

ULRR

Prioritized slotted-Circus

Item Type	Meetings and Proceedings
Authors	Gancarski, Pawel;Butterfield, Andrew
Citation	ICTAC'10 Proceedings of the 7th International Colloquium Conference on Theoretical Aspects of Computing;6255,2010/ pp. 91-105
Publisher	Springer-Verlang
Download date	2026-04-15 13:32:46
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/695

Prioritized slotted-Circus^{*}

Pawel Gancarski¹ and Andrew Butterfield¹

Lero@TCD, Trinity College Dublin, {gawcarsp,butrfield}@tcd.ie

Abstract. This paper describes an extension adding priority to *slotted-Circus*, a generic framework for reasoning about discretely timed and/or synchronously clocked systems. The semantics of prioritised external choice is given using the Unifying Theories of Programming framework (UTP). The resulting language is similar to Prioritized Timed CSP, but its semantics is not based on trace ordering, and neither does it use the notion of acceptances (e.g. PCSP). Instead, the semantics is based on the notion of refusal sets already widely used in theories of CSP, *Circus* and *slotted-Circus*. We introduce priority as a lightweight extension of *slotted-Circus*, which can be easily adapted to define a similar extension of Timed CSP. We also discuss why priority can most easily be added to specific history models, and the fact that requiring the clock to tick after every communication event results in a more tractable theory.

1 Introduction

1.1 Prioritized slotted-Circus

The formal notation *Circus* is an unification of *Z* and CSP, to give a “state-rich” process algebra with (restricted) global shared variables. *Circus* has been given a semantics in UTP [OCW09]. Apart from event sequencing, there is no notion of time in *Circus*. A timed version of *Circus* (*Circus* Time Action or CTA) has been explored [SH02, She06] that introduces the notion of discrete time-slots in which sequences of events occur. The semantics of CTA has been developed using UTP, and there we find a two-level notion of history: the top-level views history as a sequence of time-slots; whilst the bottom-level records a history of events within a given slot.

Our interest in hardware compilation languages such as Handel-C [Cel02] led to a development of a generic theory (called *slotted-Circus*), with time-slots whose bottom-level contents could be parameterised, as simple traces, or multisets of events, or as one of the three successively more complex “micro-slot” structures [BSW07]. *slotted-Circus* has also been given a semantics in UTP [GB09].

One important aspect of hardware compilation languages, namely priority, has not been addressed before in Circus based languages. Priority is a very basic and intuitively simple concept, used to express that one thing is regarded as

^{*} This research was supported by grant 07-RFP-CMSF186 from Science Foundation Ireland, as well as partial support from Lero, the Irish Software Engineering Research Centre

more important than another. Even though in the original CSP, priority was not defined, many languages that adopted the CSP model of communication added priority constructs (occam, Ada, Handel-C), usually as they provide for efficient implementations. Even though in this work we are primarily interested in priority for *slotted-Circus* we will also discuss it in the context of CSP. Because this work is focused on defining a theory for hardware description languages, we will only be interested in defining prioritized versions of external choice ($\overleftarrow{\square}$) and not in prioritized parallel composition ($\overleftarrow{\parallel}$). This is because hardware can utilise true parallelism, and does not require interleaving of notionally parallel processes, with the attendant need to be able to prioritize aspects of that interleaving (a.k.a. scheduling). One of the interesting aspects of Prioritized *slotted-Circus* is that it is very similar to Prioritized Timed CSP (PTCSP) [Low93]. The two notations have different origins and semantic models, but the degree of convergence in key notions and laws, gives us confidence that our priority concept is correct. In PTCSP we can remove non-determinism from a mixture of parallel composition and external choice by prioritising them both, as per the following law:

$$(a \xrightarrow{n} P \overleftarrow{\square} a \xrightarrow{n} Q) \overleftarrow{\parallel} (a \xrightarrow{n} Q \overleftarrow{\square} a \xrightarrow{n} P) = a \xrightarrow{n} P$$

In Prioritised *slotted-Circus*(PSC), the lack of a prioritized parallel construct means that we cannot so easily remove all non-determinism during implementation:

$$(a \xrightarrow{n} P \overleftarrow{\square} a \xrightarrow{n} Q) \parallel (a \xrightarrow{n} Q \overleftarrow{\square} a \xrightarrow{n} P) = a \xrightarrow{n} P \sqcap a \xrightarrow{n} Q$$

1.2 UTP: General Principles

Theories in UTP are expressed as second-order predicates¹ over a pre-defined collection of free observation variables, referred to as the *alphabet* of the theory. The predicates are generally used to describe a relation between a before-state and an after-state, the latter typically characterised by dashed versions of the observation variables. A predicate whose free variables are all undashed, referring only to the before-state, is called a *condition*. We note that UTP follows the key principle that “programs are predicates” [Hoa85b] and so does not distinguish between the syntax of some language and its semantics as alphabetised predicates. For example, if *ok* denotes absence of divergence, *tr* is a sequence of observed events and *ref* is a set of events currently being refused, then

$$ok' \wedge tr' = tr \frown \{a\} \wedge a \notin ref'$$

denotes an observation of a non-divergent process execution that has performed an *a*-event and is still willing to perform more. A given theory is characterised by its alphabet, and a series of *healthiness conditions* that constrain the valid assertions that predicates may make. A healthiness condition is a property of a

¹ Most definitions are in fact 1st-order, but we need 2nd-order in order to handle the notion of “healthiness”, and recursion.

```

Action ::= Skip | Stop | Chaos | Wait t | Name := Expr
        | Comm → Action | Action □ Action | Action □ Action
        | Action ; Action | Action || Action | Action \ CS
Comm ::= Name.Expr | Name!Expr | Name?Name
Expr ::= expression
      t ::= positive integer valued expression
Name ::= channel or variable names
CS ::= channel name sets

```

Fig. 1. Slotted-*Circus* Syntax

predicate that distinguishes sensible predicates from nonsense. So, for example the following predicate is clearly nonsense under our intended interpretation:

$$tr = tr' \wedge \langle a \rangle$$

It asserts that the history of events that had occurred before this process started (tr) is longer than the history at the end (tr'). It can be ruled out by the following healthiness condition:

$$P \Rightarrow tr \leq tr'$$

Note that healthiness conditions should not be confused with ordinary conditions (predicates with only before-variables).

1.3 Structure and Focus

We first present an overview of *slotted-Circus* semantics §2, before giving an extensive exposition of the prioritised theory in §3. We then wrap up by discussing related §4 and future §5 work, and concluding §6.

2 Slotted-Circus

In this section we will focus only on aspects of *slotted-Circus* relevant to this paper. More detailed definitions and explanations can be found in [BG09, GB09].

2.1 Syntax

The syntax of Slotted-*Circus* is similar to that of *Circus*, and a subset, relevant to this paper, is shown in Figure 1. Apart from assignment, we shall ignore the imperative state aspects of the theory as these are covered elsewhere and not relevant to the topic of priority. The basic actions *Skip*, *Stop*, *Chaos*, as well as event prefix ($e \rightarrow A$) and hiding ($A \setminus H$) are similar to the corresponding CSP behaviours [Hoa85a, Sch00], while we also introduce variable assignment ($:=$). Actions can be combined with internal (\sqcap) or external (\square) choice, sequential composition ($;$), or parallel composition ($||$). The key construct related to time-slots, and hence not part of *Circus*, is *Wait t*, which denotes an action that simply waits for t time-slots to elapse, and then terminates.

2.2 History models

In *slotted-Circus* a trace model is built on top of exchangeable history models. It is recorded as a sequence of slots, where every slot is defined as a pair consisting of an event history (*hist*) and a refusal set (*ref*), meaning that in the relevant time-slot that event-history *hist* occurred, with events in *ref* being refused afterwards.

$$SE \cong \mathcal{H}E \times \mathbb{P}E$$

There are currently two history models defined and working within the *slotted-Circus* framework: MSA and CTA. In CTA, a history is just a sequence (“trace”) of events in the order in which they occurred during a slot. So the following example shows a run of CTA:

$$\langle \langle \rangle, ref_1 \rangle, \langle \langle a, b \rangle, ref_2 \rangle, \langle \langle b, a, a \rangle, ref_3 \rangle, \langle \langle b, a \rangle, ref_4 \rangle, \dots \rangle$$

In the multi-set action (MSA) variant, we ignore event ordering within slots, viewing histories as a bag of events, so the above example appears as:

$$\langle \langle \{ \} \rangle, ref_1 \rangle, \langle \langle \{ a \mapsto 1, b \mapsto 1 \} \rangle, ref_2 \rangle, \langle \langle \{ a \mapsto 2, b \mapsto 1 \} \rangle, ref_3 \rangle, \langle \langle \{ a \mapsto 1, b \mapsto 1 \} \rangle, ref_4 \rangle, \dots \rangle$$

The MSA history model carries no information on event ordering within time-slots.

2.3 UTP Observations

In our UTP theory, we model *slotted-Circus* using four observations:

ok : \mathbb{B} — stability, absence of serious error.

wait : \mathbb{B} — waiting, true if process is waiting on events, false if it has terminated.

slots : $(SE)^+$ — full event history as a non-empty sequence of slots, with “clock-ticks” occurring at the boundaries between slots.

state : *Variable* \leftrightarrow *Value* — an environment giving program variable values.

The alphabet of our theory consists of the above four variables representing the state before an action starts, and dashed versions of the variables giving the (current/final) state when an action is running and either waiting for an event or just terminated.

2.4 Healthiness Conditions

Healthiness conditions are characterised by idempotent predicate transformers, with a healthy predicate being a fixed point of such a transformer. Here we shall only consider **R3**, **CSP1,2,3,4** as they are explicitly invoked. **R1** and **R2** deal with the infeasibility of time travel and (direct) memory of past events, and are well covered elsewhere, and satisfied in any case by all definitions we present.

The healthiness condition **R3** is one associated with all “reactive” systems in the UTP, covering process-algebras like ACP, CSP, and CCS.

$$\mathbf{R3}(P) \cong \mathbb{I} \triangleleft \text{wait} \triangleright P$$

R3 deals with the situation when a process has not actually started to run, because a prior process has yet to terminate, characterised by $wait = \text{TRUE}$. In this case the action of a yet-to-be started process should simply be to do nothing, an action we call “reactive-skip” (**I**).

A process is **CSP1** healthy if *all* it asserts, when started in an unstable state (due to some serious earlier failure), is that the event history may be extended:

$$\mathbf{CSP1}(P) \hat{=} P \vee \neg ok \wedge slots \preceq slots'$$

A process predicate is **CSP2** healthy if it does not mandate instability, so if true with $ok' = \text{False}$, it is also true with $ok' = \text{True}$, all other observation variables being unchanged.

$$\mathbf{CSP2}(P) \hat{=} P; (ok \Rightarrow ok') \wedge wait' = wait \wedge slots' = slots \wedge state' = state$$

CSP3 and **CSP4** state respectively that *Skip* is a left and right unit of sequential composition.

$$\mathbf{CSP3}(P) \hat{=} Skip; P \qquad \mathbf{CSP4}(P) \hat{=} P; Skip$$

A more technical aspect of **CSP3,4** is that once *Skip* starts/terminates it unconstrains the refusals set of the last slot. This causes **CSP3** to make processes insensitive to refusals of a previously terminated process, and **CSP4** to unconstrain refusals of a processes once they terminate. These properties are changed in our prioritized model, allowing information about refusals to be partially propagated even when a process terminates.

2.5 Slotted Semantics

The language constructs of sequential composition and internal choice all have the same semantics as in standard UTP:

$$\begin{aligned} P; Q &\hat{=} \exists obs_m \bullet P[obs_m/obs'] \wedge Q[obs_m/obs] \\ P \sqcap Q &\hat{=} P \vee Q \end{aligned}$$

Here *obs* is shorthand for all the observational variables.

Semantic Building Blocks We define the semantics of *slotted-Circus* in terms of a number of basic predicate building-blocks, largely to do with events and communication, that we now describe informally. The building blocks are all **R1**-, **R2**-healthy, but in general will not satisfy **R3** or the CSP healthiness conditions in themselves— they are intended to be used in constructions that do.

NOEVTS describes a situation that allows time to pass ($\#slots' > \#slots$) but disallows the occurrence of any events (all slot histories are empty).

FSTEVTS(*E*) asserts that a given set of events (*E*) have occurred immediately (in the first time slot).

IMMEVTS describes a situation when some events occur immediately (in the first slot). It can be defined as $\exists E \bullet E \neq \emptyset \wedge \text{FSTEVTS}(E)$.

$$\begin{aligned}
Chaos &\hat{=} \mathbf{R}(\mathbf{true}) \\
Miracle &\hat{=} \mathbf{CSP1}(\mathbf{R3}(\mathbf{FALSE})) \\
Stop &\hat{=} \mathbf{CSP1}(\mathbf{R3}(ok' \wedge wait' \wedge \mathbf{NOEVTS})) \\
Skip &\hat{=} \mathbf{R3}(\mathbf{CSP1}(state = state' \wedge \neg wait' \wedge ok' \wedge slots \cong slots')) \\
Wait\ t &\hat{=} Stop, \quad \text{if } len(slots' - slots) < t \\
&\quad Skip, \quad \text{if } len(slots' - slots) = t \\
c!e \rightarrow Skip &\hat{=} c.e \rightarrow Skip \\
c \rightarrow A &\hat{=} (c \rightarrow Skip); A \\
c?x \rightarrow Skip &\hat{=} \bigsqcap_{k:T} \bullet (c.k \rightarrow Skip; x := k)
\end{aligned}$$

Fig. 2. Semantics of basic actions

Semantics of Basic Actions The semantics of the basic actions are described in Fig.2. The worst possible action in *slotted-Circus* is *Chaos*. It is the most unpredictable healthy process, and bottom of the refinement lattice. Action *Miracle* is the top of the refinement lattice, and is a program satisfying any specification (clearly infeasible), but useful as a unit of nondeterministic choice ($Miracle \sqcap P = P$). Action *Stop* has deadlocked: it is stable, never terminates and never performs any event. Action *Skip* terminates immediately in a stable state, without performing any events. In keeping with the CSP definition, *Skip* ignores the refusals of any preceding process, hence the use of slot-equivalence (\cong) here, which is slightly weaker than slot-equality, in that it ignores the refusals in the last slot. The action that introduces explicit timed behavior is *Wait t*. It never performs any events and has only two possible behaviors. The first one is to wait for t clock ticks, the second to terminate when the right time is reached.

Event prefix ($c \rightarrow A$) is defined using $c \rightarrow Skip$ composed with A . Output prefixes are simply event prefixes, whilst input prefixes are modelled as an external choice over all possible input values, with assignment being used to capture the outcome. The process $c \rightarrow Skip$ is defined using two basic actions:

$WTC(c)$ allows time to pass without any events occurring, while never refusing to perform event c .

$TRMC(c)$ performs event c in the first time-slot.

Basically while non-terminated, $c \rightarrow Skip$ acts like WTC , and once event c occurs (if at all), it then has the behaviour $WTC(c); TRMC(c)$ — waiting followed by event c and termination:

$$c \rightarrow Skip \hat{=} \mathbf{CSP1} \left(ok' \wedge \mathbf{R3} \left(WTC(c) \triangleleft wait' \triangleright \left(\begin{array}{l} state' = state \wedge \\ WTC(c); TRMC(c) \end{array} \right) \right) \right)$$

Semantics of Composite Actions External choice ($A \sqcap B$) allows external events to determine which action runs, so for example if we have $(a \rightarrow A) \sqcap (b \rightarrow B)$, then, if the environment performs a , we see that event occur, followed by an execution of action A . Unfortunately, the very simple definition² of external choice proposed in [HH98] no longer suffices, as we may have to wait for several clock-ticks before an external event arises that resolves the choice.

$$A \sqcap B \hat{=} \mathbf{CSP2}(Stop \wedge A \wedge B \vee Choice(A, B) \vee Choice(B, A))$$

$$Choice(C, R) \hat{=} C \wedge \left(R \wedge NOEVTS; \left(\begin{array}{l} IMMEVTS \vee \\ slots \cong slots' \wedge (\neg wait' \vee \neg ok') \end{array} \right) \right)$$

Predicate $Choice(C, R)$ describes the circumstances where action C has been chosen, whilst R has been refused, which occurs in situations where R has performed no events. We capture these cases as follows: conjoin R with $NOEVTS$, and follow it sequentially with some “end”-condition E . All of this is conjoined with C to give

$$C \wedge (R \wedge NOEVTS; E)$$

i.e. an execution of C consistent with R having done no events, and then ending in the situation described by E .

Now we can characterise three possible cases were C either: (i) performs an event after a delay: $E = IMMEVTS$; (ii) terminates without performing any events: $E = slots \cong slots' \wedge \neg wait'$ or (iii) diverges but performs no event: $E = slots \cong slots' \wedge \neg ok'$.

The parallel composition $A \parallel B$ runs A and B in lock-step parallel (clock ticks at same time for both). Both actions run on local copies of the variables. The construct terminates when both actions have terminated — if one ends early then its behaviour is padded out with empty slots.

$$A \setminus H \hat{=} \mathbf{R3} \left(\begin{array}{l} \exists s' \bullet A[s'/slots'] \wedge \\ slots' \setminus\setminus slots = map(SHide(H))(s' \setminus\setminus slots) \\ \wedge H \subseteq \bigcap Refs(s' \setminus\setminus slots) \end{array} \right); Skip$$

The hiding operator $A \setminus H$ denotes an execution of action A , but with any events in event-set H hidden. Function $SHide(H)$ removes events in H from a slots history component, and adds them into the refusal set. We enforce a key property of hiding, namely that of *maximal progress*, i.e. hidden events occur as soon as they are enabled. Without this semantic feature the following undesirable law would hold:

$$(a \rightarrow Skip) \setminus \{a\} = Wait\ 0 \sqcap Wait\ 1 \sqcap \dots \sqcap Wait\ n \sqcap \dots$$

This law is undesirable because it makes the performance of a single hidden event followed by termination equal to a wait for an arbitrary number of clock cycles — effectively a weak form of livelock. By forcing hidden events to be refused during every slot, we prevent them from waiting for a clock-tick, because the

² $A \sqcap B \hat{=} A \wedge B \triangleleft Stop \triangleright A \vee B$

definition of prefix action requires events not to be refused when waiting. This results in the desired law, namely $(a \rightarrow Skip) \setminus \{a\} = Skip$. At the end we add $Skip$ to unconstrain the refusals of the last slot.

3 Prioritized slotted-Circus

As mentioned in the introduction, prioritized choice has been added to CSP inspired programming languages with concurrency, like `occam` and `Ada`. These languages were designed on foot of formal semantics covering most language constructs [Ros84], but with priority being an exception. The approach then taken instead was to have a semantics for external choice, or its equivalent, and then consider the prioritised forms to be refinements of the non-prioritised versions, thus

$$(P \sqcap Q) \sqsubseteq (P \sqcap^{\leftarrow} Q)$$

justified their use in implementations. We note semantics for these languages that covered priority did emerge afterwards — e.g. [Cam89]. In hardware description languages, priority is very important as implementations need to be deterministic and priority is an effective and efficient way of achieving this. Indeed, in `Handel-C`, the only form of external choice is prioritised, appearing as the so-called “prialt” language construct. Our aim is to extend the work *slotted-Circus* to cover this important feature.

3.1 Prioritized CSP

The difficulty in formalizing priority is the need for a clear idea of when it gets resolved, i.e. how long should we wait for possible options before we try to choose the one we prefer most? The problems appear in CSP because it is impossible to say how much time has passed, but it is possible to determine event ordering. The problem can be best described using two laws:

$$\begin{aligned} Stop \sqcap P &= P \\ Skip \sqcap P &= Skip \sqcap \begin{cases} P, & \text{if } P \text{ terminates or performs an event} \\ Miracle, & \text{otherwise} \end{cases} \end{aligned}$$

Now, if we consider be the consequence of the laws above after prioritizing the choice we get:

$$Stop \sqcap^{\leftarrow} P = P$$

Prioritized choice is an implementation of external choice, so no other outcome is possible. However, with $Skip$, because the termination event is immediately available and beats any other external event, we expect to see:

$$Skip \sqcap^{\leftarrow} P = Skip$$

Because in CSP we can not measure the passing time, the following law holds:

$$Q \setminus Events = \begin{cases} Skip, & \text{if } Q \text{ terminates} \\ Stop, & \text{otherwise} \end{cases}$$

for that reason, when we combine our results, we get:

$$(Q \setminus Events) \overset{\leftarrow}{\square} P = \begin{cases} Skip, & \text{if } Q \text{ terminates} \\ P, & \text{otherwise} \end{cases}$$

So, any implementation of prioritized choice would have to be able to solve the Halting problem.

3.2 Zero-causality problems

The problem in defining priority for CSP is that we have no finite deadline for the choice to be resolved. In *slotted-Circus*, by contrast, this problem seems to be solved—the deadline is a clock tick. In other words we expect prioritized choice to deal with a situation when two processes are ready to perform an event within the same time slot. Unfortunately a new problem has appeared. However we also have a property from the semantics of *slotted-Circus* of “zero-causality”, denoting the fact that communication/events take no time. Because of the maximal urgency principle and zero-causality, if we use hiding on events performed one after another $((a \rightarrow b \rightarrow Skip) \setminus \{a, b\})$, from one perspective we might say that they are performed at the same time (because within the same time slot), but on the other hand we know that an event b occurred after event a . This fact arises separate problems for both CTA and MSA.

In CTA the problem becomes visible once the prioritized choice operator has been defined.

$$((a \rightarrow b \rightarrow H \overset{\leftarrow}{\square} b \rightarrow a \rightarrow L) \parallel (b \rightarrow a \rightarrow S)) \setminus \{a, b\} =_{CTA} \text{Miracle}$$

In CTA we get a miracle, because of a conflict between CTA and priority. There is an tension between the history model in CTA that asserts that order matters, and the semantics we require for priority that waits to the end of the slot to sort everything out. This results intentially in a contradiction in CTA, leading to the miracle. Creating a definition of priority that would avoid the contradiction emerging here proved to be very difficult, requiring large changes in the semantics and for that reason was abandoned.

The MSA history model behaves properly with prioritized choice:

$$((a \rightarrow b \rightarrow H \overset{\leftarrow}{\square} b \rightarrow a \rightarrow L) \parallel (b \rightarrow a \rightarrow S)) \setminus \{a, b\} =_{MSA} (H \parallel S) \setminus \{a, b\}$$

However, the example above and the idea of zero-causality, led us to the discovery of strange properties unnoticed before in *slotted-Circus*. The first one is a violation of prefix closure:

$$(a \rightarrow b \rightarrow Skip) \parallel (b \rightarrow a \rightarrow Skip)$$

The described action can perform a and b at the same time, but can never perform either a or b alone. Another issue with the example above is that the maximal urgency property of hiding no longer holds:

$$((a \rightarrow b \rightarrow Skip) \parallel (b \rightarrow a \rightarrow Skip)) \setminus \{a, b\} =_{MSA} \bigsqcap_{n \in \mathbb{N}} Wait\ n \sqcap Stop$$

Things get more complicated if we try to “crossover” some information.

$$\begin{aligned} & \left(\begin{array}{l} (a?x \rightarrow b!x \rightarrow Skip) \\ \parallel (b?x \rightarrow a!x \rightarrow Skip) \end{array} \right) \setminus \{a, b\} =_{MSA} \bigsqcap_{n \in \mathbb{N}} (Wait\ n; x :=?) \sqcap Stop \\ & \left(\begin{array}{l} (a?x \rightarrow b!x \rightarrow Skip) \\ \parallel (b?x \rightarrow a!(x+1) \rightarrow Skip) \end{array} \right) \setminus \{a, b\} =_{MSA} Stop \end{aligned}$$

3.3 Timed Prefix

In order to deal with the problems presented above, it was decided to remove zero-causality from the language, by stipulating that communication always takes time. This was by the introduction of a new action — timed prefix:

$$a \xrightarrow{n} P =_{def} a \rightarrow Wait\ n; P$$

A similar assumption has been made in Prioritized Timed CSP and Handel-C.

So far no other problems of interaction between CTA and the prioritized model have been found, but because the notion of time assumed by priority better suits MSA our research is now focused on supporting this history model. It is still an open question what is the difference (if any), between *slotted-Circus* with CTA or MSA history model, when only timed communication is permitted. Another open problem is the existence of healthiness conditions, implying that communication takes time. So far our research leads us to suggest that it will be a special case of prefix closure, but no prefix closure healthiness condition has yet been defined in UTP theories.

3.4 Defining priority

In the timed theory, prioritized external choice is very similar in behaviour to external choice. In both cases the choice is resolved on a first-come first-served basis.

$$\begin{aligned} (a \xrightarrow{n} A \sqcap (Wait\ 1; b \xrightarrow{n} B)) \setminus \{a, b\} &= Wait\ n; A \setminus \{a, b\} \\ (a \xrightarrow{n} A \overleftarrow{\sqcap} (Wait\ 1; b \xrightarrow{n} B)) \setminus \{a, b\} &= Wait\ n; A \setminus \{a, b\} \end{aligned}$$

The differences become visible when an event (or termination) is available for both options at the same time. In that case the external choice becomes non-deterministic:

$$(a \xrightarrow{n} A \sqcap (b \xrightarrow{n} B)) \setminus \{a, b\} = Wait\ n; (A \sqcap B) \setminus \{a, b\}$$

while the prioritized choice chooses the higher priority option:

$$(a \xrightarrow{n} A \overleftarrow{\sqcap} (b \xrightarrow{n} B)) \setminus \{a, b\} = Wait\ n; A \setminus \{a, b\}$$

There are two important facts to observe here. The first one is that prioritized external choice is an implementation (refinement) of the normal one. Which

means that any behaviour accepted by it is also accepted by the external choice. The second fact is that the behaviour of the high priority choice is accepted iff it is accepted by the external choice. In other words we can describe priority using an external choice definition and a condition strengthening the low priority option.

$$A \sqsupset B \hat{=} (A \wedge B \wedge \text{Stop}) \vee \text{Choice}(A, B) \vee \text{WeakChoice}(B, A)$$

Where

$$\text{WeakChoice}(B, A) = \text{Choice}(B, A) \wedge \text{MagicCondition}$$

The only question left to be answered is: what is “MagicCondition”? To do that we need to bear in mind the following important laws:

- (1) $(a \xrightarrow{n} A \sqsupset b \xrightarrow{n} B) \setminus \{a, b\} = \text{Wait } n; A \setminus \{a, b\}$
- (2) $a \xrightarrow{n} A \sqsupset a \xrightarrow{n} B = a \xrightarrow{n} A$
- (3) $\text{Skip} \sqsupset A = \text{Skip}$
- (4) $a \xrightarrow{n} A \sqsupset b \xrightarrow{n} B \neq a \xrightarrow{n} A, \quad a \neq b$
- (5) $(P \sqcap Q) \sqsubseteq (P \sqsupset Q)$

The first three laws describe the difference in behaviour in comparison with external choice. Law (1) shows us a typical use case for priority, where two racing processes want to perform an event at exactly the same time. In that case hiding makes both of the events internal and they both are only willing to perform an event at the first time slot. In law (2) we can see a situation when we do not know when an event is going to be performed, but only that both of the racing processes will perform an event at the same time (because it is exactly the same event). For that reason the high priority option will always be chosen. Law (3) states that a termination is treated as an event and can resolve a prioritized choice. Finally law (4) is in contrast to both law (1) and (2) and it makes sure that the “MagicCondition” is not too strong.

WeakChoice can be best describe by contrast to the *Choice* definition.

$$\text{Choice}(P, S) \hat{=} \text{CSP2}\left(P \wedge \left(\begin{array}{l} (S \wedge \text{NOEVTS}); \\ \left(\begin{array}{l} \text{IMMEVTS} \\ \vee \text{slots} \cong \text{slots}' \wedge (\neg \text{wait}' \vee \neg \text{ok}') \end{array} \right) \end{array} \right) \right)$$

The simplest law to address is (3). To make it hold we only need to add a *wait'* clause in the right place of the *Choice* definition.

$$\text{CSP2}\left(LP \wedge \left(\begin{array}{l} (HP \wedge \text{NOEVTS} \wedge \mathbf{wait}'); \\ \left(\begin{array}{l} \text{IMMEVTS} \\ \vee \text{slots} \cong \text{slots}' \wedge (\neg \text{wait}' \vee \neg \text{ok}') \end{array} \right) \end{array} \right) \right)$$

That way we make sure that the LP (low priority option) behaviour can only be chosen when the HP (high priority option) at the time of resolution is in a waiting state. The second law is addressed by changing *IMMEVTS* into:

$$\exists E \bullet \text{FSTEVT}(E) \wedge E \neq \emptyset \wedge E \subseteq \text{sref}(\text{last}(\text{slots}))$$

That way LP can only perform an event which is refused by HP. Finally we can address the remaining laws - (1) and (4). According to those laws, LP can only be refused when HP is willing to perform an event and the environment is demanding the event to be performed in the first possible time slot. This behaviour is very similar to the waiting option of the prefix operator:

$$\mathbf{CSP1}(ok' \wedge \mathbf{R3}(WTC(c) \wedge wait')) \setminus \{c\} = \textit{Miracle}$$

that gets refused when the environment decides that an event needs to be performed in the first time slot. For that reason we want to copy the mechanism of interaction between hiding and prefix, by not refusing events that are not refused by HP at the time when the choice is being resolved:

$$\mathbf{CSP2}(LP \wedge \left(\begin{array}{l} (HP \wedge \textit{NOEVENTS} \wedge wait'); \\ \left(\begin{array}{l} \mathbf{sref}(\textit{head}(\textit{slots}' \searrow \textit{slots})) \subseteq \mathbf{sref}(\textit{last}(\textit{slots})) \wedge \\ \left(\begin{array}{l} (\exists E \bullet \textit{FSTEVT}(E) \wedge E \neq \emptyset \wedge E \subseteq \textit{sref}(\textit{last}(\textit{slots}))) \\ \vee \textit{slots} \cong \textit{slots}' \wedge (\neg wait' \vee \neg ok') \end{array} \right) \end{array} \right) \end{array} \right)) \end{array} \right)$$

We then obtain the following law:

$$\textit{WeakChoice}(LP, a \xrightarrow{n} HP) \setminus \{a\} = \textit{Miracle}$$

that allows us to prove (1).

3.5 Changes in slotted-Circus

One of the goals, when introducing priority to *slotted-Circus*, was to minimize changes made by the expansion to the semantics of the language. Happily there are only two aspects that need to be addressed: a new healthiness condition and a fix to the semantic definition of hiding.

Because our prioritized choice operator uses refusals sets in a new way and expands their functionality, we have to make sure that information stored in it is properly propagated and whenever a process terminates **CSP4** no longer completely unconstrains its refusals sets. We can do that by introducing a new healthiness condition:

$$\mathbf{PRI}(P) \hat{=} P \wedge (ok \Rightarrow \textit{sref}(\textit{head}(\textit{slots}' \searrow \textit{slots})) \subseteq \textit{sref}(\textit{tail}(\textit{slots})))$$

This ensures that whenever a process is not refusing an event then this information will not be omitted by the following processes. Once we make sure that *Skip* is **PRI** healthy, we ensure that **CSP4** can only reduce refusals.

The problem with the semantics of hiding is that it doesn't satisfy one of the healthiness conditions —**CSP3**. While it was not a problem before, we have to be very careful with refusals when we add priority. For that reason we make sure that refusals of a previous process have no influence at the behaviour of hiding by unconstraining them inside the definition ($\exists s \bullet \textit{slots} \cong s$).

$$A \setminus \textit{hidn} \hat{=} \mathbf{PRI} \circ \mathbf{R3} \left(\begin{array}{l} \exists \mathbf{s}, s' \bullet A[\mathbf{s}, s' / \textit{slots}, \textit{slots}'] \wedge \\ \textit{slots} \cong \mathbf{s} \wedge \textit{hidn} \subseteq \bigcap \textit{srefs}(s' \searrow \mathbf{s}) \\ \textit{slots}' \searrow \textit{slots} = \textit{map}(\textit{shide}(\textit{hidn}))(s' \searrow \mathbf{s}) \wedge \end{array} \right)$$

3.6 Prioritized choice definition

After introducing the new healthiness condition we can finally present a complete definition of prioritized choice:

$$\begin{aligned}
L \overline{\square} H &\cong H \overline{\square} L \\
H \overleftarrow{\square} L &\cong H \wedge L \wedge \text{Stop} \vee \text{Choice}(H, L) \vee \text{WeakChoice}(L, H) \\
\text{WeakChoice}(L, H) &\cong \mathbf{CSP2}(L \wedge \\
&\quad \left(\left(H \wedge \text{NOEVTS} \wedge \text{wait}' \right); \right. \\
&\quad \left. \mathbf{PRI} \left(\begin{array}{l} \exists E \bullet \left(\text{FSTEVTs}(E) \right. \\ \left. \wedge E \neq \emptyset \wedge E \subseteq \text{sref}(\text{tail}(\text{slots})) \right) \right) \right) \\
&\quad \left. \vee \text{slots} \cong \text{slots}' \wedge (\neg \text{wait}' \vee \neg \text{ok}') \right) \right)
\end{aligned}$$

4 Related Work

Work on priority in process algebras can be grouped basically into two main camps: those based on CSP and associated CSP-like languages [Bar89, Fid93, Low93]; and those focussing on labelled transitions systems (the ‘‘CCS school’’ [CH90, CW95, HL98, BG00, CLN07])

In the latter camp, priority has been investigated by adding it as a means for selecting out from many labelled arcs leaving a state, with differences based on how priorities are assigned (local vs. global) [CH90, CW95]. An emphasis has been on characterising relevant bisimulations, congruences and corresponding axiomatizations of priority in these settings [HL98, BG00, CLN07]. Despite the extensive work done on priority in a CCS setting however, it is not the case that priority in process algebras has ‘‘been done’’. When presenting our operational semantics of Handel-C [BW05] we pointed out that we had two notions of priority: one that of the `pralt` construct in the language, the other associated with the LTS we constructed, neither of which corresponded to any of the priority models described in the CCS literature.

Of interest is the BIP system developed by Sifakis and colleagues [BBS06] that views systems as components built in three layers: an LTS with ports to communicate with the outside world; a notion of interaction as a set of ports from different LTSs; and using a priority scheme to select among enabled interactions. This system at an abstract level, is quite similar to both the notion of prioritised choice in Handel-C and the slots concepts that we have formalised in *slotted-Circus*, and it may prove fruitful to investigate the relationship more closely.

In the CSP camp, more emphasis has been placed on the use of denotational semantics. In terms of priority, early work on the use of priority in implementations said very little about its semantics except that prioritised choice was a refinement of external choice. Interesting early exceptions were work giving occam an operation semantics [Bar89, Cam89]. More substantial work on priority in a denotational setting was presented by Colin Fidge [Fid93] introducing the notion of *preferences*. At the same time, Gavin Lowe characterised both probabilistic and prioritised CSP as refinements of Timed CSP, and established linkages between probability and priority [Low93].

5 Future Work

Also worthy of exploration are the details of the behaviour of the Galois links [HH98, Chp 4] between *slotted-Circus*, with and without priority, and between those and standard *Circus*. These details will provide a framework for a comprehensive refinement calculus linking all these reactive theories together. The goal is a scheme whereby *Circus* is a specification language and *slotted-Circus* is a refinement stage, on the way to a hardware implementation, captured in prioritised *slotted-Circus*. We plan to perform some case studies, looking at hardware interfaces for flash memory, as well as exploring wireless network protocols for which our prioritised model seems surprisingly well-suited (we can use priority to capture collision detection).

6 Conclusions

A denotational semantics for prioritised *slotted-Circus* has been presented, and we have shown that prioritized choice in *slotted-Circus* and its laws fit the `pralt` construct in Handel-C. Of particular interest has been the introduction of the “clock-tick after communication” constraint in order to get a sensible theory. We have also identified close linkages between our work and that on prioritised CSP, and extensions to timed CSP.

It is still an open question what is the difference (if any), between the CTA or MSA history models, when only timed communication is permitted. Another open problem is the existence of healthiness conditions, implying that communication takes time. So far our research leads us to suggest that it will be a special case of prefix closure, but no prefix closure healthiness condition has yet been formalised in UTP.

Acknowledgements We would like to thank Jim Woodcock and his colleagues for many fruitful discussions on various aspects of this work. This work was also partially supported by Lero, the Irish Software Engineering Research Centre, SFI grant 03/CE2/I303_1.

References

- [Bar89] Geoff Barrett. The semantics of priority and fairness in *occam*. In Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *MFPS*, volume 442 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 1989.
- [BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), 11-15 September 2006, Pune, India*, pages 3–12. IEEE Computer Society, 2006.
- [BG00] Mario Bravetti and Roberto Gorrieri. A complete axiomatization for observational congruence of prioritized finite-state behaviors. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 744–755. Springer, 2000.

- [BG09] Andrew Butterfield and Paweł Gancarski. slotted-Circus: A generic UTP framework for discretely-timed *circus*. Technical Report TCD-CS-09-32, School of Computer Science & Statistic Trinity College Dublin, Trinity College, Dublin 2, Ireland, July 2009. <https://www.cs.tcd.ie/publications/tech-reports/reports.09/TCD-CS-2009-32.pdf>.
- [BSW07] Andrew Butterfield, Adnan Sherif, and Jim Woodcock. Slotted-*circus*: A UTP-family of reactive theories. In Jim Davies and Jeremy Gibbons, editors, *IFM 2007*, volume 4591 of *LNCS*, pages 75–97. Springer, 2007.
- [BW05] Andrew Butterfield and Jim Woodcock. *pralt* in Handel-C: an operational semantics. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(3):248–267, June 2005.
- [Cam89] J. Camilleri. An operational semantics for OCCAM. *International Journal of Parallel Programming*, 18(5):149–167, October 1989.
- [Cel02] Celoxica Ltd. *Handel-C Language Reference Manual, v3.0*, 2002. URL: www.celoxica.com.
- [CH90] Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. *Inf. Comput.*, 87(1/2):58–77, July/August 1990.
- [CLN07] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority and abstraction in process algebra. *Inf. Comput.*, 205(9):1426–1458, 2007.
- [CW95] Juanito Camilleri and Glynn Winskel. CCS with priority choice. *Inf. Comput.*, 116(1):26–37, January 1995.
- [Fid93] C. J. Fidge. A formal definition of priority in CSP. *ACM Transactions on Programming Languages and Systems*, 15(4):681–705, September 1993.
- [GB09] Paweł Gancarski and Andrew Butterfield. The denotational semantics of *slotted-circus*. In Ana Cavalcanti and Dennis Dams, editors, *FM2009: Formal Methods*, volume 5850 of *LNCS*, pages 451–466. Springer, 2009.
- [HH98] C. A. R. Hoare and Jifeng He. *Unifying Theories of Programming*. Series in Computer Science. Prentice Hall, 1998. <http://www.unifyingtheories.org>.
- [HL98] H. Hermanns and M. Lohrey. Priority and maximal progress are completely axiomatisable. *Lecture Notes in Computer Science*, 1466:237–252, 1998.
- [Hoa85a] C. A. R. Hoare. *Communicating Sequential Processes*. Intl. Series in Computer Science. Prentice Hall, 1985.
- [Hoa85b] C. A. R. Hoare. Programs are predicates. In *Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages*, pages 141–155. Prentice-Hall, Inc., 1985.
- [Low93] G. Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, Oxford University, 1993.
- [OCW09] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. A UTP semantics for circus. *Formal Asp. Comput.*, 21(1-2):3–32, 2009.
- [Ros84] A. W. Roscoe. Denotational semantics for occam. In Stephen D. Brookes, A. W. Roscoe, and Glynn Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 306–329. Springer, 1984.
- [Sch00] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Wiley, 2000.
- [SH02] Adnan Sherif and Jifeng He. Towards a time model for circus. In Chris George and Huaikou Miao, editors, *ICFEM*, volume 2495 of *Lecture Notes in Computer Science*, pages 613–624. Springer, 2002.
- [She06] Adnan Sherif. *A Framework for Specification and Validation of Real Time Systems using Circus Action*. Ph.d. thesis, Universidade Federale de Pernambuco, Recife, Brazil, Jan 2006.