Communicating Process Architectures – 2002
James Pascoe, Peter Welch, Roger Loader and Vaidy Sunderam (Eds.)
IOS Press, 2002

Semantics of `prialt` in Handel-C™

Andrew BUTTERFIELD[1] and Jim WOODCOCK[2]
[1] *Dublin University*
[2] *University of Kent at Canterbury*
Andrew.Butterfield@cs.tcd.ie

Abstract. This paper discusses the semantics of the `prialt` construct in Handel-C[1]. The language is essentially a static subset of C, augmented with a parallel construct and channel communication, as found in CSP. All assignments and channel communication events take one clock cycle, with all updates synchronised with the clock edge marking the cycle end. The behaviour of `prialt` in Handel-C is similar in concept to that of occam [2, 3], and to the *p-priority* concept of Adrian Lawrence $\mathcal{CSPP}$[4]. However, we have to contend with both input and output guards in Handel-C, unlike the situation in occam, although `prialts` with conflicting priority requirements are not legal in Handel-C. This makes our problem simpler than the more general case including such conflicts considered in [4]. We start with an informal discussion of the issues that arise when considering the semantics of Handel-C's `prialt` construct. We define a resolution function ($\mathcal{R}$) that determines which requests in a collection of `prialts` become active. We describe a few properties that we expect to hold for resolution, and discuss the issue of compositionality.

1 Introduction

This paper discusses the semantics of the `prialt` construct in Handel-C[1], a language originally developed by the Hardware Compilation Group at Oxford University Computing Laboratory. It is a hybrid of CSP [5] and C, designed to target hardware implementations, specifically field-programmable gate arrays (FPGAs) [6, 7, 8, 9]. The language is essentially a static subset of C, augmented with a parallel construct and channel communication, as found in CSP. The type system has been modified to refer explicitly to the number of bits required to implement any given type. The language targets largely synchronous hardware with a multiple clock domains. All assignments and channel communication events take one clock cycle, with all updates synchronised with the clock edge marking the cycle end. All expression and conditional evaluations are deemed to be instantaneous, effectively being completed before the current clock-cycle ends.

1.1 Notation

In Handel-C, a typical program fragment showing two `prialts` in parallel might appear as:

```
par {
  prialt {
    a!11 : P1 ; break ;
    b?x  : P2 ; break ;
  };
  prialt {
    b!22 : P3 ; break ;
    c?y  : P4 ; break ;
  }
}
```

Here, `P1`, `P2`, `P3`, and `P4` denote the continuation code that gets executed if the corresponding guard is deemed to be active. In $\mathcal{CSPP}$ [4], this could be written as:

$$((a!11 \rightarrow P_1) \overleftarrow{\Box} (b?x \rightarrow P_2(x))) \parallel ((b!22 \rightarrow P_3) \overleftarrow{\Box} (c?y \rightarrow P_4(y)))$$

We adopt a simpler notation, in which we ignore the guard values, variables and continuation processes, viewing each `prialt` simply as a sequence of guards, and those `prialts` in parallel simply being collected into a set, to give:

$$\left\{ \begin{array}{c} \langle a!, b? \rangle \\ \langle b!, c? \rangle \end{array} \right\}$$

We do this because the concern here is how the decision is made regarding *which* guards are deemed to be true. We are not interested in this paper in the consequent actions, both of the guard communications and the subsequent process actions.

1.2 The Problem

The behaviour of `prialt` in Handel-C is similar in concept to that of occam [2, 3], and to the *p-priority* concept of Adrian Lawrence $\mathcal{CSPP}$[4]. However, we have to contend with both input and output guards in Handel-C, unlike the situation in occam. In relation to *p-priority*, Handel-C does not admit processes with conflicting priorities, so presents a simpler problem semantically, than that found in $\mathcal{CSPP}$. The synchronous nature of Handel-C also makes it easy to establish what `prialt` statements are participating in a synchronisation event at any given point in time. A particular consequence of this is that all the key events are deemed to occur simultaneously.

Our interest is in determining the outcome when a number of `prialts` are simultaneously ready to run, i.e the outcome of the following situation:

$$\left\{ \begin{array}{c} \langle g_{11}, g_{12}, \ldots, g_{1j}, \ldots, g_{1m_1} \rangle \\ \langle g_{21}, g_{22}, \ldots, g_{2j}, \ldots, g_{2m_2} \rangle \\ \vdots \\ \langle g_{i1}, g_{i2}, \ldots, g_{ij}, \ldots, g_{im_i} \rangle \\ \vdots \\ \langle g_{n1}, g_{n2}, \ldots, g_{nj}, \ldots, g_{nm_n} \rangle \end{array} \right\}$$

The guards can have only one of the following two forms: input guard $c?$ or output guard $c!$. Handel-C allows the use of a `default` guard as the last guard in a `prialt`, but again this is something which we will ignore for current purposes. Another constraint present in Handel-C is that a channel may appear only once in any given `prialt`. Handel-C also admits communication statements outside of `prialts` ("naked communication"). However these can be modelled by a singleton `prialt` with a null continuation process:

```
c!e   ≡   prialt{c!e:break}
```

In $\mathcal{CSPP}$, semantics is given using *Acceptances* which are partial functions from traces to relations between sets of of offered and accepted events, where the domain of a partial function equals the set of traces of the process it denotes. A process is therefore described by describing how, after any of its traces (s), with an environment offering a set of events (E), it is willing to accept a specific set (A) of events. The notation used for this is

$$s : E \rightsquigarrow A$$

However, in Handel-C, it seems to make more sense to view the `prialts` as making specific offers of *sequences* of events, rather than sets of events. We can view the `prialt` statement: $\langle g_1, g_2, \ldots, g_n \rangle$ as offering the events $g_1, g_2, \ldots, g_n$ in that order. In effect what we have done here is to abstract the `prialt` notation to reflect precisely the prioritised list of events that it is willing to offer/accept.

A general resolution mechanism then determines which events actually occur. This paper is a description of the formal semantics of that resolution mechanism, in a form sufficiently general that it can be incorporated into a variety of different semantics for Handel-C—denotational [10], operational [11] or otherwise. As `prialt` resolution in Handel-C is deterministic, having the form of a total function from sets of `prialts` to their resolution, we adopt a formal notation most suited to describing functions, that of the "Irish School" of the VDM (*VDM*♣) [12]. This can be described briefly as a functional subset of standard VDM (VDM-SL [13]), using equational reasoning rather than the logic of partial functions. A notation guide is provided as an appendix, as well as by explanations running through the text.

Despite the Handel-C language's superficial similarity to CSP, we did not adopt CSP as a semantic modelling language here, because it lacks any notion of priority. Also, it was not clear when the work started if it is possible to capture this at all in CSP. Clear evidence for this is the work of Adrian Lawrence [14, 4, 15], which requires extensions to CSP in order to capture the notion of priority, and simultaneous events.

2 Informal Description

We start with an informal discussion of the issues that arising when considering the semantics of Handel-C's `prialt` construct.

We consider the situation where we have n `prialt` statements which are to be resolved, and we assume, as already stated, a simplified model where `prialts` consist solely of the communication statements, hereinafter called requests. Each `prialt` is therefore a non-empty sequence of requests. We assume all `prialts` are well-formed, in that any given channel identifier occurs at most once in any given `prialt`.

2.1 Priority: Absolute or Relative ?

Initially, with Handel-C, it was assumed that a `prialt` of the form

$$\langle g_1, g_2, \cdots, g_m \rangle$$

assigned priority levels $1, 2, \ldots m$ to guards $g_1, g_2, \ldots g_m$ respectively. However all attempts to reconcile this *absolute priority* view with Handel-C's actual behaviour resulted in failure.

After experimenting with `prialts` in Handel-C it became very clear that the notion of absolute priority was completely incorrect. Instead, each `prialt` simply defines a *relative priority* for the channels to which it refers. This is a strong indicator that *p-priority*, rather than *e-priority* [14], is the correct way forward. Merging the (locally total) ordering for each `prialt` results in either an overall partial order, or a relation with cycles. In the former case we use the ordering to determine the successful requests. In the latter case Handel-C reports an error. The simplest example of this is

$$\{ \langle a!, b! \rangle, \langle b?, a? \rangle \}$$

In [4], provision is made to give a meaning to this expression, but in Handel-C, which is deterministic, this is an un-recoverable compile-time error. If we try to resolve this in the

Handel-C semantics by nondeterministically choosing to “break” cycles, we end up losing some nice laws regarding the sequencing in `prialts` and the way in which an offered event masks all subsequent events in a given `prialt`:

$$\langle g_1, \ldots, g_i, \ldots g_m \rangle = \langle g_1, \ldots, g_i \rangle \text{ when } g_i \text{ is “offered”}$$

This law does not hold if we resolve priority conflicts by breaking priority cycles.

2.2 Deriving Channel Partial Ordering

Given `prialts`, we first take each constituent `prialt` and convert it into the corresponding relation (total linear ordering). We then merge all the relations together, one from each `prialt`, using relational union, and take the transitive closure to get one big relation (ρ). This relation captures the priorities between all participating channels.

We then examine the (graph of) the relation for cycles. If present, we should flag an error condition at this point—the processes involved all diverge. If there are no cycles, then we have a partial ordering on all the channels which can then be used as the basis for determining the successful requests. Also this relation will in general have many minimal elements.

At this stage we have all the `prialts`, and an induced partial ordering on the channel identifiers. We identify active channels (offered events) as being those which appear at least twice in (different) `prialts`, with complementary directions.

2.3 Pathological Cases

Note that a channel may occur multiple times, in multiple `prialts`. This is satisfactory as long as all but two complementary requests for that channel are effectively masked by other channels with requests at higher priority. An example is:

$$\{ \langle a! \rangle, \langle a? \rangle, \langle b!, a? \rangle, \langle b?, a! \rangle \}$$

which is equivalent to:

$$\{ \langle a! \rangle, \langle a? \rangle, \langle b! \rangle, \langle b? \rangle \}.$$

In the event that we have one writer to a channel and multiple receivers, as in

$$\{ \langle \ldots, c!, \ldots \rangle, \langle \ldots, c?, \ldots \rangle, \langle \ldots, c?, \ldots \rangle, \ldots \}$$

the outcome (assuming all these channel instances are deemed active) is that each reader gets their own copy of the single written value.

What happens in the case of multiple writers ?

$$\{ \langle \ldots, c?, \ldots \rangle, \langle \ldots, c!, \ldots \rangle, \langle \ldots, c!, \ldots \rangle, \ldots \}$$

In Handel-C Version 2.1 [16], the compiler flagged this with a warning, and in simulation, the value obtained from the channel was the bitwise logical-OR of the underlying representations of all the inputs ! It is clear that a multiplexor (which ends with a logical OR gate) is getting multiple input streams switched through simultaneously to the output. This is bizarre, but at least the compiler (i) informed the user and (ii) maintained the order independence of parallel composition of `prialts`:

$$P_1 \parallel P_2 \equiv P_2 \parallel P_1$$

However, in Handel-C Version 3.0 [1], the compiler supplies no warning at all, and proceeds to accept data for the channel write occurring in the `prialt` which is first in the program text ! This means that

$$\{ \langle c?x \rangle, \langle c!11 \rangle, \langle c!22 \rangle \} \not\equiv \{ \langle c?x \rangle, \langle c!22 \rangle, \langle c!11 \rangle \}$$

The first writes 11 to `x`, the second writes 22 ! To handle this properly, we would have to talk about sequences of `prialts` being resolved, rather than sets. Also we would have to sacrifice the following very nice property of Handel-C (which was true for 2.1, but is now false for 3.0):

$$P_1 \parallel P_2 = P_2 \parallel P_1$$

We shall not concern ourselves overly with this case, but rather consider such programs at present to be ill-formed, with no obligation on our part to supply a formal semantics for them. They should certainly not be the result of any refinement methodology for calculating Handel-C programs from specifications, which is the ultimate goal of this work.

2.4 Identifying Active Channels

Once we have identified the minimal active channels and removed them and the `prialts` in which they appear, we repeat the entire process of constructing the ordering afresh, and selecting the minimal active channels, with the remaining `prialts`. We continually repeat this until no further changes occur.

3 Formal Treatment

We are defining a resolution function ($\mathcal{R}$) that determines which requests in a collection of `prialts` become active.

3.1 Input Types

We start by considering as given a collection of channel identifiers:

$$c \in Ch$$

With every I/O request in Handel-C there is an associated direction:

$$d \in Dir = \{\, \textsc{In}, \textsc{Out} \,\}$$

This allows us to define an input-output request as a channel/direction pair: Request:

$$r \in Req = Ch \times Dir$$

These "requests" are in fact our guards, as all such guards in Handel-C are of this form. In general we shall write $c!$ and $c?$ as shorthands for $(c, \textsc{Out})$ and $(c, \textsc{In})$, respectively.

Having abstracted away from the details of the processes that are guarded in `prialts`, we view them as sequences of requests, where each channel occurs at most once:

$$\begin{array}{rcl} \rho \in PriAlt &=& Req^{+} \\ \textit{inv-}PriAlt(\sigma) &\widehat{=}& (\# \circ \mathtt{elems} \circ {\pi_1}^{\star})\sigma = \mathtt{len}\,\sigma \end{array}$$

The notation *inv-*$PriAlt$ is the *VDM*♣ notion of "invariant", which gives a boolean expression defining which elements of the space are considered well-formed. which is analagous to the "healthiness conditions" of CSP/$\mathcal{CSPP}$.

What we are resolving is simply a set of `prialts`:

$$P \in PriGrp \;=\; \mathbb{P} PriAlt$$

This structure is the input to our resolution process.

3.2 Output Types

We now consider the information that needs to be output from the resolution process. Firstly, we need to know which channels are going to be active, i.e. able to communicate. We need to know the `prialts` with which these channels are associated, so we can identify the specific guards involved, and consequently which guarded processes will be involved. We cannot determine this information from knowledge of which channels are active alone, because a given channel can be active in a proper subset of those `prialts` in which it is mentioned. To see this, consider the following `prialt` collection, including guarded processes, (ignoring directions, with `prialts` labelled for ease of reference):

$$\left\{ \begin{array}{l} 1 : \langle a \to P_1,\ b \to P_2 \rangle \\ 2 : \langle a \to P_3 \rangle \\ 3 : b \to P_4 \\ 4 : b \to P_5 \end{array} \right\}$$

Here, both events a and b will proceed. Event a will involve `prialts` 1 and 2, leading to the subsequent execution of P_1 and P_3. Event b will involve `prialts` 3 and 4, leading to P_4 and P_5 being activated subsequently. However, even though `prialt` 1 mentions b, and b takes place, it is masked by the higher priority a event in that `prialt`, which is also active. Hence we need to couple a channel (event) identifier with the `prialts` which are actually involved.

Our key output structure is therefore a map from channels to `prialt`-sets:

$$\gamma \in \mathit{CPMap} = \mathit{Ch} \to \mathit{PriGrp}$$

In the example above, we expect to see an output of

$$\left\{ \begin{array}{l} \{a \mapsto \{\, \langle a \to P_1,\ b \to P_2 \rangle, \langle a \to P_3 \rangle \,\}\} \\ \{b \mapsto \{\, \langle b \to P_4 \rangle, \langle b \to P_5 \rangle \,\}\} \end{array} \right\}$$

We can then determine the active processes by using the channel value as a lookup key in the associated set of `prialts`. So, for

$$\{a \mapsto \{\, \langle a \to P_1,\ b \to P_2 \rangle, \langle a \to P_3 \rangle \,\}\}$$

we perform a lookup of a in both $\langle a \to P_1,\ b \to P_2 \rangle$ and $\langle a \to P_3 \rangle$ to get P_1 and P_3 respectively.

Of course, in this paper we are ignoring the guards processes P_i mentioned in this example, but at least this example shows in principle how that information can be incorporated.

A key aspect highlighted by this example is a well-formedness constraint on channel-`prialt`-set pairs. We expect the channel component to be present in every `prialt` in the corresponding set:

$$\begin{array}{rcl} \mathit{inv\text{-}CPMap} & : & \mathit{CPMap} \to \mathbb{B} \\ \mathit{inv\text{-}CPMap}\ \gamma & \widehat{=} & \forall c \in \mathtt{dom}\,\gamma \bullet \mathit{inv\text{-}CPSet}_\gamma(c) \\ \mathit{inv\text{-}CPSet}_\gamma(c) & \widehat{=} & \forall \rho \in \gamma(c) \bullet c \in (\mathtt{elems} \circ {\pi_1}^\star)\rho \end{array}$$

3.3 Intermediate Types

In order to describe the resolution process, it will be necessary to have some key intermediate structures. Firstly, we note that the input to the resolution process is very `prialt`-centric, but that it will be helpful to have a structure that is more channel-centric.

Each channel present occurs at most once in any given `prialt`, in association with a corresponding direction. With any given channel, over all the `prialts` in which it appears, we would like to associate the directions of the corresponding requests. We therefore map each channel to a set of the request directions associated with it:

$$\kappa \in \mathit{ChReqs} = \mathit{Ch} \to \mathbb{P}\mathit{Dir}$$

As described above, we need build a relation over channels. We choose to model relations here as set-valued functions, as is conventional in *VDM*♣:

$$\mathcal{R} \in \mathit{ChRel} = \mathit{Ch} \to \mathbb{P}\mathit{Ch}$$

This formulation, while not very conventional outside *VDM*♣, does make it very easy to extract the minimal elements, which denote the channels of highest priority.

3.4 Intermediate Operators

We now look at some operators for our intermediate semantic domains. We start with a function that converts the input of `prialt` sets into a channel request mapping:

$$\begin{array}{rcl} \text{toChReqs} & : & \mathit{PriGrp} \to \mathit{ChReqs} \\ \text{toChReqs}\ P & \widehat{=} & (\circledcup / \circ \mathbb{P}(\circledcup / \circ \imath^\star))P \\ \textbf{where} & & \\ \imath(c, d) & \widehat{=} & \{c \mapsto \{\, d \,\}\} \end{array}$$

Basically every entry ρ in P of the form

$$\langle (c_1, d_1), (c_2, d_2), \ldots, (c_k, d_k) \rangle$$

is converted to an entry of the form

$$\{c_1 \mapsto \{\, d_1 \,\},\ c_2 \mapsto \{\, d_2 \,\},\ \ldots,\ c_k \mapsto \{\, d_k \,\}\}$$

All of these (one for each $\rho \in P$) are then joined together using $\circledcup$ to provide one relation. The $\circledcup$ operator is a lifted (indexed) form of union, and has the effect of acting as relational union. In particular, we get

$$\{c \mapsto \{\, d \,\}\} \circledcup \{c \mapsto \{\, d' \,\}\} = \{c \mapsto \{\, d, d' \,\}\}$$

Once we have this channel request mapping, it becomes very easy to see if any channel is active. We simply look it up, and deduce that the channel is active if both directions are present:

$$\begin{array}{rcl} \text{Act} & : & \mathit{ChReqs} \to \mathit{Ch} \to \mathbb{B} \\ \text{Act}_\kappa c & \widehat{=} & \kappa(c) = \{\, \textsc{In}, \textsc{Out} \,\} \end{array}$$

The central intermediate structure is that which captures the partial ordering on channel priorities induced by the collection of `prialts`. Consider the following input structure:

$$P = \left\{ \begin{array}{c} \langle\ (c_{11}, d_{11}),\ \ldots,\ (c_{1j}, d_{1j}),\ \ldots,\ (c_{1k_1}, d_{1k_1})\ \rangle \\ \vdots \\ \langle\ (c_{i1}, d_{i1}),\ \ldots,\ (c_{ij}, d_{ij}),\ \ldots,\ (c_{ik_1}, d_{ik_i})\ \rangle \\ \vdots \\ \langle\ (c_{n1}, d_{n1}),\ \ldots,\ (c_{nj}, d_{nj}),\ \ldots,\ (c_{nk_1}, d_{nk_n})\ \rangle \end{array} \right\}$$

For each sequence in P, we then discard the direction data ($\pi_1{}^\star$), and then convert the resulting sequence into the corresponding total ordering relation (using `order`), modelled as a set valued mapping, but shown symbolically here:

$$S' = \left\{ \begin{array}{c} \{\ c_{11} \prec c_{12} \prec \cdots \prec c_{1j} \prec c_{1k_1}\ \} \\ \vdots \\ \{\ c_{i1} \prec c_{i2} \prec \cdots \prec c_{ij} \prec c_{1k_i}\ \} \\ \vdots \\ \{\ c_{n1} \prec c_{n2} \prec \cdots \prec c_{nj} \prec c_{nk_n}\ \} \end{array} \right\}$$

We then reduce this set with relational union ($\circledcirc$) to form a single relation, and take its transitive closure ($\mathtt{TC}$)[1]. All of this is combined into the following operator:

$$\begin{array}{rcl} \text{toChRel} & : & \mathit{PriGrp} \rightarrow \mathit{ChRel} \\ \text{toChRel} & \widehat{=} & \mathtt{TC} \circ \circledcirc / \circ \mathbb{P}(\mathtt{order} \circ \pi_1{}^\star) \end{array}$$

We can define `order` as follows:

$$\begin{array}{rcl} \mathtt{order} & : & C^\star \rightarrow (C \rightarrow \mathbb{P}C) \\ \mathtt{order}\ \Lambda & \widehat{=} & \theta \\ \mathtt{order}\ \langle c \rangle & \widehat{=} & \{c \mapsto \emptyset\} \\ \mathtt{order}\ (c : c' : \sigma) & \widehat{=} & \{c \mapsto \{\ c'\ \}\} \circledcirc \mathtt{order}\ (c' : \sigma) \end{array}$$

3.5 Input-Output Relation

We now describe the overall relationship between the inputs and outputs of the `prialt` resolution process ($\mathcal{R}$). The input is a set of `prialts`, while the output is a pair (*Resltn*) consisting of a map from all active channels to the set of associated `prialts`, and the set of `prialts` which are found to be inactive:

$$\begin{array}{rcl} (\gamma, P) \in \mathit{Resltn} & = & \mathit{CPMap} \times \mathit{PriGrp} \\ \mathcal{R} & : & \mathit{PriGrp} \rightarrow \mathit{Resltn} \end{array}$$

As a precondition, we insist that the underlying relation on channels be acyclic (i.e. a partial order):

$$\textit{pre-}\mathcal{R}\ \rho \mathrel{\widehat{=}} (\neg \circ \mathtt{acyclic} \circ \text{toChRel})P$$

We supply a partial postcondition which captures certain key properties we expect of the output. Let us assume that applying $\mathcal{R}$ to P results in output (γ, B), i.e

$$\mathcal{R}(P) = (\gamma, B)$$

[1] It is quite likely that the transitive closure is not necessary for this to work

First consider γ, which is a map from of channel to `prialt`-set. All the `prialts` mentioned in γ, given by

$$A = ({}^{\cup}/ \circ \mathtt{rng})\gamma$$

are active `prialts`. The collection of inactive `prialts` (B) should consist precisely of those not mentioned in A:

$$B = \triangleleft[A]P.$$

Secondly, consider the set of all actually active request granted) channels (G):

$$G = \mathtt{dom}\,\gamma$$

We expect this to be a subset of the potentially active channels (Q). These are determined by taking the channel-centric view of the input:

$$\kappa = \text{toChReqs}\ P$$

taking its domain, and then filtering this with the active-channel predicate, itself parameterised by κ:

$$Q = (\mathtt{filter}[\text{Act}_\kappa] \circ \mathtt{dom})\kappa$$

We expect $A \subseteq Q$. This is all collected together in the postcondition:

$$\begin{array}{rcl} \textit{post-}\mathcal{R}(P) \mapsto (\gamma, B) & \Rightarrow & \mathtt{dom}\,\gamma \subseteq (\mathtt{filter}[\text{Act}\kappa] \circ \mathtt{dom})\kappa \ \wedge\ B = \triangleleft[A]P \\ & \textbf{where} & \\ & \kappa \ \widehat{=} & \text{toChReqs}\ P \\ & A \ \widehat{=} & ({}^{\cup}/ \circ \mathtt{rng})\gamma \end{array}$$

Observe that (θ, P) satisfies the above post-condition, and the post-condition does not fully define the resolution process, but rather serves to specify some well-formedness criteria for the outcome. In particular, no mention is made of priority.

We can define an invariant on the output tuple (γ, B) which states that the `prialts` occurring in γ do not appear in B:

$$\begin{array}{rcl} \textit{inv-Resltn} & : & \textit{Resltn} \to \mathbb{B} \\ \textit{inv-Resltn}(\gamma, B) & \widehat{=} & ({}^{\cup}/ \circ \mathtt{rng})\gamma \cap B = \emptyset \end{array}$$

3.6 Input-Output Function

We now look at the actual (functional) relationship between the input and outputs for resolution. We shall view resolution as repeatedly applying a resolution step ($\mathcal{RS}$) to an initial "output value" (θ, P) until no change occurs:

$$\mathcal{R}(P) = \mathtt{iterate}\ (=)\ \mathcal{RS}\ (\theta, P)$$

Note that $\mathcal{RS}$ preserves the invariant *inv-Resltn*. We observe that $\mathcal{RS}$ repeatedly transforms the input as follows:

$$(\theta, P) \to (\gamma_1, P_1) \to (\gamma_2, P_2) \to \cdots \to (\gamma_i, P_i) \to (\gamma_n, P_n)$$

The iteration stops when we reach (γ_n, P_n) such that $\mathcal{RS}(\gamma_n, P_n) = (\gamma_n, P_n)$. If we take $(\gamma_0, P_0) = (\emptyset, P)$, we can state that $\mathcal{RS}$ has the following properties:

1. If $\mathcal{RS}$ changes the input, then γ is larger and P is smaller:

$$\mathcal{RS}(\gamma_i, P_i) = (\gamma_{i+1}, P_{i+1}) \Rightarrow \gamma_i \subset \gamma_{i+1} \wedge P_i \supset P_{i+1}$$

2. If $\mathcal{RS}$ does not change its input, then P contains no active channels, and *v.v.*:

$$\mathcal{RS}(\gamma_n, P_n) = (\gamma_n, P_n) \equiv \{\, \textsc{In}, \textsc{Out} \,\} \notin \texttt{rng}(\text{toChReqs } P_n)$$

The resolution step function applied to a tuple (γ, P) identifies active `prialt` channels of highest priority in P and removes them, putting the corresponding channel and `prialt` information into γ:

$$\mathcal{RS}(\gamma, P) = (\gamma', P')$$

The key feature is how γ' and P' are computed. First from P we obtain the partial order (ρ) and channel-centric view (κ):

$$\rho = \text{toChRel } P \quad \textbf{and} \quad \kappa = \text{toChReqs } P$$

We then identify all those channels that are active in ρ and then filter them to retain only those which are minimal (A'):

$$A' = \texttt{mins}(\texttt{filter}[\text{Act}_\kappa]\rho)$$

Given our representation of the partial order relation by $Ch \to \mathbb{P}Ch$ (rather than the more traditional $\mathbb{P}(Ch \times Ch)$) allows us to give a concise definition of `mins`:

$$\texttt{mins } \rho \mathrel{\widehat{=}} \mathrel{\ntriangleleft}[({}^{\cup}/ \circ \texttt{rng})\rho](\texttt{dom } \rho)$$

We simply take the domain, and remove from it anything that occurs in any of the range sets, thus leaving every channel which points to (lower priority) channels, but is not itself so pointed to.

For every channel in A', we extract and map them to the set of all the `prialts` in which they appear:

$$(\gamma', P') = \text{extr } \gamma \ P \ A'$$

Here "extr" scans through channels in A', using them to identify the relevant `prialts` in P which are then accumulated together with their channel in γ:

$$\begin{array}{rcl}
\text{extr } \gamma \ P \ \emptyset & \widehat{=} & (\gamma, P) \\
\text{extr } \ \gamma \ \ P \ \ (A \sqcup \{\, a \,\}) & \widehat{=} & \text{extr } \ (\gamma \sqcup \{a \mapsto \lhd[H_a]P\}) \ \ (\ntriangleleft[H_a]P) \ \ A \\
H_a \ \rho & \widehat{=} & a \in (\texttt{elems} \circ {\pi_1}^\star)\rho
\end{array}$$

Here $\lhd$ and $\ntriangleleft$ are parameterised using predicates, rather than sets.

We bring this all in together to obtain:

$$\begin{array}{rcl}
\mathcal{R}(P) & \widehat{=} & \texttt{iterate}\,(=)\ \mathcal{RS}\ (\emptyset, P) \\
\mathcal{RS}\ (\gamma, P) & \widehat{=} & \text{extr } \gamma \ A' \ P \\
\textbf{where} & & \mathcal{R} = \text{toChRel } \rho \\
 & & \kappa = \text{toChReqs } \rho \\
 & & A' = \texttt{mins}(\texttt{filter}[\text{Act}[\kappa]]\mathcal{R})
\end{array}$$

Note that the channel partial order is computed afresh for each iteration of $\mathcal{RS}$. In each iteration we simply extract all the highest priority active channels (i.e those lowest in the ordering).

4 Resolution Properties

We now mention a few properties that we expect to hold for resolution. In what follows we assume throughout that

$$\kappa = \text{toChRel } P$$

and

$$\mathcal{R}\ P = (\gamma, B)$$

and these properties hold if all variables are identically decorated (subscripts or superscripts).

4.1 Termination

$$\mathcal{R} \text{ terminates}$$

The key to proving this is to recognise the crucial rôle played by A' in the definition of $\mathcal{RS}$. If A' is empty, then extr leaves its argument unchanged.

$$\text{extr } \gamma\ P\ \emptyset = (\gamma, P)$$

In this case, the `iterate` function terminates. If A' is non-empty, then extr does at least one iteration, which increases γ and decreases P. As P is finite, its reduction in size must eventually terminate. This proof is an informal one, but basically boils down to an assertion that the size of P supplies the well-founded measure required to show termination.

4.2 Resolution Progress

As long as the set of `prialts` to be processed is non-empty and contains active requests, then $\mathcal{RS}$ will always make progress by enlarging γ and shrinking P. If P is empty or has no active channels, then $\mathcal{RS}$ leaves its input unchanged:

$$\begin{array}{rcl} (\gamma', P') & = & \mathcal{RS}(\gamma, P) \\ & \Rightarrow & \mathtt{dom}\,\gamma' \supset \mathtt{dom}\,\gamma \ \wedge\ P' \subset P \ \ \vee\ \ \gamma' = \gamma \ \wedge\ P' = P \end{array}$$

Also:

$$(\gamma, P) = \mathcal{RS}(\gamma, P) \equiv (\frown/ \circ \mathbb{P}(\mathtt{filter}[\text{Act}_k]))P = \Lambda$$

where κ is derived from P in the usual way.

4.3 Resolution Chains

Repeated application of $\mathcal{RS}$ results in the following (finite) chain:

$$(\theta, P_0) \to (\gamma_1, P_1) \to \cdots \to (\gamma_i, P_i) \to \cdots \to (\gamma_n, P_n)$$

where $\mathcal{RS}(\gamma_n, P_n) = (\gamma_n, P_n)$

So we can define an ordering:

$$(\gamma, P) \preceq (\gamma', P') \ \ \widehat{=}\ \ \exists n \ \bullet\ (\gamma', P') = \mathcal{RS}^n(\gamma, P)$$

An open question: can we find (γ_a, P_a) and (γ_b, P_b), such that neither $(\gamma_a, P_a) \preceq (\gamma_b, P_b)$, or vice versa, but there exists (γ', P'), and numbers n and m such that

$$(\gamma', P') = \mathcal{RS}^n(\gamma_a, P_a) = \mathcal{RS}^(\gamma_b, P_b)$$

If not, then each member of the chain $(\theta, P_0) \to \cdots \to (\gamma_n, P_n)$ is uniquely determined by the first element, and hence by P_0.

4.4 Resolution Invariant

The set of all `prialts` present is invariant under the $\mathcal{RS}$ operation. We can define the set of `prialts` present as:

$$\text{pAltOf}(\gamma, P) = P \cup ({}^{\cup}/ \circ \mathtt{rng})\gamma$$

We then assert that:

$$\text{pAltOf}(\gamma, P) = (\text{pAltOf} \circ \mathcal{RS})(\gamma, P)$$

We can extend this to the overall resolution process by asserting:

$$\mathcal{R}(P) = (\gamma', P') = \mathcal{R}(\theta, \text{pAltOf}(\mathcal{R}(P)))$$

4.5 Resolution Validity

We can define a predicate which determine when a $Resltn$ structure is valid, namely that it is the result of iterating $\mathcal{RS}$ a finite number of times on a starting resolution structure with no mapping and a set referring to all `prialts` present:

$$\begin{array}{rcl} \text{isValid} & : & Resltn \to \mathbb{B} \\ \text{isValid}(\gamma, P) & \widehat{=} & \exists n \bullet (\gamma, P) = \mathcal{RS}^n(\theta, \text{pAltOf}(\gamma, P)) \end{array}$$

4.6 Active Channel Screening

An active channel at a position in a `prialt` effectively removes the possibility of a subsequent channel in that `prialt` being selected.

$$\text{Act}_\kappa c \wedge \pi_1(\rho[i]) = c \Rightarrow \mathcal{R}\ (P \sqcup \{\, \rho \,\}) = \mathcal{R}(P \sqcup \{\, \rho[1 \ldots i] \,\})$$

where

$$\kappa = \text{toChReqs}(P \sqcup \{\, p \,\})$$

It is this property which is lost if we admit cycles in the priority relation and then allow for a non-deterministic breaking of such cycles—consider breaking the above just after $\rho[i]$. The entry $\rho[i+1]$ (if it exists) is no longer "screened" by the ith entry and in fact becomes minimal in the new relation.

4.7 Disjoint PriAlt Groups

If we have two disjoint collections of `prialts` (channels distinct) then we can resolve them together or separately.

$$\mathtt{dom}\ \kappa_1 \cap \mathtt{dom}\ \kappa_2 = \emptyset \Rightarrow \mathcal{R}(P_1 \sqcup P_2) = (\gamma_1 \sqcup \gamma_2, B_1 \sqcup B_2)$$

5 Compositionality of **prialt**

In order to achieve full compositionality, we need to find an operator ‡ such that

$$\mathcal{R}(P \cup Q) = \mathcal{R}(P) \ddagger \mathcal{R}(Q)$$

This property is necessary in order to provide a denotational semantics, and makes it much easier to produce a set of useful laws for reasoning about Handel-C programs.

It turns out that this is very easy to achieve, at least in a technical sense. Given the output of a resolution, we can use pAltOf to give us the original input data, so we can define ‡ as:

$$(\gamma', P') \ddagger (\delta', Q') = \mathcal{R}(\text{pAltOf}(\gamma', P') \cup \text{pAltOf}(\delta', Q'))$$

This definition, while technically correct, seems unsatisfactory, in the sense that it involves taking both resolutions, undoing them, throwing all the original `prialts` back into the pot, and redoing the entire resolution process again. This idea that composing resolutions involves retracting previously granted requests seems a little unsatisfactory.

However, as observed in [14], when processes are combined such that relative priorities get changed, then events accepted by the combined process my be different from those accepted by the individual process. This is in fact inherent in the whole concept of priority. Consider the following example:

$$P = \left\{ \begin{array}{c} \langle a!, b? \rangle \\ \langle b! \rangle \end{array} \right\} \qquad Q = \left\{ \begin{array}{c} \langle a?, c? \rangle \\ \langle c! \rangle \end{array} \right\}$$

In both cases, channel a has highest priority, but is inactive, so channels b and c are selected:

$$\begin{array}{rcl} \mathcal{R}(P) & = & (\{b \mapsto \{\, \langle a!, b? \rangle, \langle b! \rangle \,\}\}, \emptyset) \\ \mathcal{R}(Q) & = & (\{c \mapsto \{\, \langle a?, c? \rangle, \langle c! \rangle \,\}\}, \emptyset) \end{array}$$

However, when we combine P and Q:

$$P \cup Q = \left\{ \begin{array}{c} \langle a!, b? \rangle \\ \langle b! \rangle \\ \langle a?, c? \rangle \\ \langle c! \rangle \end{array} \right\}$$

we find the channel a is now active, and removing its `prialts` from the pool results in channels b and c being inactive, when the leftover `prialts` are considered:

$$\mathcal{R}(P \cup Q) = (\{a \mapsto \{\, \langle a!, b? \rangle, \langle a?, c? \rangle \,\}\}, \{\, \langle b! \rangle, \langle c! \rangle \,\})$$

It is clear that the phenomenon of retracting previously offered events in the light of new priority information resulting from the introduction of new processes is an inherent aspect of priority itself.

We are forced to accept that priority requires, in principle, that we effectively re-compute our resolutions every time new `prialts` are added. However, another question of interest arises, regarding the possibility of being able to do some form of incremental adding and re-resolving, and the circumstances under which this is possible. Unfortunately it clear that very small additions to a large collection of `prialts` can have potentially large effects. To see this consider a large collection of `prialts` where the active channels happen to be separable into two disjoint sets such that any channel is cited only in one or other set but not both. In effect we have two independent sets of `prialts`. Let us also assume that all the channels are totally ordered within those disjoint subsets. All we have to do is to add one new `prialt` which ranks the highest priority channel from one subset as lower than the lowest priority channel of the other subset to completely alter the ordering relationship. It appears very difficult to handle a case like this without re-computing the resolutions for the entire set of `prialts`.

6 Conclusions and Future Work

We have described a formal process for resolving `prialt` communication requests, suitable for handling the type of `prialts` found in the Handel-C programming language. We have shown this process to be compositional in nature, at least in a technical sense, and we have described some key properties.

The first task to be done at this stage is to integrate this material with the existing semantics for Handel-C [10, 11] without `prialt`, and then to use the denotational semantics to validate a collection of useful language-level laws for Handel-C. The long term goal is to develop a development methodology for deriving Handel-C programs from appropriate specifications. This involves the development of a suitable refinement calculus, which will be integrated into the unifying theory of Hoare & He [17], using the *Circus* refinement calculus [18].

On a more speculative note, we believe that it may be possible to encode a semantics of Handel-C in standard CSP, by using the functional language component as found in machine readable CSP (see [19, Appendix C]) to encode the resolution. This would use a mechanism of first lodging and then resolving requests similar to that described in [10] for non-`prialt` communication. The reason that the extra machinery of $\mathcal{CSPP}$ might not be required is simply the constrained deterministic nature of Handel-C. We do not see this as being possible for `prialt` in a more general setting, as described in [4]

References

[1] Celoxica Ltd. *Handel-C Language Reference Manual, v3.0*, 2002.

[2] A. W. Roscoe. Denotational Semantics for *occam*. In S. D. Brookes, A. W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *LNCS*. Carnegie-Mellon University, Pittsburgh, PA, Springer-Verlag, July 1984.

[3] A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. Technical Monograph PRG-53, Oxford University Computing Laboratory Programming Research Group, February 1986.

[4] 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.

[5] C.A.R. Hoare. *Communicating Sequential Processes*. Intl. Series in Computer Science. Prentice Hall, 1990.

[6] I. Page and W. Luk. Compiling Occam into field-programmable gate arrays. In W. Moore and W. Luk, editors, *FPGAs, Oxford Workshop on Field Programmable Logic and Applications*, pages 271–283. Abingdon EE&CS Books, 15 Harcourt Way, Abingdon OX14 1NV, UK, 1991.

[7] M. Spivey and I. Page. How to design hardware with Handel. Technical report, Oxford University Hardware Compilation Group, December 1993.

[8] J. P. Bowen, He Jifeng, and I. Page. Hardware compilation. In J. P. Bowen, editor, *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*, chapter 10, pages 193–207. Elsevier, 1994.

[9] Adrian Lawrence, Andrew Kay, Wayne Luk, Toshio Nomura, and Ian Page. Using reconfigurable hardware to speed up product development and performance. In Will Moore and Wayne Luk, editors, *Field-Programmable Logic and Applications*, pages 111–119. Springer-Verlag, Berlin, August/September 1995. Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications, FPL 1995. Lecture Notes in Computer Science 975.

[10] Andrew Butterfield. Denotational semantics for `prialt`-free Handel-C. Technical Report TCD-CS-2001-53, Dept. of Computer Science, Trinity College, Dublin University, 2001. `ftp://ftp.cs.tcd.ie/pub/tech-reports/reports.01/TCD-CS-2001-53.pdf`.

[11] Andrew Butterfield. Interpretative semantics for `prialt`-free Handel-C. Technical Report TCD-CS-2001-54, Dept. of Computer Science, Trinity College, Dublin University, 2001. `ftp://ftp.cs.tcd.ie/pub/tech-reports/reports.01/TCD-CS-2001-54.pdf`.

[12] Mícheál Mac an Airchinnigh. *Conceptual Models and Computing*. PhD dissertation, University of Dublin, Trinity College, Department of Computer Science, 1990.

[13] ISO/IEC. Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language. International Standard 13817-1, International Standards Organisation, 1996. `www.iso.ch`.

[14] A. E. Lawrence. CSPP and event priority. In Majid Mirmehdi Alan Chalmers and Henk Muller, editors, *Communicating Process Architectures 2001*, Concurrent Systems Engineering, Amsterdam, September 2001. IOS Press.

[15] 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.

[16] Embedded Solutions Ltd. (now Celoxica Ltd.). *Handel-C Language Reference Manual, v2.1*, 2000.

[17] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Series in Computer Science. Prentice Hall, 1998.

[18] Jim Woodcock and Ana Cavalcanti. The Semantics of Circus. volume 2272 of *LNCS*. 2nd International Conference of B and Z Users, Grenoble, France, Springer, January 2002.

[19] A. W. Roscoe. *The Theory and Practice of Concurrency*. international series in computer science. Prentice Hall, 1997.

A Notation and Definitions

We provide a brief glossary of notations and definitions of the less familiar *VDM♣* notations used in this paper.

Symbol	Description	*Examples*
$\#$	Cardinality	
$\mathbb{P}f$	Set Mapping	$\mathbb{P}f\{\,x,y,z\,\} = \{\,f(x), f(y), f(z)\,\}$
`elems`	Sequence Elements	$\texttt{elems}\langle b,a,c,a\rangle = \{\,a,b,c\,\}$
`len`	Sequence Length	
$f^\star$	Sequence Mapping	$\mathbb{P}f\langle x,y,z\rangle = \langle f(x), f(y), f(z)\rangle$
π_i	ith Projection	$\pi_i(a_1, \ldots a_i, \ldots, a_n) = a_i$
$\sigma[i]$	Sequence Indexing	
$\sqcup$	Map Extension	
$\dagger$	Map Override	
`dom`	Map Domain	
`rng`	Map Range	
$\mu(x)$	Map Application (lookup)	
$^\star/$	Set or Sequence Reduction	$^+/\langle 3,5,4\rangle = 12$
`TC`	Transitive Closure Operator	
`order`	Sequence to Total Order	
`acyclic`	Acyclic Predicate	
`filter`	Sequence Filter	
$\lhd[S]$	Map Removal w.r.t. S	$\lhd[\{\,a,c\,\}]\{a\mapsto 1, b\mapsto 2, c\mapsto 3\} = \{b\mapsto 2\}$
`mins`	Minimal Elements of a P.O.	

Given any binary operator $\star : A \times A \to A$ and an arbitrary indexing set X, we "index" the operator, calling it $\circledast : (X \to A) \times (X \to A) \to (X \to A)$ and defining its action so that:

$$\mu \circledast \{x \mapsto a\} = \begin{cases} \mu \sqcup \{x \mapsto a\}, & \textbf{if } x \notin \mathsf{dom}\,\mu \\ \mu \dagger \{x \mapsto \mu(x) \star a\}, & \textbf{if } x \in \mathsf{dom}\,\mu \end{cases}$$

Effectively $\star$ has been "lifted" from acting on A to acting on $X \to A$, essentially operating on the map range elements.

The iterator application $\texttt{iterate}\ P\ f\ x$ repeatedly applies f to x and uses P to examine the before and after values to determine when to stop:

$$\begin{aligned} \texttt{iterate} &: (A \times A \to \mathbb{B}) \to (A \to A) \to A \to A \\ \texttt{iterate}\ P\ f\ x &\mathrel{\widehat{=}} \textbf{let } x' = f(x) \\ &\phantom{\mathrel{\widehat{=}}}\ \textbf{in if } P(x,x') \textbf{ then } x' \textbf{ else } \texttt{iterate}\ P\ f\ x' \end{aligned}$$