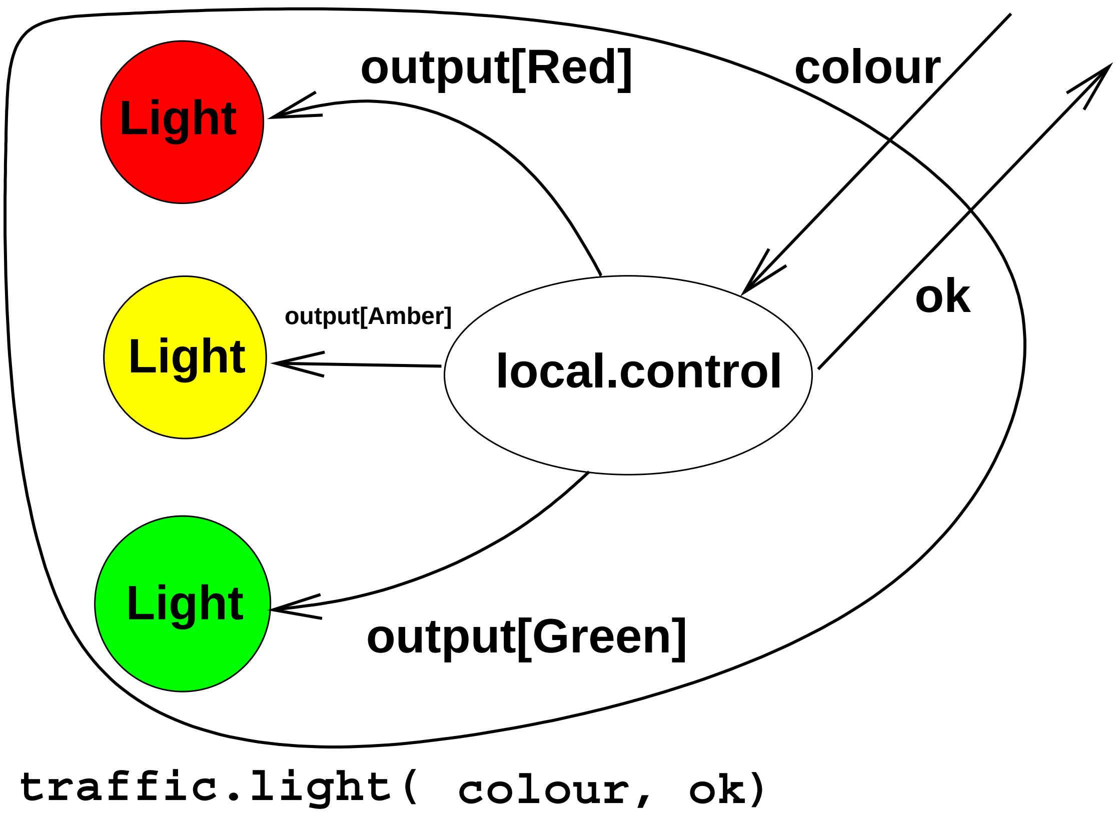

Figure 1: A 'bubble' diagram

model for a single traffic light. Here some of the internal details are shown. But it is understood that for external purposes only the two channels `colour` and `ok` need be considered. In fact the outer boundary matches the idea of the 'box' in the conceptual model above. It is understood that everything wholly contained in the outer bubble is irrelevant and may be substituted by a different implementation provided the same external behaviour is maintained. Indeed the 4 inner bubbles show no internal details: they are already hidden.

So hiding provides

- abstraction, fundamental to compositional design and comprehension; and
- isolation. The external environment cannot interfere with the internals of a hidden process.

Some cases of hiding are less than obvious: in standard CSP

$$((a \to P) \Box (b \to Q)) \setminus \{b\} = Q' \ \sqcap \ ((a \to P') \Box Q') = ((a \to P') \sqcap Stop) \Box Q' \qquad (1)$$

where $P' = P \setminus \{b\}$ and $Q' = Q \setminus \{b\}$. We will discover that this result is slightly modified in the presence of priority because we have more expressive power. And we must address cases like $\Big((a \to Stop) \overleftarrow{\Box} (b \to Stop)\Big) \setminus \{b\}$.

Hiding also reveals that any complete theory must handle livelock. What happens if we hide a in $\mu\, p \bullet a \to P$, the process $P = a \to P$ which engages in a continuous stream of a's? It is the essence of divergence, and clearly we must have

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

In **occam** we might have something like

```
PROC diverge()
  CHAN OF BOOL c:
  BOOL any:
  WHILE TRUE
    PAR
      c ! TRUE
      c ? any
:
```

The purpose of this paper is to investigate how hiding is modified when priority is included in $\mathcal{CSPP}$[8, 9] and therefore also in $\mathcal{HCSP}$[10, 11]. The next section gives a quick summary of $\mathcal{CSPP}$. Then the Acceptance denotational semantics is surveyed. The ideas of overtures and hesitant offers are then introduced as precise ways to understand hiding with Acceptances and priority. The culmination is the formal definition of hiding in $\mathcal{CSPP}$. Finally we illustrate how this works on some illuminating examples.

2 $\mathcal{CSPP}$

$\mathcal{CSPP}$ is CSP extended to include priority as it occurs in standard **occam**. The `ALT` constructor of **occam** matches external choice, $\Box$ of CSP, at least for guarded processes of the form $c?x \to P(x)$ and for *Skip*. **occam** also includes `PRI ALT` in which textually earlier guards have priority over later guards. $\mathcal{CSPP}$ introduces $\overleftarrow{\Box}$ to capture this: it is a general variant of $\Box$. $P \overleftarrow{\Box} Q$ can also be written as $Q \overrightarrow{\Box} P$.

In **occam**, `PRI ALT` is a refinement of `ALT`: that is, it is a possible implementation. Indeed most **occam** compilers have only the one primitive `PRI ALT` which is also used to implement `ALT`. So in this context, it would be natural to write

$$P \Box Q = P \overleftarrow{\Box} Q \ \ \sqcap \ \ P \overrightarrow{\Box} Q$$

and this is what an early version of $\mathcal{CSPP}$ did. This means that the only possible implementations of $\Box$ were biased. As first pointed out by Bill Roscoe, this is at variance with the intuition of the majority of those who do not use **occam** regularly: they normally think of $\Box$

as a neutral choice operator. So $\mathcal{CSPP}$ also includes a *compliant* version of external choice written naturally enough as $\overleftrightarrow{\Box}$. This is another refinement of $\Box$ in the sense that

$$P \Box Q = P \overleftarrow{\Box} Q \quad \sqcap \quad P \overleftrightarrow{\Box} Q \quad \sqcap \quad P \overrightarrow{\Box} Q$$

The idea of *compliance* now has an important role in understanding $\mathcal{CSPP}$: see [8, 9].

From the **occam** perspective, both $\overleftarrow{\Box}$ and $\overrightarrow{\Box}$ are refinements of $\Box$ in the sense above. This gives our usual embedding of standard CSP in $\mathcal{CSPP}$. However, there is another simpler embedding in which we *identify* $\Box$ with $\overleftrightarrow{\Box}$. This matches standard CSP and the intuition of many practitioners who use CSP primarily for specification and model checking. But that identification implies that **occam** need not be an implementation of (part of) CSP! Needless to say, we normally work with the first embedding, but the second is simpler and throws a good deal of light on standard CSP. It is useful when priority is not needed.

occam also includes `PRI PAR`, a prioritized version of the concurrency constructor. For completeness $\mathcal{CSPP}$ includes and extends this idea with $\overleftarrow{|||}$, $\overleftrightarrow{|||}$ and $\overrightarrow{|||}$ which are like $\overleftarrow{\Box}$, $\overleftrightarrow{\Box}$ and $\overrightarrow{\Box}$, but operate on every event rather than on just the initial events of a process.

There are really only these two ideas of priority and compliance in the extension from CSP to $\mathcal{CSPP}$, but various derived operators also get biased or compliant variants. So $x : \overleftrightarrow{\{a, b, c\}} \rightarrow Stop$ is a process which is neutral about whether it starts with a or b or c. Whereas $x : \{a, b, c\} \rightarrow Stop$ might have an implementation which gave priority to b.

3 Acceptances

In order to define $\mathcal{CSPP}$ properly, a new *Acceptance* denotational semantics was invented. This turned out to have unexpected merits and enabled some longstanding problems in describing certain sorts of infinite behaviour to be solved for standard CSP as well as for $\mathcal{CSPP}$. It is described in [8] and [9], so we only highlight some key points here.

The basic idea is to *offer* a process P a set of mutually exclusive events, and record which are *accepted*. So if $a \rightarrow Stop$ is initially offered the set of events $\{b, c, d\}$, nothing is accepted. The response, a refusal in this case, is the empty set. We write that as $\langle\rangle : \{b, c, d\} \rightsquigarrow \emptyset$. $\langle\rangle$ is the empty trace and indicates that we are at the initial stage of evolution before the process has performed any events. If we offer $\{a, b, c\}$ then a is accepted: $\langle\rangle : \{a, b, c\} \rightsquigarrow \{a\}$. In these circumstances, the process can perform a which is recorded in the trace as $\langle a\rangle$. After that, the process behaves like *Stop* and refuses to do anything else: $\langle a\rangle : X \rightsquigarrow \emptyset$ for any offer X.

The idea was to capture priority as in $(a \rightarrow Stop) \overleftarrow{\Box} (b \rightarrow Stop)$ which gives $\langle\rangle : \{a, b\} \rightsquigarrow \{a\}$ in contrast to $(a \rightarrow Stop) \overleftrightarrow{\Box} (b \rightarrow Stop)$ for which $\langle\rangle : \{a, b\} \rightsquigarrow \{a, b\}$. Taking this last compliant process as an example, it is represented as a single *behaviour p*. p is a function: its domain is the set of traces that the process can perform. In this case $\operatorname{dom} p = traces(p) = \{\langle\rangle, \langle a\rangle, \langle b\rangle\}$: in any execution at most one of $\langle a\rangle$ and $\langle b\rangle$ will be involved. p maps a trace to another function: that which records the acceptances. Thus $p(\langle a\rangle)$ is just the function with typical maplet $X \mapsto \emptyset$ which we wrote above as $X \rightsquigarrow \emptyset$. So p has type $\Sigma^* \nrightarrow (\mathbb{P}\,\Sigma \rightarrow \mathbb{P}\,\Sigma^{\checkmark\text{✗}})$, where Σ is the set of all events. Σ^* is the set of all possible traces: notice that we are now regarding p as a partial function since it is only defined on a subset of 3 traces. $\mathbb{P}\,\Sigma$ is the set of all possible offers which can be made to p. And the response is drawn from $\mathbb{P}\,\Sigma^{\checkmark\text{✗}}$ which is similar except that there is also the possibility of either $\checkmark$ representing termination or ✗
p.

In general, in acceptance semantics, a process ‘consists of’ multiple behaviours. A very simple case is $Stop \ \sqcap \ (a \to Stop)$. This has two behaviours. One matching $Stop$ for which $\langle\rangle : X \rightsquigarrow \emptyset$ and the other matching $a \to Stop$ for which $\langle\rangle : X \rightsquigarrow X \cap \{a\}$ and $\langle a \rangle : X \rightsquigarrow \emptyset$.

These very simple ideas are all that are involved in acceptances. But we do need to state the axioms which are all rather obvious given familiarity with the notation. We write the set of behaviours defining a process P as $\mathcal{B}P$, and functions are written in Curried form.

H1: $\forall b \in \mathcal{B}P \bullet \quad \langle\rangle \in traces(b)$

H2: $\forall b \in \mathcal{B}P \bullet \forall s \in \Sigma^* \bullet \forall x \in \Sigma \bullet$
$$s^\frown\langle x\rangle \in traces(b) \Leftrightarrow s \in traces(b) \wedge (\exists X \subseteq \Sigma \bullet x \in bsX)$$

H3: $\forall b \in \mathcal{B}P \bullet \forall s \in traces(b) \bullet \forall X \subseteq \Sigma \bullet \quad bsX \subseteq X^{\checkmark\text{✗}}$

H4:
$$\begin{array}{c} \forall b \in \mathcal{B}P \bullet \forall s \in traces(b) \bullet \forall X, Y \subseteq \Sigma \bullet \\ bsX = \emptyset \wedge Y \subseteq X \Rightarrow bsY = \emptyset \\ \wedge \\ bsX \cap Y \neq \emptyset \Rightarrow bsY \neq \emptyset \end{array}$$

H5:
$$\begin{array}{l} \forall b \in \mathcal{B}P \bullet \forall s \in traces(b) \bullet \forall X, Y \subseteq \Sigma \bullet \\ \quad bsX \cap Y^{\checkmark,\text{✗}} \neq \emptyset \wedge Y \subseteq X \Rightarrow bsY = bsX \cap Y^{\checkmark,\text{✗}}\ . \end{array}$$

Properties from these axioms are captured in

(C1)	$bsX = \emptyset$	$\wedge$	$Y \subseteq X$	$\Rightarrow$	$bsY = \emptyset$
(C2)	$bsX \cap Y \neq \emptyset$			$\Rightarrow$	$bsY \neq \emptyset$
(C3)	$bsX \cap Y^{\checkmark\text{✗}} \neq \emptyset$	$\wedge$	$Y \subseteq X$	$\Rightarrow$	$bsY = bsX \cap Y^{\checkmark\text{✗}}$,

for a typical trace s where X and $Y \subseteq \Sigma$. We will need the following in our definition of hiding later:

Definition 3.1 $\mathsf{behave}(b)$ *is an abbreviation for*

$$\begin{array}{c} b : \Sigma^* \nrightarrow \left(\mathbb{P}\,\Sigma \to \mathbb{P}\,\Sigma^{\checkmark\text{✗}}\right) \\ \wedge \\ \langle\rangle \in \operatorname{dom} b \\ \wedge \\ \forall s \in \operatorname{dom}(b) \bullet \forall X \subseteq \Sigma \bullet \ bsX \subseteq X^{\checkmark\text{✗}} \\ \wedge \\ \forall s \in \Sigma^* \bullet \forall x \in \Sigma \bullet \\ s^\frown\langle x\rangle \in \operatorname{dom}(b) \Leftrightarrow s \in \operatorname{dom}(b) \wedge (\exists X \subseteq \Sigma \bullet x \in bsX) \\ \wedge \\ \forall s \in \operatorname{dom} b \bullet \forall X, Y \subseteq \Sigma \bullet \\ \begin{array}{ccccc} bsX = \emptyset & \wedge\ Y \subseteq X & \Rightarrow & bsY = \emptyset & \wedge \\ bsX \cap Y \neq \emptyset & & \Rightarrow & bsY \neq \emptyset & \wedge \\ bs(X) \cap Y^{\checkmark\text{✗}} \neq \emptyset & \wedge\ Y \subseteq X & \Rightarrow & bsY = bsX \cap Y^{\checkmark\text{✗}} & . \end{array} \end{array}$$

4 Overtures and Hesitant Offers

Consider

$$\Big((a \to Stop) \overleftarrow{\Box} (b \to Stop)\Big) \setminus \{b\}$$

Suppose that it is offered $\{a\}$. Then we expect that a will be performed because it has higher priority. If an offer which does not include a is made, we expect the internal hidden b to happen, so the offer is refused: none of the offered external events occurs. So at first sight, one might expect $\Big((a \to Stop) \overleftarrow{\Box} (b \to Stop)\Big) \setminus \{b\} = a \to Stop$. In fact, there is another possibility: an offer of $\{a\}$ can be refused. In general

$$\Big((a \to Stop) \overleftarrow{\Box} (b \to Stop)\Big) \setminus \{b\} = (a \to Stop) \sqcap Stop \tag{2}$$

This shows that we need to take care in the definition of hiding.

The purpose of this section is to explain how the nondeterminism in hiding, not all of which is immediately obvious, arises. To do this we introduce *overtures* and *hesitant offers* which are tools built on the foundation of acceptances. It is particularly important to use precise instruments since $\mathcal{CSPP}$ explores new territory: hiding in the presence of priority.

To fix ideas, suppose that $P = (a \to Stop) \overleftarrow{\Box} (b \to Stop)$ is implemented as a synchronous state machine. Hiding b means that it is continuously available. Suppose this is implemented on an input wire *b_ready*: it will be continuously active. If a is offered to the circuit, this means that the corresponding *a_ready* is similarly active. If both signals are sampled as active when the circuit is first clocked, then a will happen and the circuit behaves as $a \to Stop$.

However, the *a_ready* signal will typically come from another circuit, and may not be ready on the first clock edge. On some later clock edge, the other circuit will be ready to communicate, and only then will *a_ready* become active. But meanwhile, our circuit will have performed the hidden event b on the first clock edge. And will have ceased activity since it now behaves as *Stop*. Effectively our circuit has been made the external offer $\emptyset$ before a later offer of $\{a\}$. We call a pair of offers of the form $\langle \emptyset, X \rangle$ a *hesitant offer*. So it is clear that P will behave as *Stop* when it receives an hesitant offer: $Stop \sqsupseteq P$. And equation 2 is explained.

In more general situations, especially when software rather than circuits are the target, 'tentative offers' can arise. Consider the case of two processes P and Q attempting to communicate. These might have conflicting priorities. A scheduler must resolve the conflict and determine which events can be agreed by the two processes. The scheduler may need to query each process to determine which events are acceptable in response to various offers. Thus offers may be made in order to determine the response, rather than to actually permit the event at this preliminary stage. Thus a process may be subjected to an *overture* of offers of the form $\langle X_1, \ldots, X_n \rangle$. Only the last offer X_n may actually result in an event being performed, even through earlier tentative offers may have signaled acceptance of at least one event. Such a scheduler strategy is not advocated here. We would expect practical language implementations to apply usage rules to exclude most if not all cases of priority conflict in the interests of efficiency. But $\mathcal{CSPP}$ is a closed algebra and so defines the outcome of such conflicts. And in the interests of abstraction, it must permit all possible implementations. Hiding must cover such situations.

The conclusion from the above is that an offer X should always be regarded as *the last element of an overture* $\langle X_1, \ldots, X \rangle$. But hiding is the only case in which this matters. For example, tentative offers made before a final offer to $a \to Stop$ make no difference. The response to X is always $X \cap \{a\}$. But not so for $\Big((a \to Stop) \overleftarrow{\Box} (b \to Stop)\Big) \setminus \{b\}$. The overture $\langle\{c,d\},\{e,f\},\{a\}\rangle$ results in a refusal, $\{a\} \rightsquigarrow \emptyset$, while $\langle\{a,d\},\{a,c\},\{a\}\rangle$ gives $\{a\} \rightsquigarrow \{a\}$. It is important here to realize that we are here using the ideas of acceptances in a dynamic way for understanding and to guide intuition. Our lack of knowledge of the particular overture leading up to an offer is modelled in static nondeterminism. Equation 2 just gives two ways of writing exactly the same process: neither has any trace of hidden state — that resides only in our intuitive understanding of the left hand side. But the dynamic intuition does explain *why* we equate the two expressions. This rather obvious point has occasionally been misunderstood.

Overtures seem to model all the sorts of implementation that arise, including asynchronous circuits. We use them to motivate the formal definition of hiding in $\mathcal{CSPP}$ below.

The reason that we pick out certain special overtures of the shape $\langle\emptyset, X\rangle$ and call them hesitant offers is that these are maximally permissive with respect to hidden events. Consider $P \setminus H$ and suppose that p is a behaviour of P. Suppose now that X is offered externally: the events of H are always available internally. So effectively, the internal offer is $X \cup H$. If some hidden $h \in H$ can be performed, that is $h \in pt(X \cup H)$ for some trace t of p, then the hidden trace t and event h can also occur when the offer is $\emptyset$. This is from **C3** which gives

$$pt(X \cup H) \cap H \neq \emptyset \;\Rightarrow\; ptH = pt(X \cup H) \cap H \;.$$

So hesitant offers uncover all the possible initial states. Thus we normally need only bother with the two special overtures $\langle X\rangle$ and $\langle\emptyset, X\rangle$ when deciding on the possible responses to the offer X.

5 Defining Hiding

In this section we build the general definition of hiding in $\mathcal{CSPP}$. A process is identified with a set of behaviours, and these behaviours must obey the axioms. So we will only admit functions q which obey the abbreviation $\mathsf{behave}(q)$. When we define $P \setminus H$, each behaviour $q \in \mathcal{B}\,(P \setminus H)$ arises from some matching behaviour p of P.

To define a behaviour q we need to specify what happens on one of its traces s. This trace is externally visible, and will match a hidden trace $t \in \mathit{traces}\,(p)$. s will just be t with any hidden events from H filtered out: $s = t \setminus H$. And there will be a most recent visible event of t.We capture the position in t where that occurs by an abbreviation:

Definition 5.1
$\mathsf{LastExposure}(t,H)$ *is an abbreviation for* $\max\left(\{n \in \operatorname{dom} t \mid t(n) \notin H\} \cup \{0\}\right)$.

So $\mathsf{LastExposure}(\langle h_1,h_2\rangle,\{h_1,h_2,h_3\}) = 0$ and $\mathsf{LastExposure}(\langle h_2,a,h_3\rangle,\{h_1,h_2,h_3\}) = 2$.

The next abbreviation identifies all the possibilities for hidden progress during overtures. The last visible event, if any, is the last element of s. This matches the position $\mathsf{LastExposure}(t,H)$ of t. We admit any t which involve further trailing events drawn from H under any conceivable overture. This is all accessible members of H except any eliminated by priorities within H itself. All such t give rise to the same s. We will see below that some extensions of t may also contribute to the acceptances at s, and for that reason we define a more general abbreviation which filters the 'tail' of t to permit only events available under an offer X:

Definition 5.2 *Let p be a behaviour and t a trace.* $\mathsf{Accessible}(p,t,X,H)$ *is an abbreviation for*

$$\begin{gathered}\forall\, 1 \le n \le \#t \bullet \exists\, Y \subseteq \Sigma \bullet t(n) \in p(t \downharpoonleft (n-1))(Y \cup H) \\ \wedge \\ \forall\, \mathsf{LastExposure}(t,H) \le n < \#t \bullet t(n) \in p\,(t \downharpoonleft (n-1))\,(X \cup H)\end{gathered} \qquad (3)$$

The notation $t \downharpoonright n$ is trace truncation: $t \downharpoonright 0 = \langle\rangle$ and $\langle t_1, \ldots, t_n\rangle \downharpoonright i = \langle t_1, \ldots, t_i\rangle$ at least when $i \leqslant n$. In the light of our earlier result from **C3**, setting $X = \emptyset$ above then identifies all the possible t matching s as above. These will be the possible 'starting points' for s: a shortest trace underlying s. When there are several possibilities they constitute nondeterminism. In the case of equation 2, take $s = \langle\rangle$. Then both $t = \langle\rangle$ and $t = \langle b\rangle$ are possible starting points. The two possibilities match the two components on the right hand side. That is $\mathsf{Accessible}(p, \langle\rangle, \emptyset, \{b\})$ and $\mathsf{Accessible}(p, \langle b\rangle, \emptyset, \{b\})$ are true for the only behaviour p of the hidden process on the right hand side.

The introduction noted that divergence can happen when an unbounded stream of events is hidden: external control of these events is lost, and the process can 'run wild':

Definition 5.3 $\mathsf{Wild}(p, t, X, H)$ *is an abbreviation for*

$$\exists\, w \in H^{\omega} \bullet \forall\, n > 0 \bullet (t^\frown(w \downharpoonright n)) \in \mathit{traces}\,(p) \wedge w(n) \in p(t^\frown(w \downharpoonright (n-1)))(X \cup H)\ , \tag{4}$$

where $n \in \mathbb{N}$ is understood.

w above is any infinite trace of events from H: $w = \langle h_1, h_2, h_1, h_2, \ldots\rangle$ perhaps when $H = \{h_1, h_2, h_3\}$. That provides a simple way of saying that there are arbitrary hidden extensions of the trace t when X is being offered. Notice that if a process goes into an internal loop during a tentative offer in an overture, it might be pulled back out of the loop when a later different offer is made. What matters is whether it is in an infinite loop under the final offer. Of course, in most situations, the external offer is irrelevant. But

$$\mu P \bullet \Big((a \rightarrow Stop) \overleftarrow{\Box} (b \rightarrow P)\Big) \setminus \{b\} = (a \rightarrow Stop) \overleftarrow{\Box}\ \mathsf{div}$$

is a process that can eventually be pulled out of a loop by an offer of a. Processes of this type are sometimes used in hardware, so the extra divergence information retained in $\mathcal{CSPP}$ compared to standard CSP is useful. It is also realistic since $\mathcal{CSPP}$ handles infinite behaviour properly, and divergence is inherently unbounded.

The next abbreviation is the crux of the definition of hiding: it specifies the responses

Definition 5.4 $\mathsf{Slide}(q, p, s, t, X, H)$ *is an abbreviation for*

$$qsX = \left(\begin{array}{c} \bigcup \left\{ (pt'(X \cup H)) \setminus H \;\middle|\; \begin{array}{c} t' \geqslant t \wedge t' \in \mathit{traces}\,(p) \wedge s = t' \setminus H \\ \wedge \\ \mathsf{Accessible}(p, t', X, H) \end{array} \right\} \\ \bigcup \\ (\{\text{✗} \end{array}\right) \tag{5}$$

✗ is the token representing livelock (divergence). Here t is the 'starting ' point for the trace s: as we have seen there are often several choices for that, and much of the nondeterminism in hiding arises in this way. But we will see that the response to an offer X can involve 'sliding' along trace extensions t'. The best way to understand that is with an example:

$$\Big((a \rightarrow Stop) \overleftarrow{\Box} (h \rightarrow b \rightarrow Stop)\Big) \setminus \{h\} = (a \rightarrow Stop) \overleftarrow{\Box} (b \rightarrow Stop) \quad \sqcap \quad b \rightarrow Stop$$

First of all the process might perform h internally before an effective offer is made: then it behaves like $b \rightarrow Stop$. That disposes of the second component on the right above. But suppose that an offer X is made before the hidden process has performed b. If $a \in X$, then

a occurs. However, if a is not available, then h happens: the processes ‘slides’ down an extended trace and then behaves like $b \rightarrow Stop$. It is this phenomena that Slide captures.

We are now almost equipped to define hiding in general, but there is one extra step. We need to make sure that we allocate ‘starting traces’ t to externally visible traces s in consistent way. We do that very simply with a function $\theta : traces\,(q) \rightarrow traces\,(p)$. A function ensures that there is a unique t for each s. We call this a ‘locater’ and it must be monotone:

Definition 5.5 locater(θ, S, H) *is an abbreviation for*

$$\forall\, s, s' \in S \bullet \begin{pmatrix} s > s' \;\Rightarrow\; \theta(s) > \theta(s') \\ \wedge \\ s = \theta(s) \setminus H \end{pmatrix}$$

Armed with these abbreviations, we can now give the definition which is rather simple and hopefully now transparent:

$$\mathcal{B}\,(P \setminus H) = \left\{ q \;\middle|\; \begin{array}{c} \mathsf{behave}(q) \;\wedge\; \exists\, p \in \mathcal{B}\,P \bullet \operatorname{dom} q \subseteq (\operatorname{dom} p) \setminus H \\ \wedge \\ \exists\,\theta : \operatorname{dom} q \rightarrow \operatorname{dom} p \bullet \mathsf{locater}(\theta, \operatorname{dom} q, H) \\ \wedge \\ \forall\, s \in traces\,(q) \bullet \\ \mathsf{Accessible}(p, \theta(s), \emptyset, H) \\ \wedge \\ \forall\, X \subseteq \Sigma \bullet \mathsf{Slide}(q, p, s, \theta(s), X, H) \end{array} \right\}. \tag{6}$$

Trying this on one of the simplest of compliant examples gives

$$\left((a \rightarrow Stop) \overset{\leftrightarrow}{\Box} (b \rightarrow Stop)\right) \setminus \{b\} = (a \rightarrow Stop) \;\;\sqcap\;\; Stop$$

We mentioned in section 2 that one way of embedding standard CSP in $\mathcal{CSPP}$ was to identify standard operators with the compliant versions. With that identification, we have reproduced the standard result $((a \rightarrow Stop) \Box (b \rightarrow Stop)) \setminus \{b\} = (a \rightarrow Stop) \;\;\sqcap\;\; Stop.$

In the more general embedding of standard CSP, we have

$$\begin{aligned}
((a \rightarrow Stop) \Box (b \rightarrow Stop)) \setminus \{b\} \;\; &= \begin{pmatrix} (a \rightarrow Stop) \overset{\leftarrow}{\Box} (b \rightarrow Stop) \\ \sqcap \\ (a \rightarrow Stop) \overset{\leftrightarrow}{\Box} (b \rightarrow Stop) \\ \sqcap \\ (a \rightarrow Stop) \overset{\rightarrow}{\Box} (b \rightarrow Stop) \end{pmatrix} \setminus \{b\} \\
&= \begin{pmatrix} \left((a \rightarrow Stop) \overset{\leftarrow}{\Box} (b \rightarrow Stop)\right) \setminus \{b\} \\ \sqcap \\ \left((a \rightarrow Stop) \overset{\leftrightarrow}{\Box} (b \rightarrow Stop)\right) \setminus \{b\} \\ \sqcap \\ \left((a \rightarrow Stop) \overset{\rightarrow}{\Box} (b \rightarrow Stop)\right) \setminus \{b\} \end{pmatrix} \\
&= \begin{pmatrix} (a \rightarrow Stop) \sqcap Stop \\ \sqcap \\ (a \rightarrow Stop) \sqcap Stop \\ \sqcap \\ Stop \end{pmatrix} \\
&= (a \rightarrow Stop) \sqcap Stop
\end{aligned}$$

so we reproduce the standard result once again. Revisiting the more general case in equation 1 on page 104, we have

$$\begin{aligned}
\Big((a \to P) \overset{\leftarrow}{\Box} (b \to Q)\Big) \setminus \{b\} &= (a \to P') \overset{\leftarrow}{\Box} Q' \;\sqcap\; Q' \\
\Big((a \to P) \overset{\leftrightarrow}{\Box} (b \to Q)\Big) \setminus \{b\} &= (a \to P') \overset{\leftrightarrow}{\Box} Q' \;\sqcap\; Q' \\
\Big((a \to P) \overset{\rightarrow}{\Box} (b \to Q)\Big) \setminus \{b\} &= Q'
\end{aligned}$$

When we identify standard CSP with the compliant subset, the second equation above is identified with equation 1. But in the more general embedding, we have

$$\begin{aligned}
((a \to Stop) \Box (b \to Stop)) \setminus \{b\} &= (a \to P') \overset{\leftarrow}{\Box} Q' \;\sqcap\; (a \to P') \overset{\leftrightarrow}{\Box} Q' \;\sqcap\; Q' \\
&\sqsupseteq ((a \to P') \Box Q') \;\sqcap\; Q'
\end{aligned}$$

That is we have a *refinement* of the standard result. This is entirely to be expected: we can draw finer distinctions in $\mathcal{CSPP}$ than we can in CSP.

A more interesting case of hiding a compliant process is

$$\Big((a \to Stop) \overset{\leftrightarrow}{\Box} (b \to c \to Stop)\Big) \setminus \{b\} = (a \to Stop) \overset{\leftrightarrow}{\Box} (c \to Stop) \;\sqcap\; c \to Stop$$

6 Hiding and Recursion

A proper semantics for any variant of CSP must define recursion properly. In the case of a denotational semantics, it lies at the heart of the theory. Recursion is defined by some sort of fixed point theory. This in turn is initimately connected with infinite behaviour, and acceptance semantics handles this in a novel and powerful way.

CSP and its variants have a much wider scope than just finite state machines that can be implemented in software and hardware. It is a much more powerful mathematical tool. It is typical of such mathematical theories that they can describe some rather exotic beasts. Thus there are some processes, usually involving infinite behaviour in the form of unbounded nondeterminism, which have been awkward to handle in previous semantics. This is not a defect of CSP, but a sign of its power. The acceptance semantics of $\mathcal{CSPP}$ completely solves all these problems in a simple way.

There are two complementary ways of locating fixed points in the acceptance approach to $\mathcal{CSPP}$:

1. $\mathcal{CSPP}$ is a complete lattice under refinement.

2. $\mathcal{CSPP}$ is a complete pseudometric space.

The approach based on refinement uses the classic Knaster-Tarski fixed point theorem which shows that any monotone function f on a complete lattice has a least fixed point. It is given by $\mu f = \bigwedge\{x \mid x \sqsupseteq f(x)\}$ which we would write here as $\mu f = \bigsqcap\{x \mid x \sqsupseteq f(x)\}$. If f is also *continuous* with respect to the partial order, there is a more constructive form for the fixed point, but we will omit that here.

Refinement is very simple in acceptance semantics: it is just set inclusion in the behaviours: $P \sqsupseteq Q \;\Leftrightarrow\; \mathcal{B}P \subseteq \mathcal{B}Q$. That in turn means means that hiding is monotone: $P \sqsupseteq Q \;\Rightarrow\; (P \setminus H) \sqsupseteq (Q \setminus H)$ which follows by inspection of equation 6. Since all the other $\mathcal{CSPP}$ operations are also monotone, we find that recursions involving hiding are meaningful: they represent the least fixed point. A rather extreme example is $\mu P \bullet P \setminus H = \bot \setminus H$, where $\perp$ is the most nondeterministic of all processes. Another example which will be of interest below is $P = a \to P \setminus \{a\}$. It is easy to see that the most nondeterministic solution is $a \to \perp \setminus \{a\}$. Other examples appeared in the last section.

The complementary way to define fixed points is to use a metric, or in the case of $\mathcal{CSPP}$, a pseudometric d. A *pseudo*metric because there are certain distinct processes are not distinguished by d. When functions are contracting with respect to d, then a fixed point exists. Matching recursions are called *constructive*. Because d is a pseudometric, there are cases where the fixed points are not unique, but those are well defined, and one can always identify the cluster of 'matching' processes. In such cases, we just take the most nondeterministic such process as the meaning of the recursion which matches the least fixed point with respect to refinement. This is an application of Janes maximum entropy principle, so we would have made this choice even in the absence of a refinement order. Most applications of recursion use constructive functions, and most $\mathcal{CSPP}$ operators are contracting. However, hiding is an exception. Our earlier example of $P = a \to P \setminus \{a\}$ is a guarded recursion, yet is not contracting. This is a standard phenomena and indeed a standard example, but we mention it here for completeness. This does not necessarily mean that the presence of hiding in a recursion prevents it from contracting, but it does mean that each recursion involving hiding must be examined individually.

7 Conclusions

Hiding has been shown to be a fundamental part of CSP and so also of $\mathcal{CSPP}$. Indeed it dictates the presence of $\sqcap$ and div, and accounts for much of the flavour of the theory.

The nondeterminism that is asociated with hiding has been carefully investigated and explained, and the ideas of *hesitant offer* and *overture* formulated as precise tools for the investigation.

Hiding has been extended to handle priority properly, and the formal definition in $\mathcal{CSPP}$ acceptance semantics presented with careful justification and explanation, all with a full treatment of unbounded behaviour.

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] Davide Sangiorgi and David Walker. *The π-calculus*. Cambridge University Press, 2001.

[4] Robin Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[5] P.H.Welch. Process Oriented Design for Java: Concurrency for All. In H.R.Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, volume 1, pages 51–57. CSREA, CSREA Press, June 2000.

[6] P.H.Welch, J.R.Aldous, and J.Foster. CSP networking for java (JCSP.net). In P.M.A.Sloot, C.J.K.Tan, J.J.Dongarra, and A.G.Hoekstra, editors, *Computational Science - ICCS 2002*, volume 2330 of *Lecture Notes in Computer Science*, pages 695–708. Springer-Verlag, April 2002.

[7] Peter H. Welch and Jeremy M.R. Martin. Formal analysis of concurrent java systems. In *Communicating Process Architectures – 2000*, Concurrent Systems Engineering, pages 275–301, Amsterdam, Sept 2000. IOS Press.

[8] A.E. Lawrence. Extending CSP: Denotational semantics. *IEE Proceedings: Software,* 150(1):51–60, Feb 2003.

[9] 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.

[10] A.E. Lawrence. CSP extended: imperative state and true concurrency. *IEE Proceedings: Software*, 150(1):61–69, Feb 2003.

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