Formal specification of SCJ
icecap-implementation
version 1.0, July 2015, 17:13

by
Leo Freitas\textsuperscript{1}, Ana Cavalcanti\textsuperscript{2}, Andy Wellings\textsuperscript{2}

\textsuperscript{1} Newcastle University
School of Computing Science
Centre for Software Reliability (CSR)

\textsuperscript{2} University of York
Department of Computer Science
High Integrity Systems Engineering (HISE) Group
# Contents

1 Basic definitions 1
   1.1 Basic types ............................................. 1

2 icecap Virtual Machine 5
   2.1 VM memory ................................................. 5
   2.2 VM processes .............................................. 5
      2.2.1 VM process operations .............................. 6
      2.2.2 VM process operations preconditions .............. 6

3 Abstract scheduler 8
   3.1 Abstract scheduler ...................................... 8
      3.1.1 Abstract scheduler operations ................. 8
   3.2 Abstract scheduler properties ..................... 11
   3.3 Abstract scheduler operations .................... 11
   3.4 Abstract scheduler operations preconditions .... 11

4 SCJ concurrency primitives 15
   4.1 Event handlers set ..................................... 15
   4.2 Priority scheduler of Event handlers ............. 17
   4.3 Priority scheduler operations preconditions .... 21

5 SCJ priority scheduler implementation in Circus 25
   5.0.1 Priority scheduler channels ............... 25
   5.1 Priority Scheduler channels .......................... 26
   5.2 Clock interrupt handler channels ............... 26
   5.3 SCJ priority scheduler bridge ..................... 27
   5.4 SCJ priority scheduler process ..................... 28
      5.4.1 SCJ process abstraction .......................... 28
      5.4.2 Priority frame containing schedulable queues 28
      5.4.3 Priority scheduler state ......................... 31
      5.4.4 Various state operations ....................... 33
      5.4.5 SCJ clock interrupt handler API .............. 34
      5.4.6 SCJ run-time environment API .................. 34
      5.4.7 SCJ user API actions ................................ 35
      5.4.8 Priority scheduler API network ................. 36

6 SCJ framework implementation in Circus 40
   6.1 Launcher ............................................... 40
   6.2 Clock interrupt handler ......................... 41
   6.3 Priority Scheduler (Level1) ...................... 43
   6.4 Mission ................................................ 44
   6.5 Event Handlers ........................................... 46
   6.6 Mission Sequencer ..................................... 47
   6.7 NEW SECTION ........................................... 49
A Extended Z toolkit for SCJ
A.1 Sets ................................................. 51
  A.1.1 Definitions .................................. 51
  A.1.2 Lemmas on sets ................................ 51
A.2 Relations ........................................... 53
A.3 Function spaces ..................................... 54
A.4 Injective functions .................................. 55
A.5 Finiteness .......................................... 57
A.6 Sequences ........................................... 58
  A.6.1 Lemmas on sequences ............................ 58
  A.6.2 Lemmas on Sequence concatenation ............ 59
  A.6.3 Lemmas on Sequence Decomposition ............ 60
  A.6.4 Sequence manipulation .......................... 62
  A.6.5 Sequence mapping ................................ 62
B Proof scripts .......................................... 64
B.1 Z Section Basic data types .......................... 64
B.2 Z Section VM processes operations preconditions .... 66
B.3 Abstract scheduler .................................... 67
B.4 Z Section Abstract scheduler properties .......... 67
B.5 Z Section Abstract scheduler operations preconditions .. 67
B.6 Z Section Event handlers proofs ..................... 76
B.7 Z Section Priority scheduler proofs ................. 79
  B.7.1 Priority scheduler domain checks ............... 90
B.8 Complete summary of proofs .......................... 90
C CSPM translation of icecap Circus .................. 93
C.1 Version without SCJProces explicitly given .......... 93
C.2 Version with SCJProces explicitly given .......... 99
List of Figures
# List of Tables

1.1 Summary of Z declarations for Chapter 1. .............................. 4
2.1 Summary of Z declarations for Chapter 2. .............................. 7
3.1 Summary of Z declarations for Chapter 3. .............................. 14
4.1 Summary of Z declarations for Chapter 4. .............................. 24
5.1 Summary of Z declarations for Chapter 5. .............................. 39
5.2 Summary of *Circus* declarations for Chapter 5. ..................... 39
6.1 Summary of Z declarations for Chapter 6. .............................. 50
6.2 Summary of *Circus* declarations for Chapter 6. ..................... 50
A.1 Summary of Z declarations for Chapter A. ............................ 63
B.1 Summary of Z declarations for Chapter B. ............................ 92
B.2 Summary of *Circus* declarations for Chapter B. .................... 92
Declarations summary

Abbreviations
Given sets
Free types
Generic definitions
Axiomatic definitions
Schemas
Z Sections

Z Proof Sections

B.1.1 Basic data types (see 1.1, p. 1) .................. 64
B.2.1 VM processes operations preconditions (see 2.2.2, p. 6) .................. 66
B.4.1 Abstract scheduler properties (see 3.2, p. 11) .................. 67
B.5.1 Abstract scheduler operations preconditions (see 3.4, p. 11) .................. 67
B.6.1 Event handlers proofs (see 4.1, p. 15) .................. 76
B.7.1 Priority scheduler proofs (see 4.2, p. 17) .................. 79
Theorems summary

Z Toolkit extension

A.1 Set difference distribute to the right on set difference ................. 51
A.2 Set difference equivalence modulo set intersection ....................... 52
A.3 Singleton set union absorbs set difference ............................... 52
A.4 Set union absorbs set intersection ........................................ 52
A.5 Set union exchange to the right on set difference ......................... 52
A.6 Set with an element is not empty ........................................... 52
A.7 Expose $\leftrightarrow$ property .............................................. 54
A.8 $R$ within its own domain .................................................. 54
A.9 $R$ within its own range .................................................... 54
A.10 $R$ within its own domain and range ....................................... 54
A.11 Partial function point is an element ....................................... 54
A.12 Expose $\mapsto\rightarrow$ property ........................................... 54
A.13 Non empty PFun has non empty dom ...................................... 55
A.14 Dom element is specific $\leftrightarrow$ element ............................. 55
A.15 Member of homogeneous $\Rightarrow$ forms no loop ......................... 55
A.16 $\Rightarrow$ non-immediate member .......................................... 55
A.17 $\Rightarrow$ containment ...................................................... 55
A.18 $\Rightarrow$ pointwise equivalence ............................................ 55
A.19 $\Rightarrow$ dom partitions over $<$ ........................................ 55
A.20 Expose $\mapsto\Rightarrow$ property ......................................... 56
A.21 $\mapsto$ dom element ran $\Rightarrow$ equivalence ............................. 56
A.22 $\mapsto$ ran element dom $\Rightarrow$ equivalence .................. 56
A.23 $\mapsto$ dom element expansion ........................................... 56
A.24 $\mapsto$ ran element expansion ............................................ 56
A.25 $\mapsto\leadsto$ application ................................................... 57
A.26 $\mapsto$ point is $\mapsto$ element ........................................ 57
A.27 Distinct $\mapsto$ point is not shared in $\mapsto$ .............................. 57
A.28 $\mapsto$ following application is not $\mapsto$ member ......................... 57
A.29 Finite bijection subset measurement ..................................... 57
A.30 Non-maximal cardinality equivalence .................................... 57
A.31 Non-maximal domain equivalence ......................................... 57
A.32 Expose seq property ....................................................... 58
A.33 Expose iseq property ...................................................... 58
A.34 Non-empty sequence has strictly positive size ......................... 59
A.35 Alternative for dom $s$ when involving contraction ...................... 59
A.36 Sequence domain non-maximal cardinality .............................. 59
A.37 Sequence domain non-maximal card. equivalence ....................... 59
A.38 Sequence domain non-maximal card. equivalence ....................... 59
A.39 Sequence $\wedge$ last of left seq appl ................................. 59
A.40 Sequence $\wedge$ head of right seq appl .................................. 59
A.41 Singleton sequence membership ......................................... 59
A.42 Singleton sequence element membership ................................ 59
A.43 last-$\wedge$ ( $\wedge$ ) left side equivalence ................................ 60
A.44 head-$\wedge$ ( $\wedge$ ) right side equivalence ............................. 60
A.45 Concatenation size equivalence .......................................... 60
A.46 dom of sequence concatenation ........................................... 60
A.47 tail element is in seq domain ............................................. 60
A.48 Domain of non-empty sequence’s tail ................................... 61
A.49 Sequence tail element in range of tail ................................ 61
A.50 Sequence tail is closed under seq range .............................. 61
A.51 Non singleton sequence cardinality .................................... 61
A.52 Non-singleton sequences have mid-points ............................ 61
A.53 sequence right-inductive decomp. prop. ............................... 61
A.54 Non-empty sequence right-inductive decomposition .............. 61
A.55 Range of sequence filtering ............................................. 62
A.56 Sequence mapping remains sequence when within the map ...... 62

Lemmas
A.1 \[\mapsto \mapsto \] element is part of its inverse .......................... 55
A.2 \[\mapsto \mapsto \] result uniqueness weakening rule ....................... 56
A.3 \[\mapsto \mapsto \] element is part of its inverse ........................... 56
A.4 \[\mapsto \mapsto \] ran element \sim in \[\mapsto \mapsto \] ............................. 56
A.5 Z/Eves induction layout theorem for non singleton induc. ......... 61

Theorems
Axiomatic consistency lemmas
Z/Eves housekeeping

Assumption Rules

Rewriting Rules

A.1 Expanding $\bigcup$ without knowing the type of $x$ ........................................... 52
A.2 Expanding the (non-maximal) type of $x$ from $R$ .................................................. 53
A.3 Expanding the (non-maximal) type of $y$ from $R$ .................................................. 53
A.4 Expanding the (non-maximal) type of an element of $R$ ........................................ 53
A.5 Packing pair of $R$ in its domain ................................................................. 53
A.6 Packing pair of $R$ in its range ................................................................. 53
A.7 Extracting dom $R$ subtype ........................................................................ 53
A.8 Extracting dom $R$ subtype ........................................................................ 53
A.9 Inferring finite sets are subset of infinite sets .................................................. 57
A.10 Expanding the (non-maximal) type of a sequence element .............................. 58
A.11 Expanding the (non-maximal) type of range type of sequence element .......... 58
A.12 $\exists$ max type ............................................................................................ 60
A.13 $\exists$ injective sequence to right ...................................................................... 60
A.14 tail preserves sequence injectiveness ............................................................... 61

Forward Rules
Abstract

Bla bla bla
Chapter 1

Basic definitions

section basic_defs parents standard_toolkit

1.1 Basic types

Taken from the chain datatype [2], we reuse the definition of process identifiers. Processes are represented by a non-empty range of process identifiers up to a maxpid value, where the nullpid is represented by either zero or anything beyond the maximum. [Circus tools off]

\[ \exists \text{maxpid} : \mathbb{N} \mid \text{true} \bullet \text{true} \]

[Circus tools on]

\[ \mid \text{maxpid} : \mathbb{N} \]

[Circus tools off]

\[ \exists \text{PID} : \mathbb{P}(1 \ldots \maxpid) \mid \text{true} \bullet \text{true} \]

[Circus tools on]

We model VM processes abstractly as simply PIDs. If needed, we may add other constraints from within vm.Process like time or memory.

\[ \text{PID} \equiv 1 \ldots \maxpid \]

FIX ME: Circus parser doesn’t recognise Z/Eves ’disabled’ or ’label’ flags, neither theorems or refs. Perhaps make Circus parser an extension of Z/Eves?

[Circus tools off]

\[ \exists \text{PIDNotEmpty} \]

\[ \neg \text{PID} = \{\} \]

\[ \exists \text{MinPIDValue} \]

\[ 1 \in \text{PID} \]
**theorem** grule gMaxpidMaxType
maxpid ∈ ℤ

**theorem** rule lMaxpidIsPID
maxpid ∈ PID

**theorem** basic_defs_axiom2_vc_fsb_axiom
∃ nullpid : ℕ | true • ∀ p : PID • p < nullpid

[Circus tools on]

FIX ME: Could make the axiom as maxpid < nullpid and then prove this as a property?

nullpid : ℕ

| p : PID • p < nullpid

[Circus tools off]

**theorem** rule lNullpidBound
maxpid < nullpid

**theorem** rule lNullPIDDisjoint
∀ p : PID • ¬ p = nullpid

**theorem** GPID_vc_fsb_horiz_def
∃ GPID : ℙ(PID ∪ {nullpid}) | true • true

[Circus tools on]

Another range involve the PID range including nullpid. In certain implementations, idle processes could be given specific values say 0 (hence making nullpid outside PID), or some value within PID range itself.

GPID == PID ∪ {nullpid}

[Circus tools off]

**theorem** grule gPIDMaxType
PID ∈ ℙ ℤ

**theorem** grule gGPIDMaxType
GPID ∈ ℙ ℤ

**theorem** rule lNullsGPID
nullpid ∈ GPID
\textbf{theorem} rule lNullIsNotPID
\[ \neg \text{nullpid} \in \text{PID} \]

\textbf{theorem} rule lPIDIsGPID
\[ \forall x : \text{PID} \bullet x \in \text{GPID} \]

\textbf{theorem} rule lMinPIDIsGPID
\[ 1 \in \text{GPID} \]

\textbf{theorem} rule lMaxPIDIsGPID
\[ \text{maxpid} \in \text{GPID} \]

\textbf{theorem} rule lNonNullGPIDIsPID
\[ \forall p : \text{GPID} \ | \ \neg p = \text{nullpid} \bullet p \in \text{PID} \]

\textbf{[Circus tools on]}
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>23</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

Table 1.1: Summary of Z declarations for Chapter 1.
Chapter 2

icecap Virtual Machine

The icecap VM contains key abstractions for the underlying OS concurrent resources such as memory, process IDs, addressing spaces, and so on. For our model, we care about the pool of processes available, as well as their memory. This is because most of the SCJ infrastructure makes use of these two resources for scheduling of processes and for memory areas management. The data type for the VM, scheduler, and SCJ managed events is rather similar, given their intricate relationship. This will entail a refinement of the composed data structures into parallel Circus processes [1].

section vm parents basic_defs, scj_toolkit

2.1 VM memory

VM Memory is used by javax.realtime.MemoryArea, which is then extended by javax.safetycritical.ManagedMemory and ultimately MissionMemory. This is catered by the SCJ library, so will be modelled at a separate time. It is the case that elements using VM memory are those that wrap around a vm.Process as well.

```
VMMemory
base, size, free : N
```

TODO LF: VM memory operations later on whenever we model the SCJ library classes like MissionSequencer, which uses MissionMemory.

2.2 VM processes

We add some error codes for the VM operations

```
VMErrror ::= VMOkay | VMOOutOfMemory | VMOOutOfProcess | VMUnknownProcess
```

FIX ME: list of exceptions that are modelled but not present in the code please! Ex. VMOOutOfProcess is a key one reflecting the available/free resources that isn’t caught anywhere in the code (that we could see / find).
TODO: add to the code a more structured way of annotating the code

The abstraction for the VM contains the used and free processes under the VM’s execution. This is a simpler view of the scheduler.

```
VM
createdVM, usedVM, freeVM : P PID
(createdVM, usedVM, freeVM) partition PID
```
2.2.1 VM process operations

Whenever a VM process is created, we record that by adding some free process identifier to the set of known processes to the VM. In Java, these will be the `vm.Process` objects in memory.

After creation, the VM processes are initialised upon creation of SCJ processes. Another initialisation point is the `vm.Process` the `ClockInterruptHandler` wraps to handle periodic preemption. That is, the running (C) VM interacts with the Java infrastructure via this handler’s line of execution, which in turn calls upon the `vm.Scheduler` for the scheduling of appropriate `vm.Process`. In SCJ, that is the job of either of its schedulers (e.g. `CyclicExecutive`, `PriorityScheduler`). Once running, `vm.Process` “forever” goes to the running (or used) set, hence no operation to free them exists. That’s because of a peculiar behaviour of its execution code, which enters an infinite loop at the end of its life (see `vm.Process.ProcessExecutioner.run():L73-75`).

As usual in Z, to complete the operations, we add their corresponding error cases. This makes both operations total: they always succeed execution, or deterministically fail.

2.2.2 VM process operations preconditions

To ensure they are total, we declare signatures with no extra precondition for both operations.

The preconditions for the total operations are proved next. [Circus tools off]

[Circus tools on]

FIX ME: Unfortunately, because STDZ definition of theorem doesn’t allow for a name, we couldn’t keep theorems being processed within the Circus tools. We tried to redefine `vdash` as just hard/soft space, but that doesn’t work either because theorems can’t have names.

1This Z pattern is pervasive in CZT’s VCG for Z.
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16</strong></td>
<td><strong>39</strong></td>
</tr>
</tbody>
</table>

Table 2.1: Summary of Z declarations for Chapter 2.
Chapter 3

Abstract scheduler

section abstract scheduler parents basic_defs, scj_toolkit

3.1 Abstract scheduler

The abstract scheduler specification is an adaptation to the one found in [4, Ch. 23]. A scheduler contains three separate sets of process of interest: Ready, Blocked (or sleeping), and Free (or available for allocation). We uniquely name variables to avoid confusion of names when composing this abstract scheduler with the VM (see Chapter 2) or the SCJ priority scheduler (see Chapter 4).

There is also a notion of the currently running process, which might be nullpid (i.e. current ∈ GPID). The invariant ensures that all valid (non-null) process identifiers are uniquely used across the various schedulable sets used. Moreover, that these PIDs are all PIDs known: these schedulable sets form a partition of PIDs, hence all PIDs must be uniquely determined as either readyAS, blockedAS, freeAS, or (non-null) current.

\begin{verbatim}
AScheduler
current : GPID
readyAS, blockedAS, freeAS : \notin PID
\langle \{ current \} \setminus \{ nullpid \}, readyAS, blockedAS, freeAS \rangle partition PID
\end{verbatim}

Initialisation is trivial: each set is empty, but the one with freeAS PIDs, where the current process is nullpid.

\begin{verbatim}
ASchedulerInit
AScheduler' =
current' = nullpid
readyAS' = \emptyset
blockedAS' = \emptyset
freeAS' = PID
\end{verbatim}

3.1.1 Abstract scheduler operations

Scheduler’s operations are in the table below; state elements not mentioned remain constant.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>moves a freeAS PID into readyAS</td>
<td>freeAS ≠ \emptyset</td>
</tr>
<tr>
<td>Dispatch</td>
<td>takes any readyAS as current</td>
<td>current = nullpid ∧ readyAS ≠ \emptyset</td>
</tr>
<tr>
<td>Block</td>
<td>puts current in blockedAS and makes current null</td>
<td>current ≠ nullpid</td>
</tr>
<tr>
<td>Timeout</td>
<td>puts current in readyAS and makes current null</td>
<td>current ≠ nullpid</td>
</tr>
<tr>
<td>Wakeup</td>
<td>moves a blockedAS PID into ready</td>
<td>blockedAS ≠ \emptyset</td>
</tr>
<tr>
<td>Destroy</td>
<td>moves either current, ready, blocked PID to free</td>
<td>current ≠ nullpid ∨ readyAS, blockedAS ≠ \emptyset</td>
</tr>
</tbody>
</table>

Create, Dispatch, Block and Timeout nondeterministically picks an element from the appropriate set and outputs it (as p!), whereas Wakeup and Destroy picks an element given as input (p?) from the appropriate set.
Create

\[ \Delta AScheduler \]
\[ p! : PID \]

\[
\begin{align*}
\text{freeAS} \neq \emptyset \\
\text{current}' &= \text{current} \\
\text{readyAS}' &= \text{readyAS} \cup \{ p! \} \\
\text{blockedAS}' &= \text{blockedAS} \\
\text{freeAS}' &= \text{freeAS} \setminus \{ p! \} \\
p! &\in \text{freeAS}
\end{align*}
\]

Dispatch

\[ \Delta AScheduler \]
\[ p! : PID \]

\[
\begin{align*}
\text{current} &= \text{nullpid} \\
\text{readyAS} &= \emptyset \\
\text{current}' &\in \text{readyAS} \\
\text{readyAS}' &= \text{readyAS} \setminus \{ \text{current}' \} \\
\text{blockedAS}' &= \text{blockedAS} \\
\text{freeAS}' &= \text{freeAS} \\
p! &= \text{current}'
\end{align*}
\]

Block

\[ \Delta AScheduler \]
\[ p! : PID \]

\[
\begin{align*}
\text{current} &= \text{nullpid} \\
\text{current}' &= \text{nullpid} \\
\text{readyAS}' &= \text{readyAS} \\
\text{blockedAS}' &= \text{blockedAS} \cup \{ \text{current} \} \\
\text{freeAS}' &= \text{freeAS} \\
p! &= \text{current}
\end{align*}
\]

TimeOut

\[ \Delta AScheduler \]
\[ p! : PID \]

\[
\begin{align*}
\text{current} &= \text{nullpid} \\
\text{current}' &= \text{nullpid} \\
\text{readyAS}' &= \text{readyAS} \cup \{ \text{current} \} \\
\text{blockedAS}' &= \text{blockedAS} \\
\text{freeAS}' &= \text{freeAS} \\
p! &= \text{current}
\end{align*}
\]

WakeUp

\[ \Delta AScheduler \]
\[ p? : PID \]

\[
\begin{align*}
p? &\in \text{blockedAS} \\
\text{current}' &= \text{current} \\
\text{readyAS}' &= \text{readyAS} \cup \{ p? \} \\
\text{blockedAS}' &= \text{blockedAS} \setminus \{ p? \} \\
\text{freeAS}' &= \text{freeAS}
\end{align*}
\]

Destroy is divided in its three valid disjuncts cases.
\[ A\text{DestroyCurrent} \]
\[
\Delta A\text{Scheduler} \\
p? : \text{PID} \\
p? = \text{current} \\
current' = \text{nullpid} \\
\text{readyAS}' = \text{readyAS} \\
\text{blockedAS}' = \text{blockedAS} \\
\text{freeAS}' = \text{freeAS} \cup \{ p? \}
\]

\[ A\text{DestroyReady} \]
\[
\Delta A\text{Scheduler} \\
p? : \text{PID} \\
p? \in \text{readyAS} \\
current' = \text{current} \\
\text{readyAS}' = \text{readyAS} \setminus \{ p? \} \\
\text{blockedAS}' = \text{blockedAS} \\
\text{freeAS}' = \text{freeAS} \cup \{ p? \}
\]

\[ A\text{DestroyBlocked} \]
\[
\Delta A\text{Scheduler} \\
p? : \text{PID} \\
p? \in \text{blockedAS} \\
current' = \text{current} \\
\text{readyAS}' = \text{readyAS} \\
\text{blockedAS}' = \text{blockedAS} \setminus \{ p? \} \\
\text{freeAS}' = \text{freeAS} \cup \{ p? \}
\]

and is composed by disjunction as usual.

\[ A\text{Destroy0} == A\text{DestroyCurrent} \lor A\text{DestroyReady} \lor A\text{DestroyBlocked} \]

Error cases are modelled with an extra message determining the error conditions.

\[ \text{ASchedMsg} ::\!=\! \text{AS}_\text{OKAY} \mid \text{AS}_\text{DISPATCH\_ERR} \mid \text{AS}_\text{TIMEOUT\_ERR} \mid \text{AS}_\text{BLOCK\_ERR} \mid \text{AS}_\text{WAKEUP\_ERR} \mid \text{AS}_\text{CREATE\_ERR} \mid \text{AS}_\text{DESTROY\_ERR} \]

\[ \text{ASchedOkay} \quad == \quad [ \text{aserr}! : \text{ASchedMsg} \mid \text{aserr}! = \text{AS}_\text{OKAY} ] \]

\[ \text{ASchedErr} \quad == \quad [ \exists A\text{Scheduler}; \text{aserr}! : \text{ASchedMsg} ] \]

\[ \text{ACreateErr} \quad == \quad [ \text{ASchedErr} \mid \text{aserr}! = \text{AS}_\text{CREATE\_ERR} \land \text{freeAS} = \emptyset ] \]

\[ \text{ADispatchErr} \quad == \quad [ \text{ASchedErr} \mid \text{aserr}! = \text{AS}_\text{DISPATCH\_ERR} \land (\text{current} \neq \text{nullpid} \lor \text{readyAS} = \emptyset) ] \]

\[ \text{ABlockErr} \quad == \quad [ \text{ASchedErr} \mid \text{aserr}! = \text{AS}_\text{BLOCK\_ERR} \land \text{current} = \text{nullpid} ] \]

\[ \text{ATimeOutErr} \quad == \quad [ \text{ASchedErr} \mid \text{aserr}! = \text{AS}_\text{TIMEOUT\_ERR} \land \text{current} = \text{nullpid} ] \]

\[ \text{AWakeUpErr} \quad == \quad [ \text{ASchedErr}; p? : \text{PID} \mid \text{aserr}! = \text{AS}_\text{WAKEUP\_ERR} \land p? \notin \text{blockedAS} ] \]

\[ \text{ADestroyErr} \quad == \quad [ \text{ASchedErr}; p? : \text{PID} \mid \text{aserr}! = \text{AS}_\text{DESTROY\_ERR} \land p? \in \text{freeAS} ] \]

The complete (total) operations are defined next by composing error and successful cases.

\[ \text{ACreate} \quad == \quad (\text{ACreate0} \land \text{ASchedOkay}) \lor \text{ACreateErr} \]

\[ \text{ADispatch} \quad == \quad (\text{ADispatch0} \land \text{ASchedOkay}) \lor \text{ADispatchErr} \]

\[ \text{ABlock} \quad == \quad (\text{ABlock0} \land \text{ASchedOkay}) \lor \text{ABlockErr} \]

\[ \text{ATimeOut} \quad == \quad (\text{ATimeOut0} \land \text{ASchedOkay}) \lor \text{ATimeOutErr} \]

\[ \text{AWakeUp} \quad == \quad (\text{AWakeUp0} \land \text{ASchedOkay}) \lor \text{AWakeUpErr} \]

\[ \text{ADestroy} \quad == \quad (\text{ADestroy0} \land \text{ASchedOkay}) \lor \text{ADestroyErr} \]
3.2 Abstract scheduler properties

This theorem ensures that all resources are accounted for. That is, if there is a process identifier, its status must be known. Otherwise, it must be a unallocated resource (i.e., nullpid). This is important, for instance, in the proof that ADestroy is total.

\[
\text{theorem disabled rule lScheduledPIDDissjoint}
\forall AScheduler; p? : PID | \neg p? \in \text{blockedAS} \land \neg p? \in \text{readyAS} \land p? = \text{current} \Rightarrow p? \in \text{freeAS}
\]

3.3 Abstract scheduler operations

This is useful to the proof of ADestroy being total.

3.4 Abstract scheduler operations preconditions

For each operation, we define signature schemas. They contain all that can be assumed (e.g. before state and inputs) together with calculated preconditions (e.g. the minimal predicate that will make the pre operator true) for the corresponding operations.

\[
\begin{align*}
ACreate0Sig & \equiv [AScheduler | \text{freeAS} \neq \emptyset] \\
ADispatch0Sig & \equiv [AScheduler | \text{current} = \text{nullpid} \land \text{readyAS} \neq \emptyset] \\
ATimeOut0Sig & \equiv [AScheduler | \text{current} \neq \text{nullpid}] \\
ABlock0Sig & \equiv ATimeOut0Sig \\
AWakeUp0Sig & \equiv [AScheduler; p? : PID | p? \in \text{blockedAS}] \\
ADestroyCurrentSig & \equiv [AScheduler; p? : PID | p? = \text{current}] \\
ADestroyReadySig & \equiv [AScheduler; p? : PID | p? \in \text{readyAS}] \\
ADestroyBlockedSig & \equiv [AScheduler; p? : PID | p? \in \text{blockedAS}] \\
ACreateErrSig & \equiv [AScheduler | \text{freeAS} = \emptyset] \\
ADispatchErrSig & \equiv [AScheduler | (\text{current} \neq \text{nullpid} \lor \text{readyAS} = \emptyset)] \\
ABlockErrSig & \equiv [AScheduler | \text{current} = \text{nullpid}] \\
ATimeOutErrSig & \equiv ABlockErrSig \\
AWakeUpErrSig & \equiv [AScheduler; p? : PID | p? \notin \text{blockedAS}] \\
ADestroyErrSig & \equiv [AScheduler; p? : PID | p? \in \text{freeAS}] \\
ASchedTotalSig & \equiv [AScheduler; p? : PID | \text{true}]
\end{align*}
\]

Next, we declare each of the precondition theorems to be proved for the abstract scheduler’s consistency, where initialisation is slightly different, if achieving the same end. [Circus tools off]

\[
\text{theorem ASchedulerInitFSB}
\exists AScheduler’ \Rightarrow ASchedulerInit
\]

\[
\text{theorem ADispatch0FSB}
\forall ADispatch0Sig \Rightarrow \text{pre ADispatch0}
\]

\[
\text{theorem ATimeOut0FSB}
\forall ATimeOut0Sig \Rightarrow \text{pre ATimeOut0}
\]

\[
\text{theorem ABlock0FSB}
\forall ABlock0Sig \Rightarrow \text{pre ABlock0}
\]

\[
\text{theorem AWakeUp0FSB}
\forall AWakeUp0Sig \Rightarrow \text{pre AWakeUp0}
\]
theorem ACreate0FSB
\[ \forall ACreate0Sig \cdot \text{pre } ACreate0 \]

theorem ADestroyCurrentFSB
\[ \forall ADestroyCurrentSig \cdot \text{pre } ADestroyCurrent \]

theorem ADestroyReadyFSB
\[ \forall ADestroyReadySig \cdot \text{pre } ADestroyReady \]

theorem ADestroyBlockedFSB
\[ \forall ADestroyBlockedSig \cdot \text{pre } ADestroyBlocked \]

theorem ADispatchErrFSB
\[ \forall ADispatchErrSig \cdot \text{pre } ADispatchErr \]

theorem ATimeOutErrFSB
\[ \forall ATimeOutErrSig \cdot \text{pre } ATimeOutErr \]

theorem ABlockErrFSB
\[ \forall ABlockErrSig \cdot \text{pre } ABlockErr \]

theorem AWakeUpErrFSB
\[ \forall AWakeUpErrSig \cdot \text{pre } AWakeUpErr \]

theorem ACreateErrFSB
\[ \forall ACreateErrSig \cdot \text{pre } ACreateErr \]

theorem ADestroyErrFSB
\[ \forall ADestroyErrSig \cdot \text{pre } ADestroyErr \]

theorem ADispatchIsTotal
\[ \forall AScheduler \cdot \text{pre } ADispatch \]

theorem ATimeOutIsTotal
\[ \forall AScheduler \cdot \text{pre } ATimeOut \]

theorem ABlockIsTotal
\[ \forall AScheduler \cdot \text{pre } ABlock \]
\textbf{theorem} \textit{AWakeUpIsTotal}
\[ \forall \text{ASchedTotalSig} \bullet \text{pre} \text{ AWakeUp} \]

\textbf{theorem} \textit{ACreateIsTotal}
\[ \forall \text{AScheduler} \bullet \text{pre} \text{ ACreate} \]

\textbf{theorem} \textit{ADestroyIsTotal}
\[ \forall \text{ASchedTotalSig} \bullet \text{pre} \text{ ADestroy} \]

[Circus tools on]
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>34</td>
<td>46</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>22</td>
<td>43</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>66</strong></td>
<td><strong>105</strong></td>
</tr>
</tbody>
</table>

Table 3.1: Summary of Z declarations for Chapter 3.
Chapter 4

SCJ concurrency primitives

In this chapter we model the SCJ concurrency primitives as an abstract model of its scheduler. This is an implementation of prioritised coroutines used as threads of execution within the SCJ icecap implementation.

section scj_process parents vm, ascheduler

4.1 Event handlers set

SCJ has various types of managed event handler [3, p.91], as described by the next free type. In icecap, ManagedEventHandler is an abstract class that implements ManagedSchedulable interface, yet it does not derive from javax.realtime as in SCJ’s JSR, but is an abstract base class instead. They characterise the kind of execution principle the handler is to have. Ultimately, it entails an underlying line of execution by the infrastructure. In icecap, that is a VM process wrapped through a SCJProcess. Thus, here we encode that decision of linking the managed event with its line of execution through the free type.

\[
\text{HandlerSet}0
\]
\[
\text{idle} : \text{PID}
\]
\[
\text{meh}, \text{peh}, \text{ach}, \text{oseh} : P \text{ PID}
\]
\[
\text{idle} \in \text{peh}
\]
\[
\langle \text{peh}, \text{ach}, \text{oseh} \rangle \text{ partition meh}
\]

\[
\text{HandlerSet}
\]
\[
\text{HandlerSet0}
\]
\[
\text{freeHS} : P \text{ PID}
\]
\[
\langle \text{meh}, \text{freeHS} \rangle \text{ partition PID}
\]

\[
\text{IDLE_PID_WITNESS} == \text{maxpid}
\]

[C circus tools off]

\text{theorem} \text{ grule IDLE_PID_WITNESSType}
\text{ IDLE_PID_WITNESS} \in \text{PID}

[C circus tools on]
**HandlerSetInit**

\[
\begin{align*}
\text{meh}' &= \{ \text{idle}' \} \\
\text{freeHS}' &= \text{PID} \setminus \{ \text{idle}' \}
\end{align*}
\]

**AddHandler0**

\[
\begin{align*}
\Delta \text{HandlerSet} \\
p? : \text{PID}
\end{align*}
\]

\[
\begin{align*}
p? \in \text{freeHS} \\
\text{freeHS}' &= \text{freeHS} \setminus \{ p? \} \\
\text{meh}' &= \text{meh} \cup \{ p? \}
\end{align*}
\]

**ContainsHandler**

\[
\begin{align*}
\Xi \text{HandlerSet} \\
p? : \text{PID}
\end{align*}
\]

\[
p? \in \text{meh}
\]

\[
\text{AddHandler} = \text{AddHandler0} \lor \text{ContainsHandler}
\]

**RemoveHandler0**

\[
\begin{align*}
\Delta \text{HandlerSet} \\
p? : \text{PID}
\end{align*}
\]

\[
\begin{align*}
p? \in \text{meh} \land p? \neq \text{idle} \\
\text{meh}' &= \text{meh} \setminus \{ p? \} \\
\text{freeHS}' &= \text{freeHS} \cup \{ p? \}
\end{align*}
\]

**UnknownHandler**

\[
\begin{align*}
\Xi \text{HandlerSet} \\
p? : \text{PID}
\end{align*}
\]

\[
\begin{align*}
p? \in \text{freeHS} \lor p? = \text{idle}
\end{align*}
\]

\[
\text{RemoveHandler} = \text{RemoveHandler0} \lor \text{UnknownHandler}
\]

**RemoveAperiodicHandlers**

\[
\begin{align*}
\Delta \text{HandlerSet}
\end{align*}
\]

\[
\begin{align*}
\text{meh}' &= \text{meh} \setminus \text{ach}
\end{align*}
\]

**HandlerSetSig**

\[
\begin{align*}
\text{HandlerSet} \\
p? : \text{PID}
\end{align*}
\]
4.2 Priority scheduler of Event handlers

\( MIN_{PRIO}, MAX_{PRIO} : \mathbb{N}_1 \)

\( \{\) disabled rule dMinPrio \(\}
MIN_{PRIO} = 1
\( \{\) rule dMinMaxPrio \(\}
MIN_{PRIO} \leq MAX_{PRIO}

\( PRIORITY == MIN_{PRIO} .. MAX_{PRIO} \)

\[ Circus tools on \]

\textbf{theorem} grule MIN\_PRIO\_eq1
\( MIN_{PRIO} \geq 1 \)

\textbf{theorem} grule MAX\_PRIO\_eq1
\( MAX_{PRIO} \geq 1 \)

\textbf{theorem} grule PRIORITY\_Max\_Type
\( PRIORITY \in \mathbb{P} \mathbb{Z} \)

\textbf{theorem} grule PRIORITY\_Type
\( PRIORITY \in \mathbb{P} \mathbb{N} \)

\textbf{theorem} rule lPriorityNonEmpty
\( \neg PRIORITY = {} \)

\[ Circus tools on \]
This version rely on MIN_Prio starting from 1 so that we can model prio as sequences of sequences, where the sequence index is the priority.

\[
\begin{align*}
\text{PriorityScheduler0} &
\text{AScheduler} \\
\text{HandlerSet} &
\text{idle} : \text{PID} \\
\text{prio} : \text{seq}_1 (\text{iseq} \text{ PID})
\end{align*}
\]

\[
\begin{align*}
\text{idle} \in \text{peh} \\
\text{dom_prio} = \text{PRIORITY} \\
\text{meh} = \bigcup (\text{ran (prio} \space \text{ran)}) \\
\langle \{\text{current}\} \setminus \{\text{nullpid}\}, \text{meh, freeHS} \rangle \text{ partition PID}
\end{align*}
\]

[Circus tools off]

\[
\text{PRIO\_WITNESS} \equiv (\lambda x : \text{PRIORITY} \bullet \langle \rangle) \oplus \langle \langle \text{IDLE\_PID\_WITNESS} \rangle \rangle
\]

\text{theorem grule PRIO\_WITNESS\_MaxType}
\text{PRIO\_WITNESS} \in \mathbb{P} (\mathbb{Z} \times \mathbb{P} (\mathbb{Z} \times \mathbb{Z}))

\text{theorem grule PRIO\_WITNESS\_RelType}
\text{PRIO\_WITNESS} \in \mathbb{Z} \leftrightarrow \text{iseq} \text{ PID}

\text{theorem grule PRIO\_WITNESS\_PfunType}
\text{PRIO\_WITNESS} \in \mathbb{N} \mapsto \text{iseq} \text{ PID}

\text{theorem rule PRIO\_WITNESS\_Element}
(1, \langle \text{IDLE\_PID\_WITNESS} \rangle) \in \text{PRIO\_WITNESS}

\text{theorem rule PRIO\_WITNESS\_dom}
\text{dom \ PRIO\_WITNESS} = \text{PRIORITY}

\text{theorem grule PRIO\_WITNESS\_Type}
\text{PRIO\_WITNESS} \in \text{seq}_1 (\text{iseq} \text{ PID})

\text{theorem rule PRIO\_WITNESS\_IDLE\_WITNESS\_Equiv}
\bigcup (\text{ran (PRIO\_WITNESS} \space \text{ran)}) = \{\text{IDLE\_PID\_WITNESS}\}

[Circus tools on]

\[
\begin{align*}
\text{PriorityScheduler0Init} &
\text{PriorityScheduler0'} \\
\text{AScheduler'}
\end{align*}
\]

\[
\begin{align*}
\text{current'} &= \text{nullpid} \\
\text{readyAS'} &= \{\text{idle'}\} \\
\text{blockedAS'} &= \emptyset \\
\text{HandlerSetInit} \\
\text{prio'} &= (\lambda x : \text{PRIORITY} \bullet \langle \rangle) \oplus \langle \langle \text{idle'} \rangle \rangle
\end{align*}
\]

18
The priority scheduler combines the abstract scheduler, the managed event handlers set, and the VM. This is so that all known PIDs share a common partitioning invariant, namely: their free PIDs are the same; and each has their own policy for partitioning of PIDs\(^1\).

\(^1\)This separation of concepts is also useful for taming proof expansion and equality substitution.
\[\text{PriorityScheduler}\]
\[\text{PrioSched2Prio}\]
\[\text{free} = \text{freeAS} = \text{freeHS} = \text{freeVM}\]

\[\text{Circus tools off}\]

\textbf{theorem} rule $f$\textsc{PRIOResMaxRelType}
\[\forall\, \text{prio} : \text{PRIORITY} \rightarrow \text{iseq} \text{ PID}; \, \text{pr} : \text{PRIORITY} \mid \text{pr} \in \text{dom prio} \bullet (\text{prio pr}) \in \mathbb{Z} \leftrightarrow \mathbb{Z}\]

\textbf{theorem} rule $f$\textsc{PRIOResMaxPfunType}
\[\forall\, \text{prio} : \text{PRIORITY} \rightarrow \text{iseq} \text{ PID}; \, \text{pr} : \text{PRIORITY} \mid \text{pr} \in \text{dom prio} \bullet (\text{prio pr}) \in \mathbb{Z} \rightarrow \mathbb{Z}\]

\textbf{theorem} rule $f$\textsc{SPrioResSeqMaxType}
\[\forall\, \text{prio} : \text{PRIORITY} \rightarrow \text{iseq} \text{ PID}; \, \text{pr} : \text{PRIORITY} \mid \text{pr} \in \text{dom prio} \bullet \text{prio pr} \in \text{seq} \mathbb{Z}\]

\textbf{theorem} rule $f$\textsc{SPrioResSeqType}
\[\forall\, \text{prio} : \text{PRIORITY} \rightarrow \text{iseq} \text{ PID}; \, \text{pr} : \text{PRIORITY} \mid \text{pr} \in \text{dom prio} \bullet \text{prio pr} \in \text{seq} \text{ PID}\]

\[\text{Circus tools on}\]

\textbf{PSAdd0}
\[\Delta \text{PriorityScheduler}\]
\[p? : \text{PID}\]
\[\text{pr}? : \text{PRIORITY}\]
\[p? \in \text{free}\]
\[\text{schedPIDs}' = \text{schedPIDs} \smallsetminus \{p?\}\]

\textbf{PSAddKnownPrio}
\[\text{PSAdd0}\]
\[\text{pr}? \in \text{dom prio}\]
\[\text{prio}' = \text{prio} \oplus \{ \text{pr}? \mapsto \text{prio pr}? \smallsetminus \{p?\} \}\]

\textbf{PSAddNewPrio}
\[\text{PSAdd0}\]
\[\text{pr}? \notin \text{dom prio}\]
\[\text{prio}' = \text{prio} \oplus \{ \text{pr}? \mapsto \{ p? \} \}\]

\textbf{PSAddErr}
\[\Xi \text{PriorityScheduler}; \, p? : \text{PID}\]
\[\text{vmerr}! : \text{VMError}; \, \text{aserr}! : \text{ASchedMsg}\]
\[p? \notin \text{free}\]
\[\text{vmerr}! = \text{VMOutOfProcess}\]
\[\text{aserr}! = \text{AS\_CREATE\_ERR}\]
\[ PSD\text{AddPrio} 0 \equiv (PS\text{AddNewPrio} \lor PS\text{AddKnownPrio}) \]

\[ PS\text{AddPrio} \equiv PS\text{AddPrio} 0 \land A\text{Create} \land V\text{MCreate} \land \text{AddHandler}\left[aeh/\text{aeh}'\right] \]

\[ PS\text{Add} \equiv PS\text{AddPrio} \lor PS\text{AddErr} \]

\[
\begin{align*}
PS\text{Dispatch} 0 & \\
\Delta & \text{PriorityScheduler} \\
p! : PID & \]

\[ p! \in \text{ran schedPIDs} \]

\[ PS\text{Dispatch} \equiv PS\text{Dispatch} 0 \]

## 4.3 Priority scheduler operations preconditions

\[
\begin{align*}
PS\text{AddSig} & \\
\text{PriorityScheduler} & \\
p? : PID & \\
pr?: PRIORITY & \]

\[ true \]

\[
\begin{align*}
PS\text{DispatchSig} & \\
\text{PriorityScheduler} & \]

\[ true \]

\textbf{[Circus tools off]}

\textbf{theorem} rule lVMAllAllocated

\[ \forall \text{VM} ; \ p? : PID \mid \text{freeVM} = \{\} \land \neg p? \in \text{createdVM} \bullet p? \in \text{usedVM} \]

\textbf{theorem} rule lHSFreeIsUnknown

\[ \forall \text{HandlerSet} ; \ p? : PID \mid p? \in \text{freeHS} \bullet \neg p? \in \text{meh} \]

\textbf{theorem} frule fVMInvariant

\[ \forall \text{VM} \bullet (\text{createdVM}, \text{usedVM}, \text{freeVM}) \text{ partition PID} \]

\textbf{theorem} frule fPSFreeInvariant

\[ \forall \text{PriorityScheduler} \bullet \text{free = freeAS = freeVM = freeHS} \]

\textbf{theorem} frule fASInvariant

\[ \forall \text{AScheduler} \bullet \]

\[ (\{ \text{current} \} \setminus \{ \text{nullpid} \}, \text{readyAS}, \text{blockedAS}, \text{freeAS}) \text{ partition PID} \]
\textbf{theorem} rule \textit{fASInvariantFreeNullDisjoint} \\
\forall \texttt{AScheduler}; \ p'? : \textit{PID} \ | \ current = \textit{nullpid} \land p'? \in \text{freeAS} \implies \\
\text{(disjoint(\{\}) \land (\text{readyAS} \cup \{p'\}) \land (\text{blockedAS} \land (\text{freeAS} \setminus \{p'\})))}

\textbf{theorem} rule \textit{fASInvariantFreeNonNullDisjoint} \\
\forall \texttt{AScheduler}; \ p'? : \textit{PID} \ | \ \neg current = \textit{nullpid} \land p'? \in \text{freeAS} \implies \\
\text{(disjoint(\{current\} \land (\text{readyAS} \cup \{p'\}) \land (\text{blockedAS} \land (\text{freeAS} \setminus \{p'\})))}

\textbf{theorem} rule \textit{fASInvariantRewrittenNullPartition} \\
\forall \texttt{AScheduler} \ | \ current = \textit{nullpid} \ \implies \ \text{blockedAS} \cup (\text{freeAS} \cup \text{readyAS}) = \textit{PID}

\textbf{theorem} rule \textit{fASInvariantRewrittenNonNullPartition} \\
\forall \texttt{AScheduler}; \ p'? : \textit{PID} \ | \ \neg current = \textit{nullpid} \ \implies \\
\text{blockedAS} \cup (\text{freeAS} \cup (\text{readyAS} \cup \{\text{current}\})) = \textit{PID}

\textbf{theorem} frule \textit{fHS0Invariant} \\
\forall \texttt{HandlerSet}0 \ \bullet \ \langle \text{peh}, \text{aeh}, \text{oseh} \rangle \ \text{partition} \ \text{meh}

\textbf{theorem} frule \textit{fHSInvariant} \\
\forall \texttt{HandlerSet} \ \bullet \ \langle \text{meh}, \text{freeHS} \rangle \ \text{partition} \ \text{PID}

\textbf{theorem} lPSAddPropFreeIsUnknown \\
\forall \texttt{PSAddSig} \ | \ p'? \in \text{free} \ \implies \ \neg p'? \in \text{aeh} \land \neg p'? \in \text{oseh} \land \\
\neg p'? \in \text{blockedAS} \land \neg p'? \in \text{usedVM} \land \\
\neg p'? \in \text{peh}

\textbf{theorem} rule \textit{lCollectPIDSDom} \\
\forall \texttt{prio} : \textit{PRIORITY} \rightarrow \text{iseq} \ \textit{PID} \ \bullet \ \text{dom} (\text{collectPIDS} \ \text{prio}) = \text{dom} \ \text{prio}

\textbf{theorem} rule \textit{lCollectPIDSApply} \\
\forall \texttt{prio} : \textit{PRIORITY} \rightarrow \text{iseq} \ \textit{PID}; \ pr : \textit{PRIORITY} \ | \\
pr \in \text{dom} \ \text{prio} \ \bullet \ (\text{collectPIDS} \ \text{prio}) \ pr = \text{ran} \ (\text{prio} \ pr)

\textbf{theorem} rule \textit{lRanInCollectPIDS} \\
\forall \texttt{prio} : \textit{PRIORITY} \rightarrow \text{iseq} \ \textit{PID}; \ pr : \textit{PRIORITY} \ | \\
pr \in \text{dom} \ \text{prio} \ \bullet \ \text{ran} \ (\text{prio} \ pr) \in \text{ran} \ (\text{collectPIDS} \ \text{prio})

\textbf{theorem} rule \textit{lCollectPIDSOVERRIDEKnownDom} \\
\forall \texttt{prio} : \textit{PRIORITY} \rightarrow \text{iseq} \ \textit{PID}; \ pr'? : \textit{PRIORITY}; \ p'? : \textit{PID} \ | \\

\neg p'? \in \text{ran} \ (\text{prio} \ pr') \land pr'? \in \text{dom} \ \text{prio} \ \bullet \\
\text{collectPIDS} (\text{prio} \oplus \{(pr', (\text{prio} \ pr' \land (\text{p'})))\}) = \\
\text{collectPIDS} \text{prio} \oplus \{(\text{pr'}; \text{ran} (\text{prio} \ pr') \cup \{p'}\)\}
\textbf{theorem} \textit{rule lCollectPIDSOverrideUnknownDom}

\[
\forall \text{prio} : \text{PRIORITY} \to \text{iseq PID}; \ pr? : \text{PRIORITY}; \ p? : \text{PID} |
\neg \ p? \in \text{dom prio} \bullet
\]
\[
\text{collectPIDS (prio} \oplus \{(pr?,(p?))\}) =
\text{collectPIDS prio} \oplus \{(pr?,\{p?\})\}
\]

\textbf{theorem} \textit{rule lRanFlattenPIDSKnownDom}

\[
\forall \text{prio} : \text{PRIORITY} \to \text{iseq PID}; \ pr? : \text{PRIORITY}; \ p? : \text{PID} |
pr? \in \text{dom prio} \bullet \text{ran (prio pr?)} \in \mathbb{P} (\text{flattenPIDS prio})
\]

\textbf{theorem} \textit{rule lFlattenPIDSOverrideKnownDom}

\[
\forall \text{prio} : \text{PRIORITY} \to \text{iseq PID}; \ pr? : \text{PRIORITY}; \ p? : \text{PID} |
pr? \in \text{dom prio} \land \neg \ p? \in \text{ran (prio pr?)} \bullet
\]
\[
\text{flattenPIDS (prio} \oplus \{(pr?,(prio pr? \wedge (p?)))\}) =
\text{flattenPIDS prio} \cup \{p?\} \cup \text{ran (prio pr?)}
\]

\textbf{theorem} \textit{rule lFlattenPIDSOverrideUnknownDom}

\[
\forall \text{prio} : \text{PRIORITY} \to \text{iseq PID}; \ pr? : \text{PRIORITY}; \ p? : \text{PID} |
\neg \ p? \in \text{dom prio} \bullet
\]
\[
\text{flattenPIDS (prio} \oplus \{(pr?,(p?))\}) =
\text{flattenPIDS prio} \cup \{p?\}
\]

\textbf{theorem} \textit{lPSAddPropUniquePrio}

\[
\forall \text{PSAdd} | \text{pr?} \in \text{dom prio} \land \text{p?} \in \text{free} \bullet \neg \text{p?} \in \text{ran (prio pr?)}
\]

\textbf{theorem} \textit{PSAddFSB}

\[
\forall \text{PSAdd} \bullet \text{pre PSAdd}
\]

\textbf{theorem} \textit{PSDispatchFSB}

\[
\forall \text{PSDispatch} \bullet \text{pre PSDispatch}
\]

[Circus tools on]
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>9</td>
<td>55</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>22</td>
<td>36</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>48</td>
<td>91</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>89</strong></td>
<td><strong>194</strong></td>
</tr>
</tbody>
</table>

Table 4.1: Summary of Z declarations for Chapter 4.
Chapter 5

SCJ priority scheduler implementation in Circus

section ic_sched_impl parents circus_toolkit, scj_process

Conventions: * every method will have associated channels * if processing / events happen we have Call/Ret pairs (suffix); otherwise (if just state update) just a synch call * we follow a name convention for the method visibility: priVate, Public, pacKage prefix * the visibility prefix is followed by the a prefix on the class where the channel is defined/used/comes from (PS, CIH, etc)

I am not modelling the static method/attribute associated with the scheduler instance. This would be important if we had more than one scheduler working within the same infrastructure, which is not the case for now. I am also not modelling the methods associated with monitors in SCJ level 2, or debugging methods.

[PID0, SCHID0, CLOCKID0, HANDLERID0, EXCEPTION]

5.0.1 Priority scheduler channels

channel VMthrow, PCIHcatchError : EXCEPTION

SCHID : P1 SCHID0
CLOCKID : P1 CLOCKID0

I will abstract away notions of real and absolute time used by the clock in the scheduler to know when the next periods are. Here, I wil just take time to be N, and now to correspond to what javax.realtime.AbsoluteTime.getTime() returns. model that fully later?

TIME == N

now : TIME

SCJPID : P1 PID

nullsch : SCHID
nullclock : CLOCKID
5.1 Priority Scheduler channels

The next channels represent communication with the `javax.safetycritical.PriorityScheduler` class. Given it only allows for a singleton instance, we do not add another dimension for its instance ID (i.e. in case of multiple schedulers). This is also important to minimise the model checking space.

This PID is for the mission sequencer

channel $KPSaddOuterMostSeqCall : SCJPID$
channel $KPSaddOuterMostSeqRet$

PID is for the process to be stopped. The code doesn’t check/reflect the fact $current \in PID$ might not have been processed by the scheduler, even though from what we could gather it is.

channel $KPSstopCall : PID$
channel $VprocessStart, KPSstopRet, KPSstartCall, KPSstartRet$

PID is the next process to be run. It returns SCJProcess though.

channel $KPSmoveCall$
channel $KPSmoveRet : SCJPID$

channel $KPSreleaseCall : SCJPID$
channel $KPSreleaseRet$

PID is the current process.

channel $KPSgetCurrentProcess : SCJPID$

PID is the current process.

channel $KPSinsertReadyQueueCall : SCJPID$
channel $KPSinsertReadyQueueRet$

Priorities are defined as $\mathbb{Z}$, yet assignable values are always positive within $1..150$ (see `javax.scj.util.Priorities`).

channel $PPSgetMinHWPrio, PPSgetMaxHWPrio, PPSgetMinPrio, PPSgetMaxPrio : \mathbb{Z}$

Circus tools here are not taking type declared locally in the basic process as visible for the channel declaration, yet locally declared channels are allowed. hum.

channel $ENVinstancePS, ENVcreatePSBridge : SCHID$
channel $ENVcreatePFrame$
channel $ENVinstanceRTC : TIME$
channel $PVMcRecreateProcess : PID$
channel $PVMtransferTo : PID \times PID$

5.2 Clock interrupt handler channels

Similarly, as there is only one clock interrupt handler, there is no need to add the $CLOCKID$ dimension to the channels.

channel $PCIHinitialise : SCHID \times \mathbb{Z}$
channel $PCIHregisterCall, PCIHregisterRet, PCIHyieldCall, PCIHyieldRet, PCIHenableCall, PCIHenableRet, PCIHdisableCall, PCIHdisableRet, PCIHhandleCall, PCIHhandleRet, VMcihRegister$
channel $PCIHinstance : CLOCKID$
channel $PCIHstartClockHandlerCall : PID$
channel $PCIHstartClockHandlerRet$

channel $FIXME$
5.3 SCJ priority scheduler bridge

channel PPSIgetNextProcessCall
channel PPSIgetNextProcessRet : PID

process PrioSchedulerImpl ≡ begin

We simplify the state removing the reference to its scheduler and clock id, given there is only one. This is also useful for the FDR encoding.

\[
\begin{array}{l}
\text{PSIState} \\
\quad \text{instance} : \text{SCHID} \\
\quad \text{current} : \text{GPID}
\end{array}
\]

\[
\begin{array}{l}
\text{InitPSIState} \\
\quad \text{PSIState}' \\
\quad b? : \text{SCHID} \\
\quad \text{instance}' = b? \\
\quad \text{current}' = \text{nullpid}
\end{array}
\]

\[
\begin{array}{l}
\text{state PSIState}
\end{array}
\]

Here the call to javax.safetycritical.PriorityScheduler.stop should either not need the parameter (i.e. it is always the current), or it should use the accessor method, which we model. Instead it directly access the field! We model here by calling the accessor method.

\[
\begin{align*}
\text{GetNextProcess} & \equiv \text{PCIHdisableCall} \rightarrow \text{PCIHdisableRet} \rightarrow \\
& \quad \text{KPSmoveCall} \rightarrow \text{KPSmoveRet}?p \rightarrow \text{current} := p \\
\text{InitSt} & \equiv \text{ENVcreatePSBridge}?b : (b \neq \text{nullsch}) \rightarrow (\text{InitPSIState}) \\
\text{InitCIH} & \equiv \text{PCIHinstance}?c : (c \neq \text{nullclock}) \rightarrow \text{Skip} \\
\text{InitPSISt} & \equiv \text{InitSt}
\end{align*}
\]

\[
\begin{align*}
\text{PSIRun} & \equiv \mu X \cdot \text{PPSIgetNextProcessCall} \rightarrow \text{GetNextProcess} ; \text{PPSIgetNextProcessRet}?\text{current} \rightarrow X \\
& \quad \cdot \text{InitPSISt} ; \text{PSIRun}
\end{align*}
\]

end
5.4 SCJ priority scheduler process

This basic process represents the SCJ class javax.safetycritical.PriorityScheduler.

\[
\text{process } \text{PrioScheduler } \triangleq \text{begin}
\]

5.4.1 SCJ process abstraction

SCJ process is an SCJ abstraction used to encapsulate a vm.Process within the various javax.safetycritical.ManagedScheduleable used within the SCJ API. That is, each managed object encapsulates a javax.safetycritical.ScjProcess that encapsulates a vm.Process. The former reflects the SCJ scheduling protocol: what state the process is (e.g. running, blocked, etc), and what is the next release time. The latter reflects the OS line of execution abstraction that the SCJ process represents.

In here we model this using schemas, instead of OhCircus classes for convenience and type-checking.

\[
\text{SCJSTATE ::= NEW } | \text{ READY } | \text{ EXECUTING } | \text{ BLOCKED } | \text{ SLEEPING } | \text{ HANDLED } | \text{ TERMINATED}
\]

\[
\begin{align*}
\text{ScjProcess} & : \text{SCJSTATE} \\
\text{nextActivation} & : \text{TIME} \\
\text{target} & : \text{GPID}
\end{align*}
\]

\[
\begin{align*}
\text{InitScjProcess} & : \text{ScjProcess}' \\
\text{h?} & : \text{GPID} \\
\text{h?} & \neq \text{nullpid} \\
\text{target}' & = \text{h?} \\
\text{state}' & = \text{NEW}
\end{align*}
\]

5.4.2 Priority frame containing schedulable queues

The PriorityFrame schema is used to represent the javax.safetycritical.PriorityFrame auxiliary class: it contains only “passive” methods in the sense that it does not have its own line of execution. In here we do not model the queues used for wait/notify.

\[
\begin{align*}
\text{queue0} & : \text{Queue0} \\
\text{heapSize} & : \text{N} \\
\text{tree} & : \text{seq PID}
\end{align*}
\]

Queue initialisation sets heap size (i.e., current pointer inside the tree array) to 0. The heap itself is initialised to −999 for the given capacity plus one. Because we are using Z sequences (start from 1), instead of Java arrays (start from 0) we add 1 (i.e. in Java, it is new int[capacity+1] or 0..capacity or capacity + 1 elements; in Z, we have a sequence from 1..capacity + 1).

\[
\begin{align*}
\text{InitQueue0} & : \text{Queue0}' \\
\text{capacity} & : \text{N} \\
\text{heapSize}' & = 0 \\
\text{tree}' & = (\lambda x : 1..(\text{capacity} + 1) \bullet \text{nullpid})
\end{align*}
\]

This is convoluted and will lead to unnecessarily complex proofs. I will use a partial sequence instead of a total sequence with null pointers inside.
relation(\langle sleep \rangle)
relation(\langle \text{prio} \rangle)

\langle \text{prio} \rangle: \text{PID} \leftrightarrow \text{PID}
\langle \text{sleep} \rangle: \text{PID} \leftrightarrow \text{PID}
\text{prio}: \text{PID} \rightarrow N_1
\text{sleep}: \text{PID} \rightarrow \text{TIME}

\forall s_1, s_2: \text{PID} \bullet s_1 <_{\text{sleep}} s_2 \leftrightarrow \text{sleep} s_1 < \text{sleep} s_2
\forall p_1, p_2: \text{PID} \bullet p_1 >_{\text{prio}} p_2 \leftrightarrow \text{prio} p_1 > \text{prio} p_2

Queue
\begin{align*}
\text{heapSize}, \text{heapCapacity} &: N \\
\text{tree} &: \text{seq PID}
\end{align*}

\begin{align*}
\text{heapSize} &\leq \text{heapCapacity} \\
\text{dom tree} &\subseteq 1..\text{heapSize}
\end{align*}

PQueue
Queue

\begin{align*}
\forall i, j : \text{dom tree} | i < j &\bullet \text{tree} i >_{\text{prio}} \text{tree} j
\end{align*}

SQueue
Queue

\begin{align*}
\forall i, j : \text{dom tree} | i < j &\bullet \text{tree} i <_{\text{sleep}} \text{tree} j
\end{align*}

InitQueue
Queue'
\begin{align*}
\text{capacity'?} &: N
\end{align*}

\begin{align*}
\text{heapSize'} &= 0 \\
\text{heapCapacity'} &= \text{capacity'} \\
\text{tree'} &= ()
\end{align*}

QueueEmpty
\begin{align*}
\Xi &\subseteq \text{Queue} \\
p!' &: \text{PID}
\end{align*}

\begin{align*}
\text{heapSize} &\leq 0 \\
p!' &= \text{nullpid}
\end{align*}

QueuePeek0
\begin{align*}
\Xi &\subseteq \text{Queue} \\
p!' &: \text{PID}
\end{align*}

\begin{align*}
\text{heapSize} &> 0 \\
p!' &= \text{head tree}
\end{align*}
**FixedCapacityQueue**

\[ \Delta \text{Queue} \]

heapCapacity' = heapCapacity

**QueueExtract0**

\[ \text{FixedCapacityQueue} \]

\[ p! : \text{PID} \]

heapSize > 0

\[ p! = \text{head tree} \]

\[ \text{tree}' = \text{tail tree} \]

\[ \text{heapSize}' = \text{heapSize} - 1 \]

QueuePeek == QueuePeek0 \lor QueueEmpty

QueueExtract == QueueExtract0 \lor QueueEmpty

**QueueInsert**

\[ \text{FixedCapacityQueue} \]

\[ p? : \text{PID} \]

heapSize < heapCapacity

\[ \text{heapSize}' = \text{heapSize} + 1 \]

\[ \text{tree}' = \text{tree} \setminus \{p?\} \]

PQueueInit == (InitQueue \land PQueue')

SQueueInit == (InitQueue \land SQueue')

PQueueExtractMax == (QueueExtract \land \Delta PQueue)

SQueueExtractMax == (QueueExtract \land \Delta SQueue)

PQueueInsert == (QueueInsert \land \Delta PQueue)

SQueueInsert == (QueueInsert \land \Delta SQueue)

SQueuePeek == (QueuePeek \land \Delta SQueue)

On the Circus typechecker, because the state is always in context, the error about renaming the dashed variables didn’t appear! Only in the Z/Eves mode it showed. Hum...

PrioQueue == PQueue[ptree/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}]

SleepQueue == SQueue[stree/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}]

PQInit == PQueueInit[ptree'/\text{tree}', \text{heapSize}'/\text{heapSize}'', \text{heapCapacity}'/\text{heapCapacity}'']

SQInit == SQueueInit[stree'/\text{tree}', \text{heapSize}'/\text{heapSize}'', \text{heapCapacity}'/\text{heapCapacity}'']

PQExtractMax == PQueueExtractMax[ptree'/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}, \text{ptree}'/\text{tree}]

SExtractMin == SQueueExtractMin[stree'/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}, \text{stree}'/\text{tree}]

PQInsert == PQueueInsert[ptree'/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}, \text{ptree}'/\text{tree}', \text{heapSize}]

SQInsert == SQueueInsert[stree'/\text{tree}, \text{heapSize}/\text{heapSize}, \text{heapCapacity}/\text{heapCapacity}, \text{stree}'/\text{tree}', \text{heapSize}]

[Circus tools off]

**theorem** test

PQueueExtractMax
proof\[test\]
\begin{itemize}
\item invoke PQExtractMax;
\item invoke PQueueExtractMax;
\item invoke QueueExtract;
\item invoke \(\Delta\) PQueue;
\item disjunctive;
\item cases;
\item invoke QueueExtract0;
\item invoke FixedCapacityQueue;
\item invoke \(\Delta\) Queue;
\item invoke PQueue;
\item prove;
\item rewrite;
\end{itemize}

\[Circus\ tools\ on\]

\begin{verbatim}
PQInit
PQInit
\end{verbatim}

\begin{verbatim}
SQInit
\end{verbatim}

\begin{verbatim}
\end{verbatim}

5.4.3 Priority scheduler state

The state is composed of the various declared fields.

\begin{verbatim}
SchInfo
\end{verbatim}

\begin{verbatim}
ProcInfo
\end{verbatim}

The scheduler singleton static field is used to access the scheduler object instance from other classes. Here we model this with an \textit{instance} \(\in\ SCHID\) (in Java, this is through the \textit{instance()} method). The bridge class \texttt{javax.safetycritical.PrioritySchedulerImpl} is used to link the underlying \texttt{vm.Scheduler} interface, with the SCJ run-time environment (RTE) scheduler interface (\texttt{javax.realtime.PriorityScheduler}). This is captured here with another scheduler id named \texttt{bridge}. Where appropriate, communication using these ids will represent which scheduler is being refered to. This is to both represent the \textit{instance} method, as well as the bridge but without the need for an explicit Circus
class. The scheduler also maintain a reference to a clock interrupt handler (cih) that is responsible for the actual context switch between low-level \texttt{vm.Process} representing the various SCJ processes (\texttt{javax.safetycritical.ScjProcess}), which themselves are encapsulated by the various SCJ infrastructure objects (i.e. \texttt{javax.safetycritical.PeriodicEventHandler}, \texttt{javax.safetycritical.Mission}, \texttt{javax.safetycritical.MissionSequencer}, etc.).

```
PSState
  SchInfo
  ProcInfo
  PriorityFrame
  HandlerSet

meh \cap seh = \emptyset

dom procs = meh \cup seh \setminus \{ nullpid \}
```

The scheduler maintains a reference to three \texttt{javax.safetycritical.ScjProcess} objects, which wrap a corresponding \texttt{vm.Process}. Here these are represented simply by \texttt{SCJPID} references, which are a subset of \texttt{PID} to correspond to each class instances respectively. The \texttt{mainProcess} represents the line of execution of the scheduler itself. It is to this ID that the clock interrupt handler returns control after scheduling. The \texttt{current} ID relates to the current process running as expected by the scheduling policy. The scheduler gives it package priority, yet also provide a package access method \texttt{(getCurrentProcess())}. The only relevant place it is used is to call the \texttt{stop(vm.Process)} method from the priority scheduler bridge. Arguably you could call it with a different \texttt{PID}, yet that would mean asking the scheduler to stop a process it does not manage, hence there should be no parameter to the \texttt{stop()} method. The \texttt{outerMostSeqProcess} ID represents the outermost mission sequencer line of execution. The priority frame represents the data aspects behind scheduling policy choices.

We also bring to it information that is scattered around various classes apparently for convenience access. For instance, the handler that each \texttt{ScjProcess} represents is put inside \texttt{javax.safetycritical.Mission} indirectly via another class (\texttt{javax.safetycritical.ManagedSchedulableSet}). We keep it here using the \texttt{HandlerSet} abstraction defined before: it separates what kind of handler instantiated \texttt{ScjProcesses} represent, effectively bringing the \texttt{javax.safetycritical.ManagedSchedulableSet} from \texttt{javax.safetycritical.Mission} to the \texttt{javax.safetycritical.PriorityScheduler}. It is kept in mission we guess because the mission class also knows about all created missions, hence an array of arrays of all missions’ and their corresponding set of managed scheduleables. These two dimensional separation is useful for controlling mission’s handlers, but is irrelevant to the scheduler, which is concerned only with the corresponding \texttt{javax.safetycritical.ScjProcess}. Thus, removing this structure here brings no discrepancy as far as scheduling is concerned.

The invariant about \texttt{instance} is artificial: they are guaranteed by semantics of the \texttt{new} on any sound memory manager (i.e. their instances will always have different addresses, unless they are \texttt{null}). The known \texttt{ScjProcess} are those with associated managed event handlers (meh), and they are unique (injective): you cannot schedule the same \texttt{ScjProcess} twice. This \texttt{injectivity property is not enforced through the implementation, but is implicitly expected.}

The state is initialised with the \texttt{Init} action, which corresponds to the class (private) default (and only) constructor. Instantiation of data-related classes are represented with synchronisation channels to be offered by the execution environment.

```
state PSState

InitSchInfo
  SchInfo'

  cih' = nullclock
  rtc' = 0
  instance' = bridge' = nullsch
```
InitProcInfo

ProcInfo'

\[\begin{align*}
\text{current}' &= \text{mainProcess}' = \text{outerMostSeqProcess}' = \text{nullpid} \\
\text{seh}' &= \{\text{current}', \text{mainProcess}', \text{outerMostSeqProcess}'\} \\
\text{procs}' &= \emptyset
\end{align*}\]

InstantiatePS

PSState'

InitSchInfo

InitProcInfo

InitPrioFrame

HandlerSetInit

InitSchIds

\[\begin{align*}
\Delta\text{PSState} \\
\Xi\text{ProcInfo} \\
\Xi\text{PriorityFrame} \\
\Xi\text{HandlerSet} \\
i?, b? : \text{SCHID}
\end{align*}\]

\[\begin{align*}
i? \neq \text{nullsch} \land b? \neq \text{nullsch} \\
i? \neq b? \\
\text{instance}' = i? \\
\text{bridge}' = b? \\
\text{cih}' = \text{cih} \land \text{rtc}' = \text{rtc}
\end{align*}\]

Some of the constants present in javax.safetycritical.Consts.

\[\begin{align*}
\text{STACK\_UNIT} &= 1024 \\
\text{PS\_STACK\_SIZE} &= 1 \times \text{STACK\_UNIT} \\
\text{DEFAULT\_PQUEUE\_SIZE} &= 20
\end{align*}\]

Initialisation is done in three stages: i) various references are created and initialised through the environment (i.e., in this case a memory manager responding to a new command); ii) the clock interrupt handler initialisation; and iii) the real time clock (time) initialisation.

\[\begin{align*}
\text{InitSt} \triangleq \text{ENVinstancePS?i : (i \neq \text{nullsch})} \rightarrow \text{ENVcreatePFrame} \rightarrow \\
&\text{ENVcreatePSBridge?b : (b \neq \text{nullsch})} \rightarrow \big(\text{InitSchIds}\big)
\end{align*}\]

\[\begin{align*}
\text{InitClock} \triangleq \text{ENVinstanceRTC?clock} \rightarrow \text{rtc} := \text{clock}
\end{align*}\]

\[\begin{align*}
\text{Init} \equiv \text{InitSt} \land \text{InitClock}
\end{align*}\]

5.4.4 Various state operations
theorem test2
InsRQueueOp

proof[test]
invoke InsRQueueOp;
invoke Δ PriorityFrame;
invoke PQueueInsert;
invoke QueueInsert;
invoke Δ PQueue;
disjunctive;
cases;
invoke QueueInsert0;
invoke FixedCapacityQueue;
invoke Δ Queue;
invoke PQInsert;
prove;
rewrite;

[5.4.5 SCJ clock interrupt handler API]

Actions associated with the SCJ underlying VM infrastructure through its clock interrupt handler (CIH) API: these are the priority scheduler required API by the underlying OS like what to scheduler next. All its events are to be hidden.

Move ≡ var next : PID • KPSmoveCall →→ FIXME →→ KPSmoveRet!next →→ Skip
the code does not make such filter, but I think it is a necessary one, and also an expected one given the call graph hierarchy.

SCJStop ≡ KPSstopCall? curr : (curr ≠ nullpid) →→
PVMtransferTo!curr!mainProcess →→ KPSstopRet →→ Skip

Next we define the events associated with one of these actions as well as the SCJ CIH API overall. Perhaps these channels need to be defined outside the priority scheduler process

channelset csMove == { KPSmoveCall, KPSmoveRet }
channelset csSCJStop == { KPSstopCall, KPSstopRet }
channelset csCIHApi == csMove ∪ csSCJStop

[5.4.6 SCJ run-time environment API]

Actions associated with the SCJ run time environment (RTE): these are the priority scheduler provided API to be used by the SCJ RTE like the clock interrupt handler and other parts. All its events are to be hidden.

AddOuterMostSeq ≡ Skip

Start ≡ var p : PID • KPSstartCall →→ (PQExtractMax) ; current := p;
PVMcreateProcess? m : (m ≠ nullpid) →→ mainProcess := m;
PCIHregisterCall →→ PCIHregisterRet→→
PCIHenableCall →→ PCIHenableRet→→
PCIHstartClockHandlerCall!mainProcess→→
PCIHstartClockHandlerRet→→
PCIHyieldCall →→ PCIHyieldRet →→ KPSstartRet →→ Skip

34
Here I think the apeh parameter should only be about the processes that the scheduler know, hence the need for the precondition on it being within the domain of procs? Should it be guard (block) or test (diverge)?

\[\text{ReleaseHandlerNo} \equiv \text{apeh} : \text{PID} \bullet (\text{apeh} \in \text{dom procs}) \& \text{if}((\text{procs apeh}).\text{state} = \text{EXECUTING}) \rightarrow \text{FIXME} \rightarrow \text{Release} \equiv \text{KPSreleaseCall!apeh} \rightarrow \text{PCIHdisableCall} \rightarrow \text{PCIHdisableRet} \rightarrow (\text{apeh} \in \text{dom}) \& (\text{ReleaseHandler});\]
\[\text{PCIHenableCall} \rightarrow \text{PCIHenableRet} \rightarrow \text{KPSreleaseRet} \rightarrow \text{Skip}\]

\[\text{GetCurrentProc} \equiv \text{KPSgetCurrentProcess!current} \rightarrow \text{Skip}\]

\[\text{InsertReadyQueue} \equiv \text{KPSinsertReadyQueueCall?p} \rightarrow (\text{InsertReadyQueueOp});\]
\[\text{KPSinsertReadyQueueRet} \rightarrow \text{Skip}\]

Next we define the events associated with one of these actions as well as the SCJ RTE API overall.

channelset csStart == \{ \text{KPSstartCall, KPSstartRet} \}
channelset csAddOuterMostSeq == \{ \text{KPSaddOuterMostSeqCall, KPSaddOuterMostSeqRet} \}
channelset csGetCurrentProcess == \{ \text{KPSgetCurrentProcess} \}
channelset csInsertReadyQueue == \{ \text{KPSinsertReadyQueueCall, KPSinsertReadyQueueRet} \}
channelset csRelease == \{ \text{KPSreleaseCall, KPSreleaseRet} \}
channelset csSCJRTE == csStart \cup csAddOuterMostSeq \cup csGetCurrentProcess \cup
\text{csInsertReadyQueue} \cup csRelease

5.4.7 SCJ user API actions

Actions associated with the SCJ user API: these are the scheduler provided API for the SCJ application developer. These channels are not to be hidden, and will synchronise with the Circus SCJ models for the user program.

Priorities are defined inside \texttt{javax.scj.util.Priorities} as final static constant.

| SCJ_MIN_PRIO == 1 |
| SCJ_MAX_PRIO == 100 |
| MIN_HW_PRIO == 101 |
| MAX_HW_PRIO == 150 |
The actions just offer these constants through the appropriate channel accordingly.

\[
\text{GetHW\text{Prio}} \triangleq \text{PPSgetMinHW\text{Prio}}!\text{MIN}_\text{HW\_PRIO} \rightarrow \text{Skip}
\]

\[
\text{PPSgetMaxHW\text{Prio}}!\text{MAX}_\text{HW\_PRIO} \rightarrow \text{Skip}
\]

\[
\text{GetPrio} \triangleq \text{PPSgetMinPrio}!\text{MIN\_PRIO} \rightarrow \text{Skip}
\]

\[
\text{PPSgetMaxPrio}!\text{MAX\_PRIO} \rightarrow \text{Skip}
\]

Next we define the events associated with one of these actions as well as the SCJ user API overall.

\[
\text{channelset } cs\text{GetHW\text{Prio}} == \{\text{PPSgetMinHW\text{Prio}}, \text{PPSgetMaxHW\text{Prio}} \}
\]

\[
\text{channelset } cs\text{GetPrio} == \{\text{PPSgetMinPrio}, \text{PPSgetMaxPrio} \}
\]

\[
\text{channelset } cs\text{SCJApi} == cs\text{GetHW\text{Prio}} \cup cs\text{GetPrio}
\]

### 5.4.8 Priority scheduler API network

Actions associated with plumbing together the various parts of the priority scheduler. Each separate API events are offered as an external choice to its environment, where the combination of APIs are interleaved. This reflects the fact some APIs are called within the user, the RTE, or the clock interrupt handler (CIH) line of execution (see \text{vm.ProcessLogic} and \text{vm.Process}).

\[
\text{CIHApi} \triangleq \text{Move} \square \text{SCJStop}
\]

\[
\text{SCJRTE} \triangleq \text{Start} \square \text{AddOuteMostSeq} \square \text{Release} \square \text{GetCurrentProc} \square \text{InsertReadyQueue}
\]

\[
\text{SCJApi} \triangleq \text{GetHW\text{Prio}} \square \text{GetPrio}
\]

\[
\text{Run} \triangleq SCJ\text{Api} \parallel SCJRTE \parallel CIH\text{Api}
\]

\[
\text{Execute} \triangleq \text{Run} \Delta \text{Catch}
\]

\[
\text{Catch} \triangleq \text{PCIHcatchError}\?e \rightarrow \text{Skip}
\]

\[
\bullet \text{Init} ; \text{Execute}
\]

\[
\text{end}
\]

\[
\text{process } \text{ClockInterruptHandler} \triangleq \text{begin}
\]

The \text{vm.ClockInterruptHandler} class has \textbf{static} attributes for some fields, but given we modelling a single interrupt handler as the clock, this is not relevant. If/when we model generic handlers, we would need to generalise this a bit.
Here we are only modelling a single/simple ClockInterruptHandler, and hence simplifying the clock handler dimension needed in all these events. In the implementation, other examples of interrupt handlers are those associated with IO buffers, and other hardware interfaces.

there is an implicit precondition (or guard?) here about disableCount ≥ 0. Will add as precondition
Enable ≡ (disableCount > 0) & PCHenableCall → (CIHEnable); PCHenableRet → Skip

Disable ≡ (disableCount ≥ 0) & PCHdisableCall → disableCount, hemClockReady := disableCount + 1, False; PCHdisableRet → Skip

CIHApi ≡ Disable □ Enable
   □
   Handle □ Yield
   □
   Register □ StartClockHandler

Loop ≡ μX • PPSIgetNextProcessCall → PPSIgetNextProcessRet?p → current := p;
   Enable ; PVMtransferTo.handler.current → X

Catch ≡ VMthrow?e → PCHcatchError!e → Skip

Run ≡ Loop △ Catch

Execute ≡ Run || CIHApi

• (InitCIH ; Execute) \ { VMthrow, VMcihRegister }

end

channelset CihPsInterface == {} {}
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>26</td>
<td>81</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>14</td>
<td>26</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>31</td>
<td>67</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>Proofs</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>75</strong></td>
<td><strong>269</strong></td>
</tr>
</tbody>
</table>

Table 5.1: Summary of Z declarations for Chapter 5.

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>26</td>
<td>261</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>14</td>
<td>26</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>31</td>
<td>67</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>Proofs</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>75</strong></td>
<td><strong>269</strong></td>
</tr>
</tbody>
</table>

Table 5.2: Summary of Circus declarations for Chapter 5.
Chapter 6

SCJ framework implementation in Circus

section scjfwimpl parents circus_toolkit, scj_process

6.1 Launcher

The SCJ launcher performs two tasks for a given Safelet class parameter: create necessary memory areas, and executes the appropriate scheduler (in our case PriorityScheduler) within the immortal memory area just created for the given safelet.

channel allocate_backing_store, allocate_immortal_memory
channel start_prio_scheduler
channel start_sequencer, done_sequencer
process LauncherL1 ≡ begin

As we have an abstract description of memory and the VM, we simply abstract memory management details, and flag them through events. This is okay given

\[
\begin{align*}
\text{DISCUSS: interface with memory areas} \\
\text{MemAlloc} &\equiv \text{allocate}_\text{backing_store} \rightarrow \text{allocate}_\text{immortal_memory} \rightarrow \text{Skip} \\
\text{StartMissionSeq} &\equiv \text{start}_\text{sequencer} \rightarrow \text{done}_\text{sequencer} \rightarrow \text{Skip} \\
\text{StartPrioSched} &\equiv \text{start}_\text{prio_scheduler} \rightarrow \text{Skip}
\end{align*}
\]

- MemAlloc; StartMissionSeq; StartPrioSched

end
6.2 Clock interrupt handler

channel clock_handler_init : Clock
clock_handler_register : Clock
clock_handler_enable : Clock
clock_handler_disable : Clock
clock_handler_start : Clock × PID
clock_handler_yield : Clock
proc_transfer : PID × PID
getNextProcessCall
getNextProcessRet : PID

process ClockHandler ≡ begin

State
- clock : Clock
- current : PID
- handler : PID
- clockready : B
- disablecnt : N
- current ̸= handler

InitSt

InitSt'

State'
- clockready' = False
- disablecnt' = 1

This is not total as disablecnt could go negative

EnReady

ΔState
- disablecnt' = disablecnt - 1
- disablecnt = 0 ⇒ clockready' = True
- disablecnt ≠ 0 ⇒ clockready = clockready'

state State
Init ≡ (InitSt) ; clock_handler_init.clock → clock_handler_register.clock → Skip
Disable ≡ clock_handler_disable.clock → disablecnt, clockready := disablecnt + 1, False
Enable ≡ clock_handler_enable.clock → (EnReady)
Start ≡ clock_handler_start.clock?p → current := p
$Yield \triangleq \text{clock}_{\text{handler}} \cdot \text{yield} \cdot \text{clock} \rightarrow \text{Handle}$

$Handle \triangleq \text{Disable} \; ; \; \text{proc}_{\text{transfer}} \cdot \text{current} \cdot \text{handler} \rightarrow \text{Skip}$

$Run \triangleq \mu X \cdot \text{getNextProcessCall} \rightarrow$

\[\text{getNextProcessRet} \; ? \; p \rightarrow\]
\[\text{current} \; := \; p \; ; \; \text{Enable} ;
\]
\[\text{proc}_{\text{transfer}} \cdot \text{handler} \cdot \text{current} \rightarrow X \]

$Execute \triangleq (\text{Disable} \; \circ \; \text{Enable} \; \circ \; \text{Start} \; \circ \; \text{Yield} \; \circ \; \text{Handle}) \; \parallel \; Run$

\[\bullet \; \text{Init} \; ; \; \text{Execute} \]

end
6.3 Priority Scheduler (Level1)

\[ \text{process} \text{Scheduler} \equiv \text{begin} \]

\[
\begin{array}{l}
\text{State} \\
\quad \text{PriorityScheduler} \\
\quad \text{clock} : \text{Clock} \\
\quad \text{mainProcess} : \text{PID} \\
\end{array}
\]

TODO: decide here whether to create idle explicitly like this and change the state representation to allow it, or ignore it and it is simply a specification mechanism as in the current Z model.

\[
\begin{array}{l}
\text{state} \text{State} \\
\quad \text{TODO: decide whether to have the creation as part of the state or not} \\
\text{CreateProcess} \equiv \text{var aserr : ASchedMsg; vmerr : VMError} \bullet \\
\quad \left( (ACreate \land VMCreatemainProcess/p!) \right); \\
\quad \left( \text{if}(\text{aserr} = \text{AS\_OKAY} \land \text{vmerr} = \text{VMOkay}) \rightarrow \text{Skip} \right) \\
\quad \left( \text{if}(\text{aserr} \neq \text{AS\_OKAY} \lor \text{vmerr} \neq \text{VMOkay}) \rightarrow \text{Stop} \right) \\
\end{array}
\]

\[
\begin{array}{l}
\text{CreateIdle} \equiv \text{CreateProcess} ; \text{Skip} \\
\end{array}
\]

\[
\begin{array}{l}
\text{Clock interface to discuss} \\
\text{StartClockHandler} \equiv \text{clock\_handler\_register!clock} \rightarrow \\
\quad \text{clock\_handler\_enable!clock} \rightarrow \\
\quad \text{clock\_handler\_start!clock!mainProcess} \rightarrow \\
\quad \text{clock\_handler\_yield!clock} \rightarrow \text{Skip} \\
\end{array}
\]

\[
\begin{array}{l}
\text{StartScheduler} \equiv \text{var vmerr : VMError} \bullet \\
\quad \left( \text{VMCreate}[\text{mainProcess}/p!] \right); \\
\quad \left( \text{if}(\text{vmerr} = \text{VMOkay}) \rightarrow \text{Skip} \right) \\
\quad \left( \text{if}(\text{vmerr} \neq \text{VMOkay}) \rightarrow \text{Stop} \right) \\
\end{array}
\]

\[
\begin{array}{l}
\text{Init} \equiv \text{clock\_handler\_init?clock} \rightarrow \text{CreateIdle} ; \text{StartScheduler} ; \text{StartClockHandler} \\
\end{array}
\]

\[
\begin{array}{l}
\text{NextProc} \equiv \text{getNextProcessCall} \rightarrow \text{clock\_handler\_disable!clock} \rightarrow \\
\quad \text{getNextProcessRet!current} \rightarrow \text{Skip} \\
\end{array}
\]

Here we need to be careful with the clock interrupt handler interface. Not quite right yet

\[
\begin{array}{l}
\text{Execute} \equiv \mu X \bullet \text{NextProc}; \\
\quad \left( \text{if current = nullpid} \rightarrow \right. \\
\quad \quad \text{proc\_transfer!current!mainProcess} \rightarrow \text{Skip} \\
\quad \left. \right) \\
\quad \left( \text{clock\_handler\_enable!clock} \rightarrow X \right) \\
\end{array}
\]

\[
\begin{array}{l}
\bullet \text{Init} ; \text{Execute} \\
\end{array}
\]

\[
\text{end}
\]
6.4 Mission

[MissionId]

channel getNextMissionCall
channel getNextMissionRet : MissionId
channel start_mission : MissionId
channel done_mission : MissionId
channel end_sequence_app, end_mission-fw

nullmid : MissionId

FIX ME: Stuff in ManagedEventHandler and Mission is general for the core infrastructure. Hum.. For example ManagedScheduleableSet within Mission.
Moreover, Mission.getCurrentMission() is trickier: it mixes up with the priority scheduler as well.
Why isn’t the ManagedScheduleableSet in the mission sequencer?

process MissionFW ≡ begin

Here I assume the mission sequencer PID to be like a one-shot event handler in the sense that its execution occurs entirely in one call to handleAssynchEvent().

// Relevant for Level1
MissionSequencer<?>() currMissSeq;
boolean missionTerminate = false;
ManagedScheduleableSet msSetForMission;
Phase missionPhase;

// Irrelevant for Level1
protected int missionIndex = -1;
static volatile Mission[] missionSet = null;
boolean isMissionSetInitByThis = false;
static volatile boolean isMissionSetInit = false;

FIX ME: Here is gets funny: we should model this hierarchically where Mission sequencer is the one to know about its missions, rather than centrally where missions know about their mission sequencers. And then the question: code x good-modelling practice.

State

HandlerSet
mission : MissionId
missionSeq : PID
reqTerm : B

missionSeq ∈ oseh

InitSt

Stat'e

reqTerm' = False
state State

• Skip

end
6.5 Event Handlers

FIX ME: Perhaps use OhCircus here given Mission sequencer extends managed event handler?
6.6 Mission Sequencer

Because of the way missions and sequencers management is entangled in the code, we tried to represent what the underlying model should be like here as well, although we do not make use of such parts of the state yet.

**Mission's managed scheduled set object contains some pretty dangerous OO practices. They allow attribute access across the package for variables that should never be allowed to change externally. They don't change it, but this is a nightmare to trace.**

**MissionSequencer initialisation**

\[
\text{state } \text{State} \\
\text{Start} \equiv \text{start\_sequencer } \rightarrow \text{Skip}
\]
Mission protocol

\[
\text{StartMission} \equiv (\text{current} \neq \text{nullmid} \land \text{missionmem} \neq \text{nullmmem}) \& \\
\hspace{1cm} \text{start\_mission\_current} \rightarrow \\
\hspace{2cm} \text{req\_terminate\_mission\_current.Final} \rightarrow \\
\hspace{3cm} \text{set\_mission\_seq} \rightarrow \\
\hspace{4cm} \text{enter\_mission\_mem\_init} \rightarrow \\
\hspace{5cm} \text{enter\_mission\_mem\_exec} \rightarrow \text{Skip}
\]

In the code, this is done via an Object.wait() call, here we will assume someone will notify the sequencer.

\[
\text{MissionWait} \equiv (\text{current} \neq \text{nullmid}) \land \text{notify\_mission\_current} \rightarrow \text{Skip}
\]

\[
\text{ExecuteMission} \equiv (\text{current} \neq \text{nullmid}) \land \\
\hspace{1cm} \mu X \bullet \text{req\_terminate\_mission\_current?v} \rightarrow \\
\hspace{2cm} \text{if} \ v = \text{False} \land \text{meh}_1 \neq \emptyset \rightarrow \\
\hspace{3cm} \text{MissionWait} \rightarrow X \\
\hspace{4cm} \text{skip} \rightarrow \text{Skip}
\]

\[
\text{done\_mission\_current} \rightarrow \text{Skip}
\]

\[
\text{CleanupMission} \equiv (\text{current} \neq \text{nullmid}) \land \\
\hspace{1cm} \text{enter\_mission\_mem\_cleanup} \rightarrow \text{Skip}
\]

\[
\text{Execute} \equiv \mu X \bullet \text{getNextMissionCall} \rightarrow \text{getNextMissionRet?next} \rightarrow \\
\hspace{1cm} \text{current} := \text{next}; \\
\hspace{2cm} \text{if} \ current = \text{nullmid} \land \text{reqTermination} = \text{True} \rightarrow \\
\hspace{3cm} \text{reqTermination} := \text{True} \\
\hspace{4cm} \text{StartMission}; \text{ExecuteMission}; \text{CleanupMission}; X \\
\hspace{5cm} \text{fi}
\]

MissionSequencer clean up: auxiliary actions that might be called externally.

\[
\text{ReqSeqTerm} \equiv (\text{current} \neq \text{nullmid}) \land \\
\hspace{1cm} \text{reqTermination} := \text{True}; \text{req\_terminate\_mission\_current.Final} \rightarrow \text{Skip}
\]

\[
\text{May be called by PrioScheduler.move()} \text{ when sleeping queue is empty. TODO}
\]

\[
\text{Cleanup} \equiv (\text{missionmem} \neq \text{nullmem}) \land \\
\hspace{1cm} \text{missionmem\_remove} \rightarrow \text{missionmem} := \text{nullmem}
\]

\[
\text{Register} \equiv \text{reg\_event\_handler\_proc} \rightarrow \text{Skip}
\]

\[
\text{Finish} \equiv \text{end\_sequence\_app} \rightarrow \text{end\_mission\_fw} \rightarrow \text{done\_sequencer} \rightarrow \text{Skip}
\]

\[
\text{ExecuteMissionSeq} \equiv \text{Start} \rightarrow \text{Execute} \rightarrow \text{Finish}
\]

\[
\text{External} \equiv \text{reqSeqTerm} \parallel \text{Cleanup} \parallel \text{Register}
\]

\[
\bullet \text{ExecuteMissionSeq} \parallel \text{External}
\]

end
process PROCNAME \equiv begin

State

\quad InitSt
\quad State'
\quad true

state State

• Skip

end
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>3</td>
<td>84</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>10</td>
<td>77</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>0</td>
<td>93</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>15</strong></td>
<td><strong>284</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Summary of Z declarations for Chapter 6.

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>3</td>
<td>84</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Schemas</td>
<td>10</td>
<td>77</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Theorems</td>
<td>0</td>
<td>93</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>15</strong></td>
<td><strong>284</strong></td>
</tr>
</tbody>
</table>

Table 6.2: Summary of Circus declarations for Chapter 6.
Appendix A

Extended Z toolkit for SCJ

section scj_toolkit parents standard_toolkit

A.1 Sets

A.1.1 Definitions

\[
[XX] = \bigcup \text{bigU} : \mathcal{P}(\mathcal{P} \, XX) \to \mathcal{P} \, XX
\]

\[\langle \text{disabled rule dBigU} \rangle \]

\[\forall SS : \mathcal{P}(\mathcal{P} \, XX) \bullet \text{bigU} \, SS = \{v : XX \mid \exists S : \mathcal{P} \, XX \, | \, S \in SS \bullet v \in S\} \]

A.1.2 Lemmas on sets

**Theorem** disabled rule dlBigCupAsBigU [XX]

\[\forall SS : \mathcal{P}(\mathcal{P} \, XX) \bullet \bigcup \, SS = \text{bigU} \, SS\]

**Theorem** disabled rule dlBigUDistCup [XX]

\[\forall A, B : \mathcal{P}(\mathcal{P} \, XX) \bullet \text{bigU} \, (A \cup B) = \text{bigU} \, A \cup \text{bigU} \, B\]

**Theorem** disabled rule dlBigUBigU [XX]

\[\forall C : \mathcal{P}(\mathcal{P} \, XX) \bullet \text{bigU} \, (\{\text{bigU} \, C\}) = \text{bigU} \, C\]

An easy lemma to have BigU just like \(\bigcup\)

**Theorem** disabled rule dlInBigU [XX]

\[\forall SS : \mathcal{P}(\mathcal{P} \, XX) \bullet x \in \text{bigU} \, SS \iff (\exists ss : SS \bullet x \in ss)\]

**Theorem** disabled rule dlInPowerBigU [XX]

\[\forall SS : \mathcal{P}(\mathcal{P} \, XX) \mid x \in SS \bullet x \in \mathcal{P}(\text{bigU} \, SS)\]

Toolkit theorems/lemmas are result useful during proofs. They are divided by category, similarly to the Z mathematical toolkit.

**Toolkit extension A.1 (Set difference distribute to the right on set difference)**

**Theorem** disabled rule lRightDiffLeftDistribute [X]

\[\forall S, T, U : \mathcal{P} \, X \bullet S \setminus (T \setminus U) = S \setminus T \cup S \cap U\]
Toolkit extension A.2 (Set difference equivalence modulo set intersection)

\begin{itemize}
\item \textbf{Theorem} disabled rule lCapEquivWeakensDiffEquiv \([X]\)
\[ \forall S, R, T : P \ X \mid S \cap T = S \cap R \Rightarrow S \setminus T = S \setminus R \]
\end{itemize}

Similar to computeDiff1/2

\begin{itemize}
\item \textbf{Theorem} disabled rule lElemDiffAbsorption \([X]\)
\[ \forall x : X; S : P \ X \mid \neg x \in S \Rightarrow S \setminus \{x\} = S \]
\end{itemize}

Toolkit extension A.3 (Singleton set union absorbs set difference)

\begin{itemize}
\item \textbf{Theorem} disabled rule lElemUnionAbsorbDiffRight \([X]\)
\[ \forall x : X; S : P \ X \mid x \in S \Rightarrow \{x\} \cup (S \setminus \{x\}) = S \]
\end{itemize}

Toolkit extension A.4 (Set union absorbs set intersection)

\begin{itemize}
\item \textbf{Theorem} disabled rule lUnionAbsorbInter \([X]\)
\[ \forall S, T : P \ X \mid S \cup T \cap S = S \]
\end{itemize}

Toolkit extension A.5 (Set union exchange to the right on set difference)

\begin{itemize}
\item \textbf{Theorem} disabled rule lUnionExchangeDiffLeft \([X]\)
\[ \forall S, T, Q : P \ X \mid Q \cap T = \emptyset \Rightarrow (S \setminus T) \cup Q = (S \cup Q) \setminus T \]
\end{itemize}

Trivial lemmas are those useful lemmas for algebraic proofs. For instance, where equations are not in the right shape/order to pattern match available rules.

Trivial lemma A.1 (Flipping equality)

\begin{itemize}
\item \textbf{Theorem} disabled rule lFlipEquiv
\[ x = y \Leftrightarrow y = x \]
\end{itemize}

Toolkit extension A.6 (Set with an element is not empty)

\begin{itemize}
\item \textbf{Theorem} disabled rule lElemSetNonEmpty \([X]\)
\[ \forall \text{set : } P X \Rightarrow \forall \text{elem : set \mid \neg set = \{\}} \]
\end{itemize}

Trivial lemmas named with a “Backwards” (at the end) are dual versions of original toolkit lemmas. For instance, \texttt{Z/Eves} provides the toolkit rule \texttt{cupAssociates}:

\[ \forall S, T, V : P X \Rightarrow (S \cup T) \cup V = S \cup (T \cup V) \]

Sometimes during a proof, it is useful to have this fact the other way, round \textit{(i.e., backwards)}, hence the following trivial lemma.

Trivial lemma A.2 (Set union associates to the left)

\begin{itemize}
\item \textbf{Theorem} disabled rule lCupAssociatesBackwards \([X]\)
\[ \forall S, T, V : P X \Rightarrow S \cup (T \cup V) = (S \cup T) \cup V \]
\end{itemize}

Trivial lemma A.3 (Set difference distributes over union backwards)

\begin{itemize}
\item \textbf{Theorem} disabled rule lDistDiffOverUnionLeftBackwards \([X]\)
\[ \forall A, B, C : P X \Rightarrow (A \setminus C) \cup (B \setminus C) = (A \cup B) \setminus C \]
\end{itemize}

Weakening rule A.1 (Expanding \([\cup\) without knowing the type of \(x\))

\begin{itemize}
\item \textbf{Theorem} disabled rule lBigCupElemType \([X]\)
\[ \forall S : P(P X) \mid x \in \bigcup S \Rightarrow x \in X \]
\end{itemize}
Usually, all weakening rules are enabled, as they mostly always improve the chance to increase the pattern matching of available rewriting rules. Nevertheless, sometimes we need “strengthening” (w.r.t. implication) rules, like the one above. That is, knowing a stronger fact, we can deduce a weaker one.

\[ \text{Circus tools off} \]

\[
\text{theorem rule lDiffAbsorb } [X] \\
\forall S : P X \cdot S \setminus S = \{ \}
\]

\[ \text{Circus tools on} \]

\[
\text{theorem disabled rule lInterAbsorbDiffRight } [X] \\
\forall S, T, R : P X \cdot S \cap T = \{ \} \cdot S \cap (T \setminus R) = \{ \}
\]

A.2 Relations

Weakening rule A.2 (Expanding the (non-maximal) type of \( x \) from \( R \))

\[ \text{theorem disabled rule lDomElemType } [X, Y] \\
\forall A : P X; B : P Y \cdot \forall R : A \leftrightarrow B \mid (x, y) \in R \cdot x \in A
\]

Weakening rule A.3 (Expanding the (non-maximal) type of \( y \) from \( R \))

\[ \text{theorem disabled rule lRanElemType } [X, Y] \\
\forall A : P X; B : P Y \cdot \forall R : A \leftrightarrow B \mid (x, y) \in R \cdot y \in B
\]

Weakening rule A.4 (Expanding the (non-maximal) type of an element of \( R \))

\[ \text{theorem disabled rule lRelElemType } [X, Y] \\
\forall A : P X; B : P Y \cdot \forall R : A \leftrightarrow B \mid x \in R \cdot x \in A \times B
\]

Weakening rule A.5 (Packing pair of \( R \) in its domain)

\[ \text{theorem disabled rule lRelElemInDom } [X, Y] \\
\forall x : X; y : Y; R : X \leftrightarrow Y \mid (x, y) \in R \cdot x \in \text{dom } R
\]

Weakening rule A.6 (Packing pair of \( R \) in its range)

\[ \text{theorem disabled rule lRelElemInRan } [X, Y] \\
\forall x : X; y : Y; R : X \leftrightarrow Y \mid (x, y) \in R \cdot y \in \text{ran } R
\]

\[ \text{Circus tools off} \]

Leave enabled. Maybe add \( A, B \) subtypes for \( X, Y \).

Weakening rule A.7 (Extracting dom \( R \) subtype)

\[ \text{theorem rule lRelDomSubType } [X, Y] \\
\forall R : X \leftrightarrow Y \cdot \text{dom } R \in P X
\]

Leave enabled

Weakening rule A.8 (Extracting dom \( R \) subtype)

\[ \text{theorem rule lRelRanSubType } [X, Y] \\
\forall R : X \leftrightarrow Y \cdot \text{ran } R \in P Y
\]

\[ \text{Circus tools on} \]

This one is useful when one knows fact between \( (x, y) \) and \( R \). Needs to be used with \textbf{with disabled (ranCup) XXXX}; otherwise the prover will follow a undesired path.

53
Trivial lemma A.4 (ran absorbs $\cup$-unit to the left)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRanAbsorbsCupUnitLeft } [X, Y] \\
\forall x : X; \ y : Y; \ R : X \leftrightarrow Y \ \bullet \ \{ y \} \cup \text{ran} \ R = \text{ran} \ (\{ (x, y) \} \cup R)
\end{align*}
\]

Toolkit extension A.7 (Expose $\leftrightarrow$ property)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRelWeakening } [X, Y] \\
R \in X \leftrightarrow Y & \iff \\
R \in \mathcal{P}(X \times Y) & \land \\
(\forall \text{elem} : R \ \bullet \ \text{elem} \in X \times Y)
\end{align*}
\]

Toolkit extension A.8 ($R$ within its own domain)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRelWithinDom } [X, Y] \\
\forall R : X \leftrightarrow Y \ \bullet \ R \in \text{dom} \ R \leftrightarrow Y
\end{align*}
\]

Toolkit extension A.9 ($R$ within its own range)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRelWithinRan } [X, Y] \\
\forall R : X \leftrightarrow Y \ \bullet \ R \in X \leftrightarrow \text{ran} \ R
\end{align*}
\]

 Toolkit extension A.10 ($R$ within its own domain and range)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRelSelfContained } [X, Y] \\
\forall R : X \leftrightarrow Y \ \bullet \ R \in \text{dom} \ R \leftrightarrow \text{ran} \ R
\end{align*}
\]

Just like domSingleton

Trivial lemma A.5 (dom of singleton set backwards)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lDomSingletonBackwards } [X, Y] \\
\forall x : X; \ y : Y \ \bullet \ \{ x \} = \text{dom} \ (\{ (x, y) \})
\end{align*}
\]

Just like ranSingleton

Trivial lemma A.6 (ran of singleton set backwards)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lRanSingletonBackwards } [X, Y] \\
\forall x : X; \ y : Y \ \bullet \ \{ y \} = \text{ran} \ (\{ (x, y) \})
\end{align*}
\]

Trivial lemma A.7 (Non-maximal domain extension)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lNonMaxDom } [X, Y] \\
\forall A : \mathcal{P} X; \ B : \mathcal{P} Y \ \bullet \ \forall R : A \leftrightarrow B \ \bullet \ \text{dom}[X, Y] \ R = \text{dom}[A, B] \ R
\end{align*}
\]

A.3 Function spaces

Toolkit extension A.11 (Partial function point is an element)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lPfunPointIsPfunElem } [X, Y] \\
\forall x : X; \ y : Y; \ f : X \rightarrow Y \ | \ x \in \text{dom} \ f \land y = fx \ \bullet \ (x, y) \in f
\end{align*}
\]

Toolkit extension A.12 (Expose $\rightarrow$ property)

\[
\begin{align*}
\text{theorem disabled rule} & \text{ lPFunWeakening } [X, Y] \\
f \in X \rightarrow Y & \iff \\
f \in X \leftrightarrow Y \land \\
(\forall d : X; \ r1, r2 : Y \ | \ (d, r1) \in f \land (d, r2) \in f \ \bullet \ r1 = r2)
\end{align*}
\]
Trivial lemma A.8 (Expose \(\mapsto\to\) property with fresh names)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lPFunWeakeningFresh} \ [X, Y] \\
&f \in X \to Y \\
\iff \\
&f \in X \leftrightarrow Y \land \\
&(\forall df : X; rf1, rf2 : Y \mid (df, rf1) \in f \land (df, rf2) \in f \bullet rf1 = rf2)
\end{align*}
\]

The next lemma is laid out for maximal automation. It is equivalent to \(\text{dom } f = \{\} \Rightarrow f = \{\}\), which cannot be given as a rule.

Toolkit extension A.13 (Non empty PFun has non empty dom)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lEmptyDomIsEmptyPFun} \ [X, Y] \\
&\forall f : X \to Y \mid \neg f = \{\} \bullet \neg \text{dom } f = \{\}
\end{align*}
\]

Toolkit extension A.14 (dom element is specific \(\mapsto\to\) element)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lDomElemIsElemPFun} \ [X, Y] \\
&\forall x : X; f : X \to Y \mid x \in \text{dom } f \bullet (x, f x) \in f
\end{align*}
\]

Toolkit extension A.15 (Member of homogeneous \(\mapsto\to\) forms no loop)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lHomogeneousMemberNoLoop} \ [X] \\
&\forall x : X; f : X \to X \mid x \in \text{dom } f \land \neg x \in \text{ran } f \bullet \neg f x = x
\end{align*}
\]

Toolkit extension A.16 (\(\mapsto\) non-immediate member)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lNotImmediateMemberPFun} \ [X, Y] \\
&\forall f : X \to Y; x : X; y : Y \mid x \in \text{dom } f \land \neg f x = y \bullet \neg (x, y) \in f
\end{align*}
\]

Toolkit extension A.17 (\(\mapsto\to\oplus\) containment)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lPFunSubsetOplus} \ [X, Y] \\
&\forall f, g : X \to Y \mid g \subseteq f \bullet f \oplus g = f \oplus (\text{dom } g \triangleleft f)
\end{align*}
\]

Toolkit extension A.18 (\(\mapsto\to\oplus\) pointwise equivalence)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lPFunElemAbsorbsUnitOplusRight} \ [X, Y] \\
&\forall x : X; y : Y; f : X \to Y \mid (x, y) \in f \bullet f \oplus \{(x, y)\} = f
\end{align*}
\]

Toolkit extension A.19 (\(\mapsto\)dom partitions over \(<\))

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lPFunDomPartitionsPFunDRes} \ [X, Y] \\
&\forall f : X \to Y; s, t : \mathbb{P} X \mid (s, t) \text{ partition } \text{dom } f \bullet (s \triangleleft f, t \triangleleft f) \text{ partition } f
\end{align*}
\]

### A.4 Injective functions

Lemmas are toolkit theorems that are not rules (i.e., cannot be automatically applied)

Lemma A.1 (\(\mapsto\) element is part of its inverse)

\[
\begin{align*}
\textbf{theorem} \quad &\text{disabled rule lPairInvInPInj} \ [X, Y] \\
&\forall A : \mathbb{P} X; B : \mathbb{P} Y \bullet \forall f : A \leftrightarrow B \mid (x, y) \in f \bullet x \in A \land y \in B \land x = (f ^\sim) y \land (y, x) \in (f ^\sim)
\end{align*}
\]
Toolkit extension A.20 (Expose $\iff$ property)

\textbf{theorem} disabled rule lPInjWeakening \([X, Y]\)
\[ f \in X \iff Y \]
\[ \iff f \in X \rightarrow Y \land \left( \forall r : Y; \; d1, \; d2 : X \mid \langle d1, r \rangle \in f \land \langle d2, r \rangle \in f \iff d1 = d2 \right) \]

Trivial lemma A.9 (Expose $\iff$ property with fresh names)

\textbf{theorem} disabled rule lPInjWeakeningFresh \([X, Y]\)
\[ f \in X \iff Y \]
\[ \iff f \in X \rightarrow Y \land \left( \forall rf : Y; \; df1, \; df2 : X \mid \langle df1, rf \rangle \in f \land \langle df2, rf \rangle \in f \iff df1 = df2 \right) \]

Lemma A.2 (\(\iff\) result uniqueness weakening rule)

\textbf{theorem} disabled rule lPInjUniqueResWeakening \([X, Y]\)
\[ f \in X \iff Y \]
\[ \iff f \in X \rightarrow Y \land \left( \forall a, b : X \mid a \in \text{dom}[X, Y] \land b \in \text{dom}[X, Y] \land f \land a \neq b \iff (f(a), f(b)) \in _\neq [Y] \right) \]

Lemma A.3 (\(\iff\) element is part of its inverse)

\textbf{theorem} disabled rule lPInjPairIsRanPoint \([X, Y]\)
\[ \forall A : \; \mathbb{P} \; X; \; B : \; \mathbb{P} \; Y \cdot \forall f : A \rightarrow B \mid \langle x, y \rangle \in f \iff x \in A \land \; y \in B \land x \in \text{dom} f \land y \in \text{ran} f \iff x \circ x = (f \iff y) \iff \]

Lemma A.4 (\(\iff\) ran element \(_\sim\) in \(\iff\))

\textbf{theorem} disabled rule lApplyInvInDomPInj \([X, Y]\)
\[ \forall A : \; \mathbb{P} \; X; \; B : \; \mathbb{P} \; Y \cdot \forall f : A \rightarrow B \cdot \forall a : \text{ran} \; f \cdot (f \iff a) \in A \land (f \iff a) \in \text{dom} \; f \]

Toolkit extension A.21 (\(\iff\)-dom element \(\iff\)-equivalence)

\textbf{theorem} disabled rule lHomogeneousElemRanNDresPInj \([X]\)
\[ \forall x : X; \; f : X \iff X \mid x \in \text{dom} \; f \iff \text{ran} \{ \{ x \} \iff f \} = \text{ran} \; f \setminus \text{ran} \{ f \iff \}

Toolkit extension A.22 (\(\iff\)-ran element \(\iff\)-equivalence)

\textbf{theorem} disabled rule lHomogeneousElemDomNRresPInj \([X]\)
\[ \forall y : X; \; f : X \rightarrow X \mid y \in \text{ran} \; f \iff \text{dom} \; (f \iff \{ y \}) = \text{dom} \; f \setminus \{ f \iff y \}

Toolkit extension A.23 (\(\iff\)-dom element expansion)

\textbf{theorem} disabled rule lInDomInjection \([X, Y]\)
\[ \forall f : X \rightarrow Y \cdot x \in \text{dom} \; f \iff (\exists y : X \mid x \in \text{ran} \; f \iff x = (f \iff y)

Toolkit extension A.24 (\(\iff\)-ran element expansion)

\textbf{theorem} disabled rule lInRanInjection \([X, Y]\)
\[ \forall f : X \rightarrow Y \cdot y \in \text{ran} \; f \iff (\exists x : X \mid x \in \text{ran} \; f \iff x = (f \iff y)

This lemmas is similar (yet complementary) to \textbf{Z/Eves} toolkit rule \texttt{applyInverse}.
Toolkit extension A.25 (⇒_¬-application)

**theorem** disabled rule lApplyInverse2 \([X, Y]\)
\[ \forall A : \mathbb{P} X; B : \mathbb{P} Y \leadsto \forall f : A \mapsto B \bullet \forall y : \text{ran } f \bullet f((f \sim) y) = y \]

Toolkit extension A.26 (⇒ point is ⇒ element)

**theorem** disabled rule lPInjInvPointIsPInjElem \([X, Y]\)
\[ \forall f : X \mapsto Y \mid y \in \text{ran } f \land (f \sim) y = x \bullet (x, y) \in f \]

Toolkit extension A.27 (Distinct ⇒ point is not shared in ⇒)

**theorem** disabled rule lPInjPointIsNotShared \([X, Y]\)
\[ \forall x_1, x_2 : X \mid f : X \mapsto Y \mid x_1 \in \text{dom } f \land \neg x_1 = x_2 \bullet \neg (x_2, f x_1) \in f \]

Toolkit extension A.28 (⇒ following application is not ⇒ member)

**theorem** disabled rule lFollowingApplicationNoLoopHomogeneousPInj \([X]\)
\[ \forall x : X; f : X \mapsto X \mid x \in \text{dom } f \land f x \in \text{dom } f \land \neg f x = x \bullet \neg (f x, f x) \in f \]

A.5 Finiteness

[Circus tools off]
Leave enabled!

Trivial lemma A.10 (Improved version of toolkit rule crossFinite)

**theorem** disabled rule lCrossFinite2
\[ A \times B \in \mathbb{F}(C \times D) \leftrightarrow \begin{cases} A = \emptyset \lor B = \emptyset \\ \forall (A \in \mathbb{F} C \land B \in \mathbb{F} D) \end{cases} \]

[Circus tools on]

Trivial lemma A.11 (Subset of finite set is finite)

**theorem** disabled rule lFinsetSubset
\[ X \in \mathbb{P} Y \land Y \in \mathbb{F} Z \Rightarrow X \in \mathbb{F} Z \]

[Circus tools off]
Leave enabled!

Weakening rule A.9 (Inferring finite sets are subset of infinite sets)

**theorem** rule lIsFinite
\[ x \in \mathbb{F} X \Rightarrow x \in \mathbb{P} X \]

[Circus tools on]

Toolkit extension A.29 (Finite bijection subset measurement)

**theorem** disabled rule lBijectionFinite \([X, Y]\)
\[ \forall A : \mathbb{F} X; B : \mathbb{P} Y \bullet \forall f : A \mapsto B \bullet f \in A \mapsto B \land B \in \mathbb{F} Y \land \# A = \# B = \# f \]

Toolkit extension A.30 (Non-maximal cardinality equivalence)

**theorem** disabled rule lNonMaximalCardEquiv \([X]\)
\[ \forall A : \mathbb{P} X \bullet \forall S : \mathbb{F} A \bullet \# S = (\#_\_)[A] S \]

Toolkit extension A.31 (Non-maximal domain equivalence)

**theorem** disabled rule lNonMaxDomEquiv \([X, Y]\)
\[ \forall A : \mathbb{P} X; B : \mathbb{P} Y \bullet \forall R : A \leftrightarrow B \bullet \text{dom}[X, Y] R = \text{dom}[A, B] R \]
A.6 Sequences

A.6.1 Lemmas on sequences

**Theorem** disabled rule lDisjointCatMinusUnion [X]
\[ \forall S, T, R : P \ X \ | \ disjoint(S) \land (R) \land disjoint(S \setminus T) \land ((R \cup T)) \]

**Theorem** disabled rule lDisjointCatUnionMinus [X]
\[ \forall S, T, R : P \ X \ | \ disjoint(S) \land (R) \land disjoint(S \cup T) \land ((R \setminus T)) \]

**Theorem** disabled rule lDisjointCatMinus [X]
\[ \forall S, T, R : P \ X \ | \ disjoint(S) \land (R) \land disjoint(S \setminus T) \land (R) \]

**Toolkit extension A.32** (Expose seq property)

**Theorem** disabled rule lSeqWeakening [X]
\[ s \in \text{seq} X \iff s \in N \Rightarrow X \land (\exists n : N \cdot \text{dom}[\mathbb{Z}, X] s = 1 \ldots n) \]

**Toolkit extension A.33** (Expose iseq property)

**Theorem** disabled rule llSeqWeakening [X]
\[ s \in \text{iseq} X \iff s \in \text{seq} X \land s \in N \Rightarrow X \]

**Weakening rule A.10** (Expanding the (non-maximal) type of a sequence element)

**Theorem** disabled rule lSeqElemType [X]
\[ \forall A : P \ X \land \forall s : \text{seq} A \ | \ e \in s \land e \in N \times A \]

**Trivial lemma A.12** (Sequence with element is not empty)

**Theorem** disabled rule lSeqWithElemIsNonEmpty [X]
\[ \forall s : \text{seq} X \ | \ e \in s \land \lnot s = () \]

**Trivial lemma A.13** (Non-empty sequence has an element)

**Theorem** disabled rule lNonEmptySeqHasElem [X]
\[ \forall A : P \ X \land \forall s : \text{seq}_1 A \land \exists e : N \times A \land e \in s \]

**Weakening rule A.11** (Expanding the (non-maximal) type of range type of sequence element)

**Theorem** disabled rule lSeqRanElemType [X]
\[ \forall A : P \ X \land \forall s : \text{seq} A \ | \ (i, y) \in s \land y \in A \]
Toolkit extension A.34 (Non-empty sequence has strictly positive size)

**Theorem** disabled rule lSeqNonEmptySize [X]
\[ \forall s : \text{seq } X \mid \neg s = \langle \rangle \cdot 1 \leq \# s \]

Toolkit extension A.35 (Alternative for \( \text{dom } s \) when involving contraction)

**Theorem** disabled rule lSeqDomEquiv [X]
\[ \forall s : \text{seq } X \cdot \text{dom } s \setminus \{ \# s \} = 1 .. -1 + \# s \]

Toolkit extension A.36 (Sequence domain non-maximal cardinality)

**Theorem** disabled rule lNonMaxDomSeq [X]
\[ \forall A : \mathbb{P} X \cdot \forall s : \text{seq } A \cdot \text{dom}[\mathbb{Z}, A] s = 1 .. (\# s)[(\mathbb{Z} \times A)] s \]

Toolkit extension A.37 (Sequence domain non-maximal card. equivalence)

**Theorem** disabled rule lNonMaxDomSeqEquiv [X]
\[ \forall A : \mathbb{P} X \cdot \forall s : \text{seq } A \cdot \text{dom}[\mathbb{Z}, X] s = \text{dom}[\mathbb{N}, A] s \]

**Theorem** disabled rule lNonMaxDomSeqNatEquiv [X]
\[ \forall A : \mathbb{P} X \cdot \forall s : \text{seq } A \cdot \text{dom}[\mathbb{Z}, X] s = \text{dom}[\mathbb{N}, A] s \]

A.6.2 Lemmas on Sequence concatenation

Toolkit extension A.39 (Sequence \( \mapsto \) last of left seq appl.)

**Theorem** disabled rule lSeqCatLastLeftApply [X]
\[ \forall s, t : \text{seq } X \mid \neg s = \langle \rangle \land \neg t = \langle \rangle \cdot (s \mapsto t)(\# s) = \text{last } s \]

Toolkit extension A.40 (Sequence \( \mapsto \) head of right seq appl.)

**Theorem** disabled rule lSeqCatHeadRightApply [X]
\[ \forall s, t : \text{seq } X \mid \neg s = \langle \rangle \land \neg t = \langle \rangle \cdot (s \mapsto t)(1 + \# s) = \text{head } t \]

Toolkit extension A.41 (Singleton sequence membership)

**Theorem** disabled rule lUnitBelongs
\[ (i, a) \in \langle x \rangle \Leftrightarrow i = 1 \land a = x \]

Toolkit extension A.42 (Singleton sequence element membership)

**Theorem** disabled rule lUnitElemBelongs
\[ e \in \langle x \rangle \Leftrightarrow e = (1, x) \]
Toolkit extension A.43 (last-\(\_ \bowtie \_\) left side equivalence)

\textbf{theorem} disabled rule lCatLeftLastEquiv \([X]\)
\[
\forall r, s, t : \text{seq } X \mid r = s \bowtie t \land \neg s = () \Rightarrow \text{last } s = r (\# s)
\]

Toolkit extension A.44 (head-\(\_ \bowtie \_\) right side equivalence)

\textbf{theorem} disabled rule lCatRightHeadEquiv \([X]\)
\[
\forall r, s, t : \text{seq } X \mid r = s \bowtie t \land \neg t = () \Rightarrow \text{head } t = r (1 + \# s)
\]

Toolkit extension A.45 (Concatenation size equivalence)

\textbf{theorem} disabled rule lSizeCatEquals \([X]\)
\[
\forall s, t : \text{seq } X \mid r = s \bowtie t \Rightarrow \# s + \# t = \# r
\]

Toolkit extension A.46 (\textit{dom} of sequence concatenation)

\textbf{theorem} disabled rule lDomCat \([X]\)
\[
\forall s, t : \text{seq } X \Rightarrow \text{dom}(s \bowtie t) = 1..\# s + \# t
\]

[Circus tools off]

Weakening rule A.12 (\(\_ \bowtie \_\) max type)

\textbf{theorem} rule lSeqInCatMaxType \([X]\)
\[
\forall s : \text{seq } X; x : X \cdot s \bowtie \langle x \rangle \in \mathbb{P}(\mathbb{Z} \times X)
\]

[Circus tools on]
[Circus tools off]

Weakening rule A.13 (\(\_ \bowtie \_\) injective sequence to right)

\textbf{theorem} rule lInISeqCatRight \([X]\)
\[
\forall A : \mathbb{P} X \cdot \forall s : \text{iseq } A; a : A \mid \neg a \in \text{ran } s \Rightarrow s \bowtie \langle a \rangle \in \text{iseq } A
\]

[Circus tools on]

A.6.3 Lemmas on Sequence Decomposition

We need this lemma to prove the next rewrite rule, which appear often when one uses injective sequences. Further lemmas such as

\[(i, a) \in \text{tail } s \Rightarrow (1 + i, a) \in s\]

are trivial and encouraged (TODO: add later)

Toolkit extension A.47 (tail element is in seq domain)

\textbf{theorem} disabled rule lTailElemInDom \([X]\)
\[
\forall Y : \mathbb{P} X \cdot \forall s : \text{seq } Y \mid \neg s = () \land (i, a) \in \text{tail } s \Rightarrow 1 + i \in \text{dom } s
\]

[Circus tools on]

Leave enabled!
Weakening rule A.14 (\textit{tail} preserves sequence injectiveness)

\textbf{theorem} rule lTailISeqIsISeq \[ X \]
\[ \forall Y : P \ X \bullet \forall s : \text{iseq} \ Y \ | \ s = \langle \rangle \bullet \text{tail} \ s \in \text{iseq} \ Y \]

[Circus tools on]

Toolkit extension A.48 (Domain of non-empty sequence’s \textit{tail})

\textbf{theorem} disabled rule lTailDom \[ X \]
\[ \forall s : \text{seq} \ X \ | \ s \neq \langle \rangle \bullet \text{dom} (\text{tail} \ s) = 1..1 + \# s \]

This is useful when we have sequences as sets that are shrunk due to updates over the sequence.

Toolkit extension A.49 (Sequence \textit{tail} element in range of \textit{tail})

\textbf{theorem} disabled rule lTailElemInRanTail \[ X \]
\[ \forall s : \text{seq} \ X \ | \ s \neq \langle \rangle \land (i, y) \in \text{tail} \ s \bullet y \in \text{ran} (\text{tail} \ s) \]

Toolkit extension A.50 (Sequence \textit{tail} is closed under seq range)

\textbf{theorem} disabled rule lTailRanSubset \[ X \]
\[ \forall s : \text{seq} \ X \ | \ s \neq \langle \rangle \land y \in \text{ran} (\text{tail} \ s) \bullet y \in s \]

Toolkit extension A.51 (Non singleton sequence cardinality)

\textbf{theorem} disabled rule lNonSingletonSeqCard \[ X \]
\[ \forall A : P \ X \bullet \forall s : \text{seq} \ A \bullet \# s > 1 \iff \neg s = \langle \rangle \land \neg (\exists e : A \bullet s = \langle e \rangle) \]

Non-singleton sequence property

\[
\begin{align*}
\text{NonSingletonSeqInducProperty} [X] \\
\text{seq} A \subseteq \{ r : \text{seq} A \mid \# r > 1 \Rightarrow (\exists i, j : \text{seq} A \bullet r = i \mathbin{\bowtie} j) \}
\end{align*}
\]

Lemma A.5 (Z/Eves induction layout theorem for non singleton induc.)

\textbf{theorem} disabled lNonSingletonSeqZEvesInduc \[ X \]
\[ \forall A : P \ X \bullet \text{NonSingletonSeqInducProperty} [X] \]

Toolkit extension A.52 (Non-singleton sequences have mid-points)

\textbf{theorem} disabled lNonSingletonSeqMidPoint \[ X \]
\[ \forall A : P \ X \bullet \forall s : \text{seq} A \bullet \# s > 1 \bullet \exists a, b : \text{seq} A \bullet s = a \mathbin{\bowtie} b \]

[Circus tools off]

Toolkit extension A.53 (sequence right-inductive decomp. prop.)

\textbf{theorem} lSeqRightInducDecompZEvesInduc \[ X \]
\[ \text{seq} A \subseteq \{ t : \text{seq} X \mid x : X \bullet t \mathbin{\bowtie} \langle x \rangle \} \]

[Circus tools on]

Toolkit extension A.54 (Non-empty sequence right-inductive decomposition)

\textbf{theorem} disabled rule lSeqRightInducDecomp \[ X \]
\[ s \in \text{seq} \ X \iff (\exists t : \text{seq} X \bullet x : X \bullet s = t \mathbin{\bowtie} \langle x \rangle) \]
A.6.4 Sequence manipulation

[Circus tools off]

Toolkit extension A.55 (Range of sequence filtering)

\[
\text{theorem rule lRanFilter } [X] \\
\forall A : \mathbb{P}(X) \bullet \forall s : \text{seq } A; \; S : \mathbb{P}(A) \bullet \text{ran}(s \upharpoonright S) = S \cap \text{ran } s
\]

[Circus tools on]

A.6.5 Sequence mapping

[Circus tools off]

Toolkit extension A.56 (Sequence mapping remains sequence when within the map)

\[
\text{theorem rule lSeqMapIsSeq } [X, Y] \\
\forall s : \text{seq } X; \; f : X \twoheadrightarrow Y \; | \; \text{ran } s \in \mathbb{P}(\text{dom } f) \bullet s \circ f \in \text{seq } Y
\]

[Circus tools on]
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Schemas</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Theorems</td>
<td>99</td>
<td>192</td>
</tr>
<tr>
<td>Proofs</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>102</strong></td>
<td><strong>386</strong></td>
</tr>
</tbody>
</table>

Table A.1: Summary of Z declarations for Chapter A.
Appendix B

Proof scripts

section basic_defs_proofs parents basic_defs

B.1 Z Section Basic data types

Z Proof Section B.1.1 (Basic data types (see 1.1, p. 1))

proof[basic_defs_axiom1_we_fsb_axiom]
  instantiate maxpid == 1;
  prove;

proof[basic_defs_axiom2_we_fsb_axiom]
  invoke PID;
  instantiate nullpid == maxpid + 1;
  prove;
  use lMaxPIDPositive;
  simplify;

proof[PID_we_fsb_horiz_def]
  instantiate PID == {};
  prove;

proof[GPID_we_fsb_horiz_def]
  instantiate GPID == {};
  prove;

proof[lMaxPIDPositive]
  use maxpid$declaration;
  apply inNat1;
  simplify;
proof[PIDNotEmpty]
  apply extensionality;
  invoke PID;
  prove by rewrite;
  instantiate x == 1;
  rewrite;

proof[MinPIDValue]
  with enabled (PID) prove by reduce;

proof[gMaxpidMaxType]
  rewrite;

proof[MaxpidIsPID]
  with enabled (PID) prove by reduce;

proof[NullpidBound]
  use dNullPID[p := maxpid];
  invoke PID;
  rewrite;

proof[NullPIDDisjoint]
  use dNullPID;
  rearrange;
  rewrite;

proof[gPIDMaxType]
  invoke PID;
  rewrite;

proof[gGPIDMaxType]
  invoke GPID;
  invoke PID;
  prove by rewrite;

proof[NullIsGPID]
  invoke GPID;
  rewrite;
proof[NullIsNotPID]
    use lNullpidBound;
    invoke PID;
    apply inRange;
    simplify;

proof[PIDIIsGPID]
    with enabled (GPID) prove by reduce;

proof[MinPIDIIsGPID]
    rewrite;

proof[MaxPIDIIsGPID]
    rewrite;

proof[NonNullGPIDIspID]
    invoke GPID;
    prove by rewrite;

section vm_proofs parents vm

B.2 Z Section VM processes operations preconditions
Z Proof Section B.2.1 (VM processes operations preconditions (see 2.2.2, p. 6))

proof[VMCreateFSB]
    split freeVM = ∅;
    prove by reduce;
    cases;
    instantiate p ! == nullpid;
    rewrite;
    next;
    apply extensionality to predicate freeVM = {};
    prenex;
    instantiate p! == x;
    with enabled (disjointCat, distributeCapOverCupLeft, distributeCapOverCupRight, lElemUnionAbsorbDiffRight, lInterAbsorbDiffRight) prove;
    apply extensionality;
    instantiate x₀ == x;
    prove;
    next;

proof[VMRunFSB]
split \( p_? \in createdVM; 
with enabled (disjointCat, distributeCapOverCupLeft, 
distributeCapOverCupRight, lElemUnionAbsorbDiffRight,
InterAbsorbDiffRight) prove by reduce;
apply extensionality;
instatiate \( x_0 == p_?; \)
prove;

section ascheduler_proofs parents ascheduler

B.3 Abstract scheduler

No proofs needed

B.4 Z Section Abstract scheduler properties

Z Proof Section B.4.1 (Abstract scheduler properties (see 3.2, p. 11))

proof[ScheduledPIDDisjoint]
invoke AScheduler;
invoke GPID;
prove by rewrite;
split current = nullpid;
with enabled (disjointCat) prove by rewrite;
cases;
    split blockedAS \( \cup \) (freeAS \( \cup \) readyAS) = PID;
simplify;
equality substitute PID;
rearrange;
rewrite;
rewrite;
next;
    split blockedAS \( \cup \) (freeAS \( \cup \) (readyAS \( \cup \) \{current\})) = PID;
simplify;
equality substitute PID;
rearrange;
rewrite;
rewrite;
rewrite;
next;

B.5 Z Section Abstract scheduler operations preconditions

Z Proof Section B.5.1 (Abstract scheduler operations preconditions (see 3.4, p. 11))

proof[ASchedulerInitFSB]
with enabled (disjointCat) prove by reduce;


proof[ACreate0FSB]
with disabled (PID) with enabled (disjointCat) prove by reduce;
apply extensionality to predicate freeAS = {};
with normalization prove by rewrite;
cases;
instantiate p! == x;
rearrange;
apply lElemUnionAbsorbDiffRight to expression \{x\} \cup (freeAS \setminus \{x\});
with enabled (disjointCat) rewrite;
apply extensionality to predicate blockedAS \cap freeAS = {};
apply extensionality to predicate blockedAS \cap (freeAS \setminus \{x\}) = {};
prenex;
rewrite;
instantiate x_1 == x_0;
rewrite;
apply distributeCapOverCupRight to expression (readyAS \cup \{x\}) \cap (blockedAS \cup (freeAS \setminus \{x\}));
apply distributeCapOverCupRight to expression readyAS \cap (blockedAS \cup freeAS);
rewrite;
apply distributeCapOverCupRight to expression blockedAS \cap (readyAS \cup \{x\});
rewrite;
apply distributeCapOverCupLeft to expression (readyAS \cup \{x\}) \cap (freeAS \setminus \{x\});
rewrite;
apply extensionality to predicate freeAS \cap readyAS = {};
apply extensionality to predicate readyAS \cap (freeAS \setminus \{x\}) = {};
prenex;
rewrite;
split x \in \text{blockedAS};
rewrite;
cases;
instantiate x_2 == x;
rewrite;
next;
instantiate x_5 == x_1;
rewrite;
next;
with enabled (disjointCat) prove by rewrite;
instantiate p! == x;
apply lElemUnionAbsorbDiffRight to expression \{x\} \cup (freeAS \setminus \{x\});
rewrite;
apply distributeCapOverCupRight to expression (readyAS \cup \{x\}) \cap (blockedAS \cup (freeAS \setminus \{x\}));
apply distributeCapOverCupRight to expression readyAS \cap (blockedAS \cup freeAS);
rewrite;
apply extensionality to predicate blockedAS \cap (freeAS \setminus \{x\}) = {};
apply extensionality to predicate blockedAS \cap freeAS = {};
prenex;
rewrite;
instantiate x_1 == x_0;
rearrange;
rewrite;
apply distributeCapOverCupRight to expression blockedAS \cap (readyAS \cup \{x\});
rewrite;
apply distributeCapOverCupLeft to expression (readyAS \cup \{x\}) \cap (freeAS \setminus \{x\});
rewrite;
split x \in \text{blockedAS};
rewrite;
cases;
instantiate x_1 == x;
prove by rewrite;
next;
apply extensionality to predicate freeAS \cap readyAS = {};
apply extensionality to predicate readyAS \cap (freeAS \setminus \{x\}) = {};
prenex;
rewrite;
proof\[ADispatch0FSB\]
with disabled (PID) with enabled (disjointCat) prove by reduce:
apply extensionality to predicate readyAS \cap (blockedAS \cup freeAS) = \{\};
apply extensionality to predicate readyAS = \{\};
prove by rewrite;
instantiate current' == x;
instantiate x_0 == x;
rewrite;
apply \text{lElemUnionAbsorbDiffRight};
rewrite;
apply extensionality to predicate (blockedAS \cup freeAS) \cap (readyAS \setminus \{x\}) = \{\};
prove by rewrite;
instantiate x_1 == x_0;
prove by rewrite;

proof\[ABlock0FSB\]
with disabled (PID) with enabled (disjointCat) prove by reduce:
distributeCapOverCupRight;
rewrite;
distributeCapOverCupRight to expression readyAS \cap (freeAS \cup \{current\});
rewrite;

proof\[ATimeOut0FSB\]
with disabled (PID) with enabled (disjointCat) prove by reduce:
distributeCapOverCupRight to expression (blockedAS \cup freeAS) \cap (readyAS \cup \{current\});
rewrite;


proof[AWakeUp0FSB]

with disabled (PID) with enabled (disjointCat) prove by reduce;
apply lElemUnionAbsorbDiffRight to expression \{p?\} \cup (\text{blockedAS} \setminus \{p\})
rewrite;
with normalization rewrite;
cases;
  with enabled (disjointCat) prove by rewrite;
  apply distributeCapOverCupRight to expression (\text{readyAS} \cup \{p?\}) \cap (\text{freeAS} \cup (\text{blockedAS} \setminus \{p\})
  apply distributeCapOverCupRight to expression \text{readyAS} \cap (\text{blockedAS} \cup \text{freeAS})
  rewrite;
  cases;
    apply extensionality to predicate \text{freeAS} \cap (\text{blockedAS} \setminus \{p\}) = \{
    apply extensionality to predicate \text{blockedAS} \cap \text{freeAS} = \{
    prove by rewrite;
    instantiate x_0 == x;
    rewrite;
  next;
    apply distributeCapOverCupRight to expression \text{freeAS} \cap (\text{readyAS} \cup \{p\});
    apply extensionality to predicate \text{blockedAS} \cap \text{freeAS} = \{
    prove by rewrite;
    instantiate x == p?
    prove by rewrite;
  next;
    apply distributeCapOverCupLeft to expression (\text{readyAS} \cup \{p\}) \cap (\text{blockedAS} \setminus \{p\})
    rewrite;
    apply extensionality to predicate \text{readyAS} \cap (\text{blockedAS} \setminus \{p\}) = \{
    apply extensionality to predicate \text{blockedAS} \cap \text{readyAS} = \{
    prove by rewrite;
    instantiate x_0 == x;
    prove by rewrite;
  next;
    cases;
    apply extensionality to predicate \text{freeAS} \cap (\text{blockedAS} \setminus \{p\}) = \{
    apply extensionality to predicate \text{blockedAS} \cap \text{freeAS} = \{
    prove by rewrite;
    instantiate x_0 == x;
    prove by rewrite;
  next;
    apply distributeCapOverCupLeft to expression (\text{readyAS} \cup \{p\}) \cap (\text{freeAS} \cup (\text{blockedAS} \setminus \{p\})
    rewrite;
    cases;
      apply extensionality to predicate \text{blockedAS} \cap \text{freeAS} = \{
      instantiate x == p?
      prove by rewrite;
    next;
      apply extensionality to predicate \text{readyAS} \cap (\text{freeAS} \cup (\text{blockedAS} \setminus \{p\})) = \{
      apply extensionality to predicate \text{readyAS} \cap (\text{blockedAS} \cup \text{freeAS}) = \{
      prove by rewrite;
      instantiate x_0 == x;
      prove by rewrite;
    next;
proof[ADestroyCurrentFSB]
  with disabled (PID) with enabled (disjointCat) prove by reduce;
cases;
  apply distributeCapOverCapRight;
  rewrite;
next;
  apply distributeCapOverCapRight;
  rewrite;
  apply extensionality to predicate readyAS ∩ (freeAS ∪ {p?}) = {};
  apply extensionality to predicate freeAS ∩ readyAS = {};
  prove by rewrite;
  instantiate x_0 == x;
  prove by rewrite;
next;
proof\[A\text{DestroyReadyFSB}\]
with disabled (PID) with enabled (disjointCat) prove by reduce;
apply lCupAssociatesBackwards to expression freeAS \cup \{p?\} \cup (readyAS \setminus \{p?\});
apply distributeCapOverCupLeft to expression (blockedAS \cup (freeAS \cup \{p?\})) \cap (readyAS \setminus \{p?\}); rewrite;
apply distributeCapOverCupLeft to expression \{p\} \cup (readyAS \setminus \{p?\}); rewrite;
apply lCupAssociatesBackwards to expression \{p\} \cup (readyAS \setminus \{p\}); rewrite;
with normalization rewrite;
cases;
with enabled (disjointCat) prove by rewrite;
cases;
apply distributeCapOverCapRight to expression blockedAS \cap (freeAS \cup \{p?\});
apply distributeCapOverCapRight to expression readyAS \cap (blockedAS \cup freeAS); rewrite;
apply extensionality to predicate blockedAS \cap readyAS = \{};
instantiate x == p?;
rewrite;
next;
apply distributeCapOverCapRight to expression readyAS \cap (blockedAS \cup freeAS); rewrite;
apply extensionality to predicate blockedAS \cap readyAS = \{};
apply extensionality to predicate blockedAS \cap (readyAS \setminus \{p\}) = \{};
prenex;
rewrite;
instantiate x_0 == x;
rewrite;
next;
apply distributeCapOverCapRight to expression readyAS \cap (blockedAS \cup freeAS); rewrite;
apply extensionality to predicate freeAS \cap (readyAS \setminus \{p\}) = \{};
apply extensionality to predicate freeAS \cap readyAS = \{};
prenex;
rewrite;
instantiate x_0 == x;
rewrite;
next;
cases;
apply distributeCapOverCapRight to expression blockedAS \cap (freeAS \cup \{p\});
apply distributeCapOverCapRight to expression readyAS \cap (blockedAS \cup freeAS); rewrite;
apply extensionality to predicate blockedAS \cap readyAS = \{};
instantiate x == p?;
rewrite;
next;
apply extensionality to predicate readyAS \cap (blockedAS \cup freeAS) = \{};
apply extensionality to predicate blockedAS \cap (readyAS \setminus \{p\}) = \{};
prenex;
instantiate x_0 == x;
rewrite;
next;
apply distributeCapOverCapRight to expression readyAS \cap (blockedAS \cup freeAS); rewrite;
apply extensionality to predicate freeAS \cap (readyAS \setminus \{p\}) = \{};
apply extensionality to predicate freeAS \cap readyAS = \{};
prenex;
instantiate x_0 == x;
rewrite;
next;

72
\textbf{proof}[\texttt{ADestroyBlockedFSB}] \\
with disabled (PID) with enabled (disjointCat) prove by reduce; 
apply distributeCapOverCupLeft to expression \((\text{freeAS} \cup \{p\}) \cap (\text{blockedAS} \setminus \{p\})\); 
apply lElemUnionAbsorbDiffRight to expression \(\{p\} \cup (\text{blockedAS} \setminus \{p\})\); 
rewrite; 
apply distributeCapOverCupRight to expression \(\text{readyAS} \cap (\text{blockedAS} \cup \text{freeAS})\); 
rewrite; 
with normalization rewrite; 
cases; 
with enabled (disjointCat) prove by rewrite; 
apply extensionality to predicate \(\text{freeAS} \cap (\text{blockedAS} \setminus \{p\}) = \{}\); 
prenex; 
instantiate \(x\_0 == x\); 
rewrite; 
next; 
apply extensionality to predicate \(\text{freeAS} \cap (\text{blockedAS} \setminus \{p\}) = \{}\); 
prenex; 
instantiate \(x\_0 == x\); 
rewrite; 
next; 

\item

\textbf{proof}[\texttt{ADispatchErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

\textbf{proof}[\texttt{ATimeOutErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

\textbf{proof}[\texttt{ABlockErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

\textbf{proof}[\texttt{AWakeUpErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

\textbf{proof}[\texttt{ACreateErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

\textbf{proof}[\texttt{ADestroyErrFSB}] \\
with disabled (AScheduler, PID) prove by reduce; 
\item

73
proof[ADispatchIsTotal]
  use ADispatch0FSB;
  use ADispatchErrFSB;
  with disabled (AScheduler, PID) prove by reduce;
  split current = nullpid ⇒ readyAS = {};
  rewrite;
  cases;
    instantiate p==0_! == 0;
    rewrite;
    next;
    rewrite;
    instantiate current' == current_0';
    rewrite;
    next;

proof[ATimeOutIsTotal]
  use ATimeOut0FSB;
  use ATimeOutErrFSB;
  with disabled (AScheduler, PID) prove by reduce;
  split current = nullpid;
  rewrite;
  instantiate p! == 0;
  rewrite;

proof[ABlockIsTotal]
  use ABlock0FSB;
  use ABlockErrFSB;
  with disabled (AScheduler, PID) prove by reduce;
  split current = nullpid;
  rewrite;
  instantiate p! == 0;
  rewrite;

proof[AWakeUpIsTotal]
  use AWakeUp0FSB;
  use AWakeUpErrFSB;
  with disabled (AScheduler, PID) prove by reduce;
  split p? ∈ blockedAS;
  rewrite;
proof[ACreateIsTotal]
  use ACreate0FSB;
  use ACreateErrFSB;
  with disabled (AScheduler, PID) prove by reduce;
  split freeAS = {};
  rewrite;
  cases;
    instantiate p_0! == 0;
    rewrite;
  next;
  rearrange;
    instantiate p_1! == p!;
    rewrite;
  next;

■
proof[ADestroyIsTotal]
invoke ADestroy;
invoke ADestroyErr;
split p? ∈ freeAS;
cases;
use ADestroyErrFSB;
with disabled (AScheduler, ADestroy0, PID) prove by reduce;
instantiate aserr_0 == aserr!, blockedAS_0′ == blockedAS′, current_0′ == current′,
freeAS_0′ == freeAS′, readyAS_0′ == readyAS′;
rewrite;
next;
invoke ASchedOkay;
rewrite;
invoke ADestroy0;
split p? = current;
cases;
use ADestroyCurrentFSB;
with disabled (ADestroyReady, ADestroyBlocked, AScheduler, PID) prove by reduce;
instantiate blockedAS_0′ == blockedAS′, current_0′ == current′,
freeAS_0′ == freeAS′, readyAS_0′ == readyAS′;
rewrite;
next;
split p? ∈ readyAS;
cases;
use ADestroyReadyFSB;
with disabled (ADestroyCurrent, ADestroyBlocked, AScheduler, PID) prove by reduce;
instantiate blockedAS_0′ == blockedAS′, current_0′ == current′,
freeAS_0′ == freeAS′, readyAS_0′ == readyAS′;
rewrite;
next;
split p? ∈ blockedAS;
cases;
use ADestroyBlockedFSB;
with disabled (ADestroyReady, ADestroyCurrent, AScheduler, PID) prove by reduce;
instantiate blockedAS_0′ == blockedAS′, current_0′ == current′,
freeAS_0′ == freeAS′, readyAS_0′ == readyAS′;
rewrite;
next;
use lScheduledPIDDisjoint;
invoke ASchedTotalSig;
prove by rewrite;

section scj_process_proofs parents scj_process

B.6 Z Section Event handlers proofs

Z Proof Section B.6.1 (Event handlers proofs (see 4.1, p. 15))

proof[IDLE_PID_WITNESSType]
with enabled (PID) prove by reduce;

■
proof[HandlerSetInitFSB]
  instantiate meh' == { IDLE_PID_WITNESS }, peh' == { IDLE_PID_WITNESS },
  aeh' == ∅, oseh' == ∅, freeHS' == PID \ { IDLE_PID_WITNESS }, idle' == IDLE_PID_WITNESS
  with enabled (disjointCat, HandlerSet0, lElemUnionAbsorbDiffRight) prove by reduce;

proof[AddHandlerFSB]
  prove by reduce;
  split p? ∈ meh;
  rewrite;
  cases;
  instantiate meh' == meh, peh' == peh, aeh' == aeh, oseh' == oseh,
  freeHS' == freeHS, idle' == idle;
  prove;
  next;
  apply lElemUnionAbsorbDiffRight;
  apply lDisjointCatUnionMinus;
  with enabled (disjointCat) prove;
  invoke HandlerSet0;
  instantiate aeh' == aeh, oseh' == oseh, peh' == peh \ { p? }, idle' == idle;
  with enabled (disjointCat) prove;
  apply distributeCapOverCupLeft ;
  apply distributeCapOverCupRight;
  rewrite;
  apply extensionality to predicate aeh \ (freeHS \ (oseh \ peh)) = PID;
  instantiate y == p?;
  rewrite;
  next;
proof[RemoveHandlerFSB]
prove by reduce;
split \( p? \in \text{freeHS} \);
rewrite;
cases;
instantiate meh' == meh, peh' == peh, aeh' == aeh, oseh' == oseh,
freeHS' == freeHS, idle' == idle;
prove;
next;
split \( \neg p? = \text{idle} \);
rewrite;
apply iElemUnionAbsorbDiffRight;
apply iDisjointCatMinusUnion;
invoke HandlerSet0;
with enabled (disjointCat, distributeCapOverCupLeft,
distributeCapOverCupRight, lInterAbsorbDiffRight) prove;
apply extensionality to predicate aeh \cup (freeHS \cup (oseh \cup peh)) = PID;
instantiate y == \( p? \);
with normalization rewrite;
cases;
instantiate aeh' == aeh \setminus \{ p? \}, oseh' == oseh, peh' == peh , idle' == idle;
with enabled (lInterAbsorbDiffRight) prove;
apply extensionality to predicate oseh \cup (peh \cup (aeh \setminus \{ p? \})) = aeh \cup (oseh \cup peh) \setminus \{ p? \};
apply extensionality to predicate aeh \cap peh = \{\};
apply extensionality to predicate aeh \cap oseh = \{\};
prove;
instantiate x_0 == x;
instantiate x_1 == x;
prove;
next;
rewrite;
cases;
instantiate aeh' == aeh, oseh' == oseh \setminus \{ p? \}, peh' == peh, idle' == idle ;
with enabled (lInterAbsorbDiffRight) prove;
apply extensionality to predicate aeh \cup (peh \cup (oseh \setminus \{ p? \})) = aeh \cup (oseh \cup peh) \setminus \{ p? \};
prove;
apply extensionality to predicate oseh \cap peh = \{\};
instantiate x_0 == x;
prove;
next;
rewrite;
instantiate aeh' == aeh, oseh' == oseh, peh' == peh \setminus \{ p? \}, idle' == idle;
with enabled (lInterAbsorbDiffRight) prove;
apply extensionality to predicate aeh \cup (oseh \cup (peh \setminus \{ p? \})) = aeh \cup (oseh \cup peh) \setminus \{ p? \};
prove;
next;
\textbf{proof}[\textit{RemoveAperiodicHandlersFSB}]  
prove by reduce;  
\begin{itemize}
\item instantiate \texttt{aeh} \( = \emptyset \), \texttt{oseh} \( = \texttt{oseh} \), \texttt{peh} \( = \texttt{peh} \), \texttt{freeHS} \( = \texttt{freeHS} \cup \texttt{aeh} \), \texttt{idle} \( = \texttt{idle} \);
\item with enabled \( \texttt{IDisjointCatMinusUnion} \) with disabled \( \texttt{cupCommutes} \) rewrite; invoke \texttt{HandlerSet0};
\item with enabled \( \texttt{disjointCat}, \texttt{distributeCapOverCupRight}, \texttt{distributeCapOverCupLeft} \) prove;
\item apply \texttt{distributeDiffOverCupLeft};
\item apply extensionality to predicate \texttt{oseh} \( \cup \texttt{peh} = \texttt{oseh} \cup \texttt{peh} \setminus \texttt{aeh} \);
\item with normalization prove;
\item cases;
\item apply extensionality to predicate \texttt{aeh} \( \cap \texttt{oseh} = \{\} \);
\item instantiate \texttt{x} \( \_0 \) \( = \texttt{x} \);
\item prove;
\item next;
\item apply extensionality to predicate \texttt{aeh} \( \cap \texttt{peh} = \{\} \);
\item instantiate \texttt{x} \( \_0 \) \( = \texttt{x} \);
\item prove;
\item next;
\end{itemize}

\section*{B.7 \textit{Z} Section \textit{Priority scheduler proofs}}

\textit{Z} Proof Section B.7.1 (\textit{Priority scheduler proofs} (see 4.2, p. 17))

\textbf{proof}[\textit{MIN\_PRIO\_geq1}]  
use \texttt{MIN\_PRIO} declaration;
apply \texttt{inNat1};
rewrite;
\[
\]

\textbf{proof}[\textit{MAX\_PRIO\_geq1}]  
use \texttt{MAX\_PRIO} declaration;
apply \texttt{inNat1};
rewrite;
\[
\]

\textbf{proof}[\textit{PRIORITY\_MaxType}]  
invoke \texttt{PRIORITY};
rewrite;
\[
\]

\textbf{proof}[\textit{PRIORITY\_Type}]  
invoke \texttt{PRIORITY};
rewrite;
\[
\]

\textbf{proof}[\textit{lPriorityNonEmpty}]  
apply extensionality;
invoke \texttt{PRIORITY};
\texttt{instantiate} \texttt{x} \( = \texttt{MIN\_PRIO} \);
rewrite;
\[
\]

79
proof[PRIQ_WITNESSMaxType]
  invoke PRIQ_WITNESS;
  apply overrideInPowerCross;
  prove;
-

proof[PRIQ_WITNESSRelType]
  invoke PRIQ_WITNESS;
  apply overrideInRel;
  prove;
-

proof[PRIQ_WITNESSPfunType]
  invoke PRIQ_WITNESS;
  apply overrideInPfun;
  prove;
-

proof[PRIQ_WITNESSElement]
  invoke PRIQ_WITNESS;
  with enabled (oplusDef, unitSeqDef) prove;
-

proof[PRIQ_WITNESSdom]
  invoke PRIQ_WITNESS;
  invoke PRIORITY;
  with enabled (cupSubsetLeft, dMinPrio) prove;
-

proof[PRIQ_WITNESSSType]
  apply inSeq1;
  invoke (seq_);
  prove;
  cases;
  instantiate n == MAX_PRIQ;
  use lNonMaxDom|Z, P(Z × Z)|[A := Z, B := iseq PID, R := PRIQ_WITNESS];
  prove;
  invoke PRIORITY;
  with enabled (dMinPrio) prove;
  next;
  apply extensionality;
  instantiate x == (MIN_PRIQ, ⟨ IDLE_PID_WITNESS ⟩ );
  with enabled (nullSeqDef, dMinPrio) prove;
  next;
-
proof[PRIO\_WITNESS\_IDLE\_WITNESSEquiv]
apply extensibility;
with enabled (dlBigCupAsBigU, dlInBigU) prove;
cases;
with enabled (inRanFunction) prove;
apply applyComp;
prove;
split \!_0 \in PRIORITY \land PRIO\_WITNESS \!_0 \in \mathbb{Z} \leftrightarrow \mathbb{Z};
rewrite;
cases;
apply domComp;
rewrite;
invoke PRIO\_WITNESS;
apply applyOverride;
prove;
next;
apply domComp;
rewrite;
next;
invoke \mathbb{R} \_WITNESS;
invoke PRIORITY;
apply applyOverride;
with enabled (dMinPrio) prove;
next;

proof[PriorityScheduler0InitFSB]
instantiate current' == nullpid, readyAS' == \{ IDLE\_PID\_WITNESS \}, blockedAS' == {},
freeHS' == PID \setminus \{ IDLE\_PID\_WITNESS \},
freeAS' == PID \setminus \{ IDLE\_PID\_WITNESS \},
idle' == IDLE\_PID\_WITNESS,
meh' == \{ IDLE\_PID\_WITNESS \},
peh' == \{ IDLE\_PID\_WITNESS \},
aeh' == {},
oseh' == {},
prio' == PRIO\_WITNESS;

with enabled (HandlerSet0, disjointCat, \{ElemUnion\_Absorb\_DiffRight\}) prove by reduce;
with enabled (PRIO\_WITNESS) reduce;

proof[lCollectPIDSIsTotal]
use collectPIDS$declaration;
prove;

proof[lFlattenPIDSIsTotal]
use flattenPIDS$declaration;
prove;
proof[fPRIOResMaxRelType]
  use applyInRanPfun[\Z, P(\Z \times \Z)][A := PRIORITY, B := seq \Z, f := prio, a := pr];
prove;

proof[fPRIOResMaxPfunType]
  use applyInRanPfun[\Z, P(\Z \times \Z)][A := PRIORITY, B := seq \Z, f := prio, a := pr];
prove;

proof[fPSPrioResSeqMaxType]
  use applyInRanPfun[\Z, P(\Z \times \Z)][A := PRIORITY, B := seq \Z, f := prio, a := pr];
prove;

proof[fPSPrioResSeqType]
  use applyInRanPfun[\Z, P(\Z \times \Z)][A := PRIORITY, B := seq PID, f := prio, a := pr];
prove;

proof[fPSPrioResISeqType]
  use applyInRanPfun[\Z, P(\Z \times \Z)][A := PRIORITY, B := iseq PID, f := prio, a := pr];
prove;

Now to the lemmas...

proof[lVMAllAllocated]
  invoke VM;
  with enabled (disjointCat) rewrite;
  equality substitute PID;
prove;

proof[lHSFreeIsUnknown]
  invoke HandlerSet;
  with enabled (disjointCat) rewrite;
  equality substitute PID;
  apply extensionality;
  instantiate x == p?;
prove;

proof[fVMInvariant]
  prove by reduce;

proof[fPSFreeInvariant]
  invoke PriorityScheduler;
  simplify;
\[ \text{proof}[\text{fASInvariant}] \]
invoke AScheduler;
simplify;

\[ \text{proof}[\text{fASInvariantFreeNullDisjoint}] \]
invoke AScheduler;
rearrange;
with enabled (disjointCat, distributeCapOverCupLeft, distributeCapOverCupRight, lInterAbsorbDiffRight) rewrite;
apply extensionality to predicate blockedAS \cap \text{freeAS} = \{\};
instantiate \( x = p ? \);
rewrite;

\[ \text{proof}[\text{fASInvariantFreeNonNullDisjoint}] \]
invoke AScheduler;
invoke GPID;
with enabled (disjointCat, distributeCapOverCupLeft, distributeCapOverCupRight, lInterAbsorbDiffRight) prove;
apply extensionality to predicate blockedAS \cap \text{freeAS} = \{\};
instantiate \( x = p ? \);
rewrite;

\[ \text{proof}[\text{fASInvariantRewrittenNullPartition}] \]
invoke AScheduler;
rewrite;

\[ \text{proof}[\text{fASInvariantRewrittenNonNullPartition}] \]
invoke AScheduler;
rewrite;

\[ \text{proof}[\text{fHS0Invariant}] \]
invoke HandlerSet0;
rewrite;

\[ \text{proof}[\text{fHSInvariant}] \]
invoke HandlerSet;
rewrite;
proof[PSAddPropFreeIsUnknown]
use lHSSFreeIsUnknown;
  invoke;
  rearrange;
  simplify;
  invoke HandlerSet0;
  rewrite;
  split aeh ∪ (oseh ∪ peh) = meh;
  simplify;
  equality substitute meh;
  with enabled (disjointCat, distributeCapOverCupRight) rewrite;
  cases;
  split blockedAS ∩ freeAS = {};
  simplify;
  apply extensionality to predicate blockedAS ∩ freeAS = {};
  instantiate x == p?;
  prove;
  next;
  apply extensionality to predicate freeVM ∩ usedVM = {};
  instantiate x == p?;
  rewrite;
  next;
proof[lCollectPIDSDom]
apply dCollectPIDS;
  rewrite;
  apply extensionality;
  prove;
  cases;
  apply inDom to predicate x ∈ dom {x_0 : dom prio • (x_0, (ran (prio x_0)))};
  with normalization with enabled (lRelWeakening) rewrite;
  next;
  apply inDom to predicate y ∈ dom {x : dom prio • (x, (ran (prio x)))};
  with normalization with enabled (lRelWeakening) rewrite;
  next;
proof[lCollectPIDSApply]
apply dCollectPIDS;
  rewrite;
  use pairInFunction[Z, P Z][f := \{ x : dom prio • (x, ran (prio x)) \}, x := pr, y := ran (prio pr)];
  rearrange;
  with normalization rewrite;
  apply lPFunWeakening to predicate \{x : dom prio • (x, ran (prio x))\} ∈ Z → P Z;
  with enabled (lRelWeakening) rewrite ;
proof[lRanInCollectPIDS]
apply inRanFunction;
  instantiate x == pr;
  prove;
  ■
proof[CollectPIDSOverrideKnownDom]
apply extensionality;
prove;
cases;
apply oplusDef;
apply dCollectPIDS;
with enabled (lRelWeakening) prove;
next;
apply oplusDef to expression collectPIDS prio ⊕ { (pr?, ( { p? } U ran (prio pr?)) ) } ;
prove;
apply dCollectPIDS to expression collectPIDS (prio ⊕ { (pr?, (prio pr? \times ( p ? ))) } );
prove;
with normalization prove;
apply applyOverride;
prove;
apply dCollectPIDS;
prove;
next;

proof[CollectPIDSOverrideUnknownDom]
apply extensionality;
prove;
cases;
apply oplusDef;
apply dCollectPIDS;
with enabled (lRelWeakening) prove;
next;
apply oplusDef ; prove;
apply dCollectPIDS ; prove;
with normalization prove;
next;

proof[RanFlattenPIDSKnownDom]
apply inPower;
apply dFlattenPIDS;
prove;
instantiate B == ran (prio pr?);
rewrite;

85
proof[FlattenPIDSOverrideKnownDom]
apply dFlattenPIDS;
 rewrite;
apply extensionality;
prove;
apply dBigCupAsBigU;
 rewrite;
apply dInBigU;
prove;
cases;
apply inRanFunction to predicate ss ∈ ran (collectPIDS prio ⊕ \{\{?\}, \{\{?\} ∪ ran (prio pr?)\}\});
with normalization prove;
cases;
 instantiate B == ran (prio pr?)
 prove;
 next;
simplify;
 instantiate B == ss
 prove;
 next;
with normalization rewrite;
cases;
instantiate ss == \{?\} ∪ ran (prio pr?)
 rewrite;
apply inRanFunction to predicate \{?\} ∪ ran (prio pr?) ∈ ran (collectPIDS prio ⊕ \{\{?\}, \{?\} ∪ ran (prio pr?)\});
 prove;
instantiate x == pr?
 prove;
 next;
simplify;
cases;
instantiate ss == \{?\} ∪ ran (prio pr?)
 rewrite;
apply inRan to predicate \{?\} ∪ ran (prio pr?) ∈ ran (collectPIDS prio ⊕ \{\{?\}, \{?\} ∪ ran (prio pr?)\});
with enabled (oplusDef) prove;
instantiate x == pr?
 rewrite;
 next;
simplify;
instantiate ss == ss_0
 apply inRanFunction to predicate ss ∈ ran (collectPIDS prio ⊕ \{\{?\}, \{\{?\} ∪ ran (prio pr?)\}\});
apply inRanFunction to predicate ss ∈ ran (collectPIDS prio);
prove;
instantiate x_0 == x;
apply applyOverride;
 rewrite;
 next;
proof[FlattenPIDS Override Unknown Dom]
apply dFlattenPIDS;
rewrite;
apply extensionality;
prove;
apply dlBigCupAsBigU;
rewrite;
apply dlInBigU;
prove;
apply inRanFunction;
prove;
cases;
with normalization rewrite;
cases;
apply applyOverride;
prove;
next;
apply applyOverride;
prove;
proves B == ss;
prove;
next;
with normalization rewrite;
cases;
instantiate ss == \{ p ? \};
apply inRanFunction;
prove;
instantiate x == pr ?;
prove;
next;
simplify;
instantiate ss == ss_0;
apply inRanFunction to predicate ss ∈ ran (collectPIDS prio ⊕ \{ pr? , \{ p ? \} \});
prove;
apply applyOverride;
rewrite;
instantiate x_0 == x;
rewrite;
next;


proof[PSAddPropUniquePrio]
invoke PSAddSig;
ninvoke PriorityScheduler;
split ¬ p? ∈ flattenPIDS prio;
cases;
  rearrange;
split ¬ p? ∈ ran (prio pr??);
rewrite;
next;
invoke PrioSched2Prio;
use JVMInvariant;
  apply partitionDef to predicate (createdVM ) ^ ( (usedVM ) ^ (freeVM )) partition PID;
  with enabled (disjointCat, distributeCapOverCupRight) rewrite;
  rearrange;
  apply extensionality to predicate createdVM ∪ usedVM = flattenPIDS prio;
  apply extensionality to predicate freeVM ∩ usedVM = {};
  apply extensionality to predicate createdVM ∩ freeVM = {};
  rewrite;
  instantiate y == p?;
  instantiate x_0 == p?;
  instantiate x_1 == p?;
  rewrite;
next;

proof[PSAddFSB]
  use IPSAddPropUniquePrio;
  use IPSAddPropFreeIsUnknown;
  rearrange;
  simplify;

  invoke PSAdd;
  invoke PSAddSig;
  invoke PSAddPrio;
  invoke PSAddErr;

  invoke PSAddPrio0;
  invoke PSAddKnownPrio;
  invoke PSAddNewPrio;
  invoke PSAdd0;
  invoke \Xi PriorityScheduler;
  invoke \Delta PriorityScheduler;
  rewrite;
  rearrange;

  split free = \emptyset;
  rewrite;
  cases;
  invoke PriorityScheduler;
  invoke PrioSched2Prio;
  invoke PrioSched1ASHS;
  invoke PrioSched0Comp;
  rearrange;
  simplify;
  rearrange;
  instantiate \textit{p} != \textit{maxpid};
  rewrite;
  next;

  invoke VMCreate;
  invoke ACreate;
  invoke AddHandler;

  invoke OutOfProcessErr;
  invoke VMOkayErr;
  invoke ACreateErr;
  invoke AShedErr;
  invoke AShedOkay;
  invoke \Xi VM;
  invoke \Xi AScheduler;
  rewrite;

  split \textit{p}? \in free;
  rewrite;
  cases;
  invoke ContainsHandler;
  invoke \Xi HandlerSet;

  invoke VMCreate0;
  invoke \Delta VM;
  invoke VM;
  rewrite;
  rearrange;

  invoke ACreate0;
  invoke \Delta AScheduler;
  invoke ASheduler;
  simplify;
B.7.1 Priority scheduler domain checks

proof[collectPIDS\$domainCheck]
prove;

proof[flattenPIDS\$domainCheck]
prove;

proof[PrioSched2Prio\$domainCheck]
prove;

proof[PSAddKnownPrio\$domainCheck]
prove;

proof[PSDispatchFSB]
invoke PSDispatchSig;
invoke PSDispatch;
invoke PSDispatch0;
simplify;
invoke \Delta PriorityScheduler;
invoke PriorityScheduler;
invoke PrioSched2Prio;
invoke PrioSched1ASHS;
invoke PrioSched0Comp;
rearrange;
simplify;
rewrite;
rearrange;
rewrite;
invoke VM;
invoke AScheduler;
invoke HandlerSet;
simplify;
rewrite;

B.8 Complete summary of proofs

This Z section collects all delayed proofs to ensure all proof sections are run through. This decoupling between what is claimed (e.g. conjectured theorems) and when it is proved (e.g. zproof \LaTeX environments) makes development faster when playing with possibly false conjectures. The Eclipse interface ensures that the user is aware of what is yet to be proved, so nothing is forgotten; or else, Eclipse complains to the user :-).
As tools process the information for each of these sections, and their corresponding dependent sections, we get a complete picture of all the modelling and proof work in the tables below.
For better maintenance, proof scripts appear in Appendix B (see Section B.8 on page 90).

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Schemas</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Theorems</td>
<td>0</td>
<td>192</td>
</tr>
<tr>
<td>Proofs</td>
<td>95</td>
<td>97</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>95</strong></td>
<td><strong>481</strong></td>
</tr>
</tbody>
</table>

Table B.1: Summary of Z declarations for Chapter B.

<table>
<thead>
<tr>
<th>Z Declarations</th>
<th>This Chapter</th>
<th>Globally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unboxed items</td>
<td>0</td>
<td>84</td>
</tr>
<tr>
<td>Axiomatic definitions</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>Generic axiomatic defs.</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Schemas</td>
<td>0</td>
<td>77</td>
</tr>
<tr>
<td>Generic schemas</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Theorems</td>
<td>0</td>
<td>192</td>
</tr>
<tr>
<td>Proofs</td>
<td>95</td>
<td>97</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>95</strong></td>
<td><strong>481</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circus Declarations</th>
<th>This Chapter</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel decls.</td>
<td>0</td>
<td>51</td>
</tr>
<tr>
<td>Channel set decls.</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Process decls.</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Process ref. assertions</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Name sets</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Actions</td>
<td>2</td>
<td>83</td>
</tr>
<tr>
<td>Action ref. assertions</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>202</strong></td>
<td><strong>638</strong></td>
</tr>
</tbody>
</table>

Table B.2: Summary of Circus declarations for Chapter B.

Complete summary does not contain categorisation of lemmas (e.g. rule, grules, frules, toolkit, etc).
Appendix C

CSPM translation of icecap Circus

This appendix shows an initial translation from the Circus model presented in Chapter 5 in FDR’s 3 CSPm. As we investigate what features to translate and what not, we had a variation of models, where the key components were kept. In what follows, we present a variety of models up to the latest. These are useful to understand the design decisions taken, and hence the needed changes in underlying models, both Circus and CSPm.

C.1 Version without SCJProcess explicitly given

This version abstracts SCJProcess as simply PID, and hence uses a mapping between PIDs and SCJSTATE. This is a valid abstraction since as far as scheduling is concerned, only these information are needed from SCJProcess. Notice we use HandlerSet to differentiate the kind of event handler. Here all checks pass. This is not good enough because the used map in PrioSched leads to state explosion.

--- Safety-Critical Java Implementation Circus Model in FDR v3, ----
--- based on Frank’s 2011 v3 version ----
------------------------------------------------------------------------

N=3
nullId = 0
GPID = {0..N}
PID = diff(GPID, {nullId})--{1..N}--GPID \ {nullId}

------------- Process ClockInterruptHandler ---------------

--- channels

channel PCIHdisableCall, PCIHdisableRet
channel PCIHenableCall, PCIHenableRet
channel PCIHstartClockHandler -- : PID
channel PCIHinitialise -- SCHID . NAT .CLOCKID?
channel PCIHhandleCall, PCIHhandleRet
channel PCIHyieldCall, PCIHyieldRet
channel PCIHregisterCall, PCIHregisterRet
channel PCIHcatchError
channel PPSIgetNextProcessCall
channel PPSIgetNextProcessRet : PID
channel VMthrow, VMcihRegister
channel PVMtransferTo : PID . PID

VMChan = {| VMthrow, VMcihRegister |}

------------- State Components -------------
--- disable count ---
channel disable_count_set, disable_count_get : {0..N}

disable_count_ops = {| disable_count_set, disable_count_get |}

disable_count_field(v) =
  disable_count_set ? v -> disable_count_field(v) []
  disable_count_get ! v -> disable_count_get(v)

--- HVM clock ready ---
channel hvm_clock_ready_set, hvm_clock_ready_get : Bool

hvm_clock_ready_ops = {| hvm_clock_ready_set, hvm_clock_ready_get |}

hvm_clock_ready_field(v) =
  hvm_clock_ready_set ? v -> hvm_clock_ready_field(v) []
  hvm_clock_ready_get ! v -> hvm_clock_ready_field(v)

--- current process ---
channel current_cih_pid_set, current_cih_pid_get : GPID

current_cih_pid_ops = {| current_cih_pid_set, current_cih_pid_get |}

current_cih_pid_field(v) =
  current_cih_pid_set ? v -> current_cih_pid_field(v) []
  current_cih_pid_get ! v -> current_cih_pid_field(v)

--- handler process ---
channel handler_cih_pid_set, handler_cih_pid_get : GPID

handler_cih_pid_ops = {| handler_cih_pid_set, handler_cih_pid_get |}

handler_cih_pid_field(v) =
  handler_cih_pid_set ? v -> handler_cih_pid_field(v) []
  handler_cih_pid_get ! v -> handler_cih_pid_field(v)

--- Process ---

ClockInterruptHandler = let

--- Local State ---

State = hvm_clock_ready_field(false) |||
  disable_count_field(0) |||
  current_cih_pid_field(nullId) |||
  handler_cih_pid_field(nullId)

StateChan = union(hvm_clock_ready_ops, disable_count_ops)

InitCIHSt = hvm_clock_ready_set ! true -> SKIP ;
  disable_count_set ! 1 -> SKIP ;
  current_cih_pid_set ! nullId -> SKIP ;
  handler_cih_pid_set ! nullId -> SKIP

CIHEnable(c) = disable_count_set ! (c - 1) -> SKIP ;
disable_count_get ? n ->
  (if n == 0 then hvm_clock_ready_set ! true -> SKIP else SKIP)

CIHDisable = disable_count_get ? c ->
  (c < N & disable_count_set ! (c+1) -> hvm_clock_ready_set ! false -> SKIP)

--- Local Actions ---

InitCIH = InitCIHSt ; PCIHinitialise -> SKIP

-- not used CIH start

StartClockHandler = PCIHstartClockHandler -> SKIP

Yield = PCIHyieldCall -> Handle ; PCIHyieldRet -> SKIP

Handle = PCIHhandleCall -> Disable ; PCIHhandleRet -> SKIP

Register = PCIHregisterCall -> VMcihRegister -> PCIHregisterRet -> SKIP

-- use guards to block the enable call if precondition fails; see RTSS paper
Enable = disable_count_get?c ->
  (c > 0 & PCIHenableCall -> CIHEnable(c) ; PCIHenableRet -> SKIP)

Disable = PCIHdisableCall -> CIHDisable ; PCIHdisableRet -> SKIP

--- Local actions plumbing

Loop = (let X =
  PPSIgetNextProcessCall -> PPSIgetNextProcessRet?p ->
  current_cih_pid_set!p -> Enable ; handler_cih_pid_get?h ->
  (h != nullId & current_cih_pid_get?c ->
  (c != nullId & PVMtransferTo.h.c -> X)) within X)

Catch = VMthrow -> PCIHcatchError -> SKIP

CIHRun = Loop \ Catch

CIHApi = StartClockHandler [] Disable [] Enable [] Handle [] Yield [] Register

Execute = CIHRun ||| CIHApi

--- Main Action ---

Main = (InitCIH ; Execute) \ VMChan

MainWithState = (Main [] StateChan [] State) \ StateChan

within MainWithState

----------------------------- --- Process PrioSchedImpl ---
-----------------------------

--- channels

channel ENVcreatePSBridge

channel KPSmoveCall

channel KPSmoveRet : PID
channel KPSGetCurrentProcess, KPSstopCall: PID
channel KPSstopRet

--- State Components ---

--- current process ---

channel current_psi_pid_set, current_psi_pid_get : GPID
current_psi_pid_ops = {| current_psi_pid_set, current_psi_pid_get |}
current_psi_pid_field(v) =
  current_psi_pid_set ? v -> current_psi_pid_field(v) []
  current_psi_pid_get ! v -> current_psi_pid_field(v)

--- Process ---

PrioSchedImpl = let

--- Local State ---

State = current_psi_pid_field(nullId)
StateChan = current_psi_pid_ops
InitSt = current_psi_pid_set!nullId -> SKIP

--- Local Actions ---

InitPSISt = ENVcreatePSBridge -> InitSt
GetNextProcess = PCIHdisableCall -> PCIHdisableRet -> KPSmoveCall ->
  KPSmoveRet?p -> current_psi_pid_set!p ->
  (current_psi_pid_get ? current ->
    (if (current == nullId) then
     KPSgetCurrentProcess?current -> KPSstopCall!current ->
     KPSstopRet -> SKIP
    else
     SKIP)) ; PCIHenableCall -> PCIHenableRet -> SKIP
PSIRun = (let X =
  PPSIgetNextProcessCall -> GetNextProcess ;
  (current_psi_pid_get ? c -> (c != nullId & PPSIgetNextProcessRet!c -> X))
within X)

--- Main Action ---

Main = InitPSISt ; PSIRun
MainWithState = (Main [| StateChan |] State) \ StateChan
within MainWithState

----------------------- Process PrioSched -----------------------

--- channels

datatype HW_PRIO = MIN_HW_PRIO | MAX_HW_PRIO
datatype PRIO = MIN_PRIO | MAX_PRIO

channel PPSgetMinHWPrrio, PPSgetMaxHWPrrio : HW_PRIO
channel PPSgetMinPrio, PPSgetMaxPrio: PRIO

channel KPSreleaseCall: PID
channel KPSreleaseRet

channel FIXME: PID

--- State Components ---

--- main process ---

channel main_process_set, main_process_get : GPID

main_process_ops = {| main_process_set, main_process_get |}

main_process_field(v) = main_process_set ? v -> main_process_field(v) []
main_process_get ! v -> main_process_field(v)

--- current process ---

channel current_ps_pid_set, current_ps_pid_get : GPID

current_ps_pid_ops = {| current_ps_pid_set, current_ps_pid_get |}

current_ps_pid_field(v) = 
current_ps_pid_set ? v -> current_ps_pid_field(v) []
current_ps_pid_get ! v -> current_ps_pid_field(v)

--- Process ---

PrioSched = let

--- Local State ---

State = main_process_field(nullId) ||| current_ps_pid_field(nullId)

StateChan = main_process_ops

InitSt = main_process_set!nullId -> SKIP ; current_ps_pid_set!nullId -> SKIP

--- Local Actions ---

InitPSSt = InitSt

------- SCJ API provision -------

GetHWPrrio = PPSgetMinHWPrrio!MIN_HW_PRIO -> SKIP []
PPSgetMaxHWPrrio!MAX_HW_PRIO -> SKIP

GetPrrio = PPSgetMinPrio!MIN_PRIO -> SKIP [] PPSgetMaxPrio!MAX_PRIO -> SKIP

SCJapi = GetHWPrrio [] GetPrrio

------- CIH API use -------

SCJStop = KPSstopCall?c ->
(c != nullId) & main_process_get?main ->
(main != nullId) & PVMtransferTo.c.main -> KPSstopRet -> SKIP

Move = KPSmoveCall -> FIXME?p -> KPSmoveRet!p -> SKIP

CIHApi = Move [] SCJStop

------- SCJ RTE use -------

Start = SKIP

AddOuterMostSeq = SKIP

Release_PRE(apeh) =
    if apeh != nullId then
        False
    else
        False

ReleaseHandler(apeh) =
    Release_PRE(apeh) & FIXME!apeh -> SKIP

Release = KPSreleaseCall?apeh -> PCIHdisableCall -> PCIHdisableRet ->
    ReleaseHandler(apeh) ; PCIHenableCall ->
    PCIHenableRet -> KPSreleaseRet -> SKIP

GetCurrentProc = SKIP

InsertReadyQueue = SKIP

SCJRTE = Start [] AddOuterMostSeq [] Release [] GetCurrentProc [] InsertReadyQueue

--- Local actions plumbing

PSRun = SCJApi ||| SCJRTE ||| CIHApi

PSCatch = PCIHcatchError -> SKIP

PSEexecute = PSRun \ PSCatch

--- Main Action ---

Main = InitPSSt ; PSEexecute

MainWithState = (Main [] StateChan [] State) \ StateChan

within MainWithState

---------------------------------------------------- Assertion checks ----------------------------------------------------

assert ClockInterruptHandler :[deadlock free [FD]]
assert ClockInterruptHandler:[divergence free [FD]]
assert PrioSchedImpl :[deadlock free [FD]]
assert PrioSchedImpl :[divergence free [FD]]
assert PrioSched :[deadlock free [FD]]
assert PrioSched :[divergence free [FD]]
C.2 Version with SCJProces explicitly given

This is the latest (still on going) version of the translation that takes into account SCJProces as a separate CSPm process, hence avoiding the use of maps. It is still not quite right because we do not take into account the needed set of PIDs that SCJProces is meant to manage. Notice we also introduce guards for the Circus calculated preconditions.

nullId = 0
N=3
GPID = {0..N}
PID = diff(GPID, {nullId})--{1..N}--GPID \ {nullId}

--- channels ---

channel PCIHdisableCall, PCIHdisableRet
channel PCIHenableCall, PCIHenableRet
channel PCIHstartClockHandlerCall : PID
channel PCIHstartClockHandlerRet
channel PCIH initialise -- SCHID . NAT . CLOCKID?
channel PCIH handleCall, PCIH handleRet
channel PCIH yieldCall, PCIH yieldRet
channel PCIH registerCall, PCIH registerRet
channel PCIH catchErrorCall, PCIH catchErrorRet
channel PPSI getNextProcessCall
channel PPSI getNextProcessRet : PID
channel VM throw, VM cihRegister
channel VM createProcess: PID
channel VM transfer : PID . PID

VMChan = {| VM throw, VM cihRegister, VM createProcess |}

--- HVM clock ready ---

channel hvm_clock_ready_set, hvm_clock_ready_get : Bool

hvm_clock_ready_ops = {| hvm_clock_ready_set, hvm_clock_ready_get |}

hvm_clock_ready_field(v) =
    hvm_clock_ready_set ? x -> hvm_clock_ready_field(x) []
    hvm_clock_ready_get ! v -> hvm_clock_ready_field(v)

--- disable count ---

channel disable_count_set, disable_count_get : {0..N}

disable_count_ops = {| disable_count_set, disable_count_get |}

disable_count_field(v) =
    disable_count_set ? x -> disable_count_field(x) []
disable_count_get ! v -> disable_count_field(v)

--- current process ---

channel current_cih_pid_set, current_cih_pid_get : GPID

current_cih_pid_ops = {| current_cih_pid_set, current_cih_pid_get |

current_cih_pid_field(v) =
    current_cih_pid_set ? x -> current_cih_pid_field(x) []
    current_cih_pid_get ! v -> current_cih_pid_field(v)

--- handler process ---

channel handler_cih_pid_set, handler_cih_pid_get : GPID

handler_cih_pid_ops = {| handler_cih_pid_set, handler_cih_pid_get |

handler_cih_pid_field(v) =
    handler_cih_pid_set ? x -> handler_cih_pid_field(x) []
    handler_cih_pid_get ! v -> handler_cih_pid_field(v)

--- Process ---

ClockInterruptHandler = let

--- Local State ---

State = hvm_clock_ready_field(false) |||
    disable_count_field(0) |||
    current_cih_pid_field(nullId) |||
    handler_cih_pid_field(nullId)

StateChan = union(union(union(hvm_clock_ready_ops, disable_count_ops),
    current_cih_pid_ops), handler_cih_pid_ops)

InitCIHSt = hvm_clock_ready_set ! true -> SKIP ;
    disable_count_set ! 1 -> SKIP ;
    current_cih_pid_set ! nullId -> SKIP ;
    handler_cih_pid_set ! nullId -> SKIP

CIHEnable(c) = disable_count_set ! (c - 1) -> SKIP ;
    disable_count_get ? n ->
        (if n == 0 then
            hvm_clock_ready_set ! true -> SKIP
        else
            SKIP)

CIHDisable = disable_count_get ? c ->
    (c < N & disable_count_set ! (c+1) ->
        hvm_clock_ready_set ! false -> SKIP)

--- Local Actions ---

InitCIH = InitCIHSt ; PCIHinitialize -> SKIP

StartClockHandler = PCIHstartClockHandlerCall?h -> handler_cih_pid_set ! h ->
    PCIHstartClockHandlerRet -> SKIP
Yield = PCIHyieldCall -> Handle ; PCIHyieldRet -> SKIP

Handle = PCIHhandleCall -> Disable ; PCIHhandleRet -> SKIP

Register = PCIHregisterCall -> VMcihRegister -> PCIHregisterRet -> SKIP

-- use guards to block the enable call if precondition fails; see RTSS paper
Enable = disable_count_get?c -> (c > 0 & PCIHenableCall -> CIHEnable(c) ;
PCIHenableRet -> SKIP)

Disable = PCIHdisableCall -> CIHdisable ; PCIHdisableRet -> SKIP

--- Local actions plumbing

Loop = (let X =
PPSIgetNextProcessCall -> PPSIgetNextProcessRet?p ->
current_cih_pid_set!p -> Enable ;
handler_cih_pid_get?h ->
  (h != nullId & current_cih_pid_get?c ->
    (c != nullId & VMtransfer.h.c -> X))
within X)

CIHErr = PCIHcatchErrorCall -> PCIHcatchErrorRet -> SKIP

-- here we should call usercode between Call/Ret?

CIHCatch = VMthrow -> PCIHcatchErrorCall -> PCIHcatchErrorRet -> SKIP

CIHRun = Loop \ CIHCatch

CIHApi = StartClockHandler [] Disable [] Enable [] Handle [] Yield [] Register

Execute = CIHRun ||| CIHApi

--- Main Action ---

Main = (InitCIH ; Execute) \ VMChan

MainWithState = (Main [] StateChan [] State) \ StateChan

within MainWithState

-------------------------------
--- Process PrioSchedImpl ---
-------------------------------

--- channels

channel ENVcreatePSBridge

channel KPSmoveCall
channel KPSmoveRet : PID
channel KPSgetCurrentProcess, KPSstopCall: PID
channel KPSstopRet

--- State Components ---

--- current process ---
channel current_psi_pid_set, current_psi_pid_get : GPID

current_psi_pid_ops = {| current_psi_pid_set, current_psi_pid_get |

current_psi_pid_field(v) =
    current_psi_pid_set ? x -> current_psi_pid_field(x) []
    current_psi_pid_get ! v -> current_psi_pid_field(v)

--- Process ---
PrioSchedImpl = let

--- Local State ---
State = current_psi_pid_field(nullId)
StateChan = current_psi_pid_ops
InitSt = current_psi_pid_set!nullId -> SKIP

--- Local Actions ---
InitPSISt =
    ENVcreatePSBridge -> InitSt

GetNextProcess =
    PCIHdisableCall -> PCIHdisableRet ->
    KPSmoveCall -> KPSmoveRet?p -> current_psi_pid_set!p ->
    (current_psi_pid_get ? current ->
    (if (current == nullId) then
        KPSgetCurrentProcess?current ->
        KPSstopCall!current -> KPSstopRet -> SKIP
    else
        SKIP)
    )
    PCIHenableCall -> PCIHenableRet -> SKIP

PSIRun = (let X =
    PPSIgetNextProcessCall -> GetNextProcess ;
    (current_psi_pid_get ? c ->
    (c != nullId & PPSIgetNextProcessRet!c -> X))
    within X)

--- Main Action ---
Main = InitPSISt ; PSIRun
MainWithState = (Main []| StateChan [] State) \ StateChan
within MainWithState

-----------------------------
--- ScjProcess ---
-----------------------------

--- channels ---

channel FIXME: PID
channel VMexecuteWithStack : PID

channel KPSinsertRQueueRet, KPSinsertSQueueRet
channel KPSinsertRQueueCall, KPSinsertSQueueCall : PID

channel PMEHhandleAsyncEvent: PID . SCJTARGET

--- State Components ---

--- VM process started? ---

channel vm_process_started_get, vm_process_started_set: Bool

vm_process_started_ops = {| vm_process_started_get, vm_process_started_set |

vm_process_started_field(v) =
    vm_process_started_set ? x -> vm_process_started_field(x) []
    vm_process_started_get ! v -> vm_process_started_field(v)

--- SCJ State ---

datatype SCJSTATE = NEW | READY | EXECUTING | BLOCKED | SLEEPING | HANDLED | TERMINATED

datatype SCJTARGET = PEH | AEH | OEH | MSEQ | OTHER

channel KSCJPcreate: PID . SCJTARGET

channel scj_process_state_set, scj_process_state_get : SCJSTATE

scj_process_state_ops = {| scj_process_state_set, scj_process_state_get |

scj_process_state_field(s) =
    scj_process_state_set ? x -> scj_process_state_field(x) []
    scj_process_state_get ! s -> scj_process_state_field(s)

--- SCJ target pid ---

channel scj_pid_target_set, scj_pid_target_get : GPID . SCJTARGET

scj_pid_target_ops = {| scj_pid_target_set, scj_pid_target_get |

scj_pid_target_field(v, w) =
    scj_pid_target_set ? x ? y -> scj_pid_target_field(x, y) []
    scj_pid_target_get ! v ! w -> scj_pid_target_field(v, w)

--- Process -------

SCJProcess = let

--- Local State ---

State = scj_process_state_field(NEW) |||
    scj_pid_target_field(nullId, OTHER) |||
    vm_process_started_field(false)

StateChan = union(union(scj_process_state_ops, scj_pid_target_ops),
    vm_process_started_ops)
InitSt = scj_process_state_set!NEW -> SKIP ;
    scj_pid_target_set!nullId!OTHER -> SKIP ;
    vm_process_started_set!false -> SKIP

--- Local Actions ---

InitSCJP = InitSt ; SCJInitialise

--- Local actions plumbing

--SCJ: we are eliding away what the vm.Process.ProcessExecutor (162-79) does!
-- Here just masking it with VMexecuteWithStack, which is like VMtransferTo
SCJInitialise = KSCJPcreate ? pid ? target -> (SCJStart1(pid) ;
    scj_pid_target_set ! pid ! target -> SKIP)

MakeReady(t) = scj_process_state_set!READY -> KPSinsertRQueueCall!t -> SKIP

PEHNextState(s, p) = if (s == HANDLED) then
    (scj_process_state_set!TERMINATED -> SKIP
    |~|
    scj_process_state_set!SLEEPING -> SCJStart2(p) ;
KPSinsertSQueueCall!p -> SKIP)

-- else if (s == WAITING) then
--   SKIP
-- [avoids a SKIP on external choice? not featuring in L1?
--   scj_process_state_set!WAITING -> SKIP
else -- if (s != WAITING) then
    MakeReady(p)

AEHNextState(s, p) = if (s == HANDLED) then
    (scj_process_state_set!TERMINATED -> SKIP
    |~|
    scj_process_state_set!BLOCKED -> SKIP)
else
    MakeReady(p)

OEHNextState(s, p) = if (s == HANDLED) then
    scj_process_state_set!TERMINATED -> SKIP
else
    MakeReady(p)

MSEQNextState(s, p) = OEHNextState(s, p)

NextState = scj_process_state_get?s -> scj_pid_target_get?p?t ->
    if (t == PEH) then
        PEHNextState(s, p)
    else if (t == AEH) then
        AEHNextState(s, p)
    else if (t == OEH) then
        OEHNextState(s, p)
    else if (t == MSEQ) then
        MSEQNextState(s, p)
    else
        SKIP

SCJStart1(p) = VMexecuteWithStack ! p -> vm_process_started_set ! true ->
    VMtransfer ! p ! p -> SKIP

    -- complicated indirect call for native method executeWithStack
-- eWS -> vm.Process.ProcessExecutor.Run with started set to false and
-- transfer to itself.
-- how are going to model that from the environment offering VMtransfer?
-- TODO
-- here we are not modelling vm.Process, perhaps we should? TODO

SCJStart2(p) = scj_pid_target_get?q?t ->
    (p==q and q != nullId) & VMexecuteWithStack ! q ->
    vm_process_started_get ? b ->
        (if (b) then
            SCJExecute_(q)
        else
            SKIP
    )

SCJCatch = VMthrow -> SKIP
-- SCJProcess has a execution reporting mechanism we are not modelling here.
-- See ScjProcess constructor inner method ProcessLogic.catchError()

SCJRun(p) = scj_pid_target_get?q?t -> ((p==q and q != nullId) &
    PMEHandleAsyncEvent ! q ! t -> SKIP)

-- we use the parameterised version because of the two calls, one without a
-- p, another with an expected p. TODO: better way?
--
-- following ProcessLogic within ScjProcess constructor you get to inner
-- class method ManagedMemory.SinglecoreBehavior.enter(ManagedEventHandler)
-- I am eliding away all the details about memory area management and
-- getting straight to HandlerSet execution upon running
-- to model memory, we would need more complicated state and FDR might not
-- like that.
--
-- The call to "logic.run()" inside
-- ManagedMemory.SinglecoreBehavior.enter(mevh)
-- will trigger javax.realtime.AsyncEventHandler.run(); this
-- class is the super class of ManagedEventHandler, hence all the
-- Scj Handler types (AEH, PEH, OEH, MS, etc.).
--
-- ManagedEventHandler.handleAsyncEvent() is not being modelled directly,
-- given our handler sets only consider PiDs. Thus, the "p" given as
-- a parameter is this such MEH that will come from the scheduler’s run
-- method that calls next stat?

SCJExecute =
    scj_process_state_get?s -> (s == READY & ((scj_pid_target_get?q?t ->
        SCJRun(q)) \ SCJCatch))

SCJExecute_(p) = scj_process_state_get?s ->
    (s == READY & (SCJRun(p) \ SCJCatch))

--- Main Action ---
Main = InitSCJP ; SCJExecute

MainWithState = (Main [] StateChan [] State) \ StateChan

within MainWithState

-----------------------------
--- Process PrioSched ---
--- channels

datatype HW_PRIO = MIN_HW_PRIO | MAX_HW_PRIO
datatype PRIO = MIN_PRIO | MAX_PRIO

channel PPSgetMinHWPrrio, PPSgetMaxHWPrrio : HW_PRIO
channel PPSgetMinPrrio, PPSgetMaxPrrio: PRIO

channel KPSreleaseCall: PID
channel KPSreleaseRet

channel KPSstartCall, KPSstartRet

--- State Components ---

channel scj_process_map_set, scj_process_map_get : Map(PID, SCJSTATE)
--(| PID => SCJSTATE |)
scj_process_map_field(m) = SKIP

--- main process ---

channel main_process_set, main_process_get : GPID
main_process_ops = {| main_process_set, main_process_get |}
main_process_field(v) =
  main_process_set ? x -> main_process_field(x) []
  main_process_get ! v -> main_process_field(v)

--- current process ---

channel current_ps_pid_set, current_ps_pid_get : GPID
current_ps_pid_ops = {| current_ps_pid_set, current_ps_pid_get |}
current_ps_pid_field(v) =
  current_ps_pid_set ? x -> current_ps_pid_field(x) []
  current_ps_pid_get ! v -> current_ps_pid_field(v)

--- outermost missionSeq process ---

channel mseq_process_set, mseq_process_get : GPID
mseq_process_ops = {| mseq_process_set, mseq_process_get |}
mseq_process_field(v) =
  mseq_process_set ? x -> mseq_process_field(x) []
  mseq_process_get ! v -> mseq_process_field(v)

--- Priority Frame ---

channel ready_queue_set, sleep_queue_set, ready_queue_get, sleep_queue_get: Seq(PID)
ready_queue_ops = {| ready_queue_set, ready_queue_get |}
sleep_queue_ops = {| sleep_queue_set, sleep_queue_get |}
ready_queue_field(q) =
    sleep_queue_get ? s -> ready_queue_set ? x ->
    (inter(set(s), set(x)) == {}) &
ready_queue_field(x) []
    ready_queue_get ! q -> ready_queue_field(q)

sleep_queue_field(q) =
    -- for the invariant that ran\sq \cap ran\rd = {} 
    ready_queue_get ? r -> sleep_queue_set ? x ->
    (inter(set(r), set(x)) == {}) & sleep_queue_field(x)
    []
    sleep_queue_get ! q -> sleep_queue_field(q)

--- Process ---

PrioSched = let

--- Local State ---

State = main_process_field(nullId) |||
    current_ps_pid_field(nullId) |||
    mseq_process_field(nullId) |||
    scj_process_map_field((| |)) ||| -- emptyMap(PID)) |||
    ready_queue_field(<>) |||
    sleep_queue_field(<>)

StateChan =
    union(union(union(union(main_process_ops, current_ps_pid_ops),
        mseq_process_ops), ready_queue_ops), sleep_queue_ops)

InitSt = main_process_set!nullId -> SKIP ;
    current_ps_pid_set!nullId -> SKIP ;
    mseq_process_set!nullId -> SKIP ;
    scj_process_map_set!(| |) -> SKIP ;
    ready_queue_set!<> -> SKIP ;
    sleep_queue_set!<> -> SKIP

--- Local Actions ---

InitPSSt = InitSt ; FIXME ? createIdleProcess -> SKIP

------- SCJ API provision -------

GetHWPrio = PPSgetMinHWprio!MIN_HW_PRIO -> SKIP
    []
    PPSgetMaxHWprio!MAX_HW_PRIO -> SKIP

GetPrio = PPSgetMinPrio!MIN_PRIO -> SKIP
    []
    PPSgetMaxPrio!MAX_PRIO -> SKIP

SCJApi = GetHWPrio | GetPrio

------- CIH API use ----

SCJStop = KPSstopCall?c -> (c != nullId) &
    main_process_get?main -> (main != nullId) &
    VMtransfer.c.main -> KPSstopRet -> SKIP

Move = KPSmoveCall -> FIXME?p -> KPSmoveRet!p -> SKIP
CIHApi = Move [] SCJStop

------- SCJ RTE use -------

Start = KPSstartCall -> FIXME?p -> current_ps_pid_set ! p ->
VMcreateProcess?m -> main_process_set ! m ->
PCIHregisterCall -> PCIHregisterRet -> PCIHenableCall ->
PCIHenableRet -> PCIHstartClockHandlerCall ! m ->
PCIHstartClockHandlerRet -> PCIHyieldCall -> PCIHyieldRet ->
KPSstartRet -> SKIP

AddOuterMostSeq = SKIP

-- dom(m, k) = mapMember(m, k)
-- Release_PRE(m, apeh) = ((apeh != nullId) and dom(m, apeh))
Release_PRE(state, apeh) = ((apeh != nullId) and state == READY)

ReleaseHandler(apeh) = scj_process_state_get?state -> Release_PRE(state, apeh) &
(if (state == BLOCKED) then
  scj_process_state_set!READY ->
  ready_queue_get ? q -> ready_queue_set ! (q ^ <apeh>) -> SKIP
else
  -- EXECUTING is already running
  -- anything else, is already ready
  SKIP)
-- scj_process_map_get?m -> Release_PRE(m, apeh) &
--(if (mapLookup(m, apeh) == BLOCKED) then
-- scj_process_map_set!mapUpdate(m, apeh, READY) ->
-- ready_queue_get ? q -> ready_queue_set ! (q ^ <apeh>) -> SKIP
--else
-- -- EXECUTING is already running
-- -- anything else, is already ready
-- SKIP)

Release = KPSreleaseCall?apeh -> PCIHdisableCall -> PCIHdisableRet ->
ReleaseHandler(apeh) ;
PCIHenableCall -> PCIHenableRet -> KPSreleaseRet -> SKIP

GetCurrentProc = SKIP
InsertReadyQueue = SKIP

SCJRTE = Start [] AddOuterMostSeq [] Release []
GetCurrentProc [] InsertReadyQueue

--- Local actions plumbing

PSRun = SCJApi ||| SCJRTE ||| CIHApi

PSCatch = PCIHcatchErrorCall -> PCIHcatchErrorRet -> SKIP
  -- call user code here?

PSEexecute = PSRun \ PSCatch

--- Main Action ---

Main = InitPSSt ; PSEexecute

MainWithState = (Main [] StateChan [] State) \ StateChan
within MainWithState
-----------------------------
--- Assertion checks ---
-----------------------------

assert ClockInterruptHandler : [deadlock free [FD]]
assert ClockInterruptHandler : [divergence free [FD]]
assert PrioSchedImpl : [deadlock free [FD]]
assert PrioSchedImpl : [divergence free [FD]]
assert SCJProcess : [deadlock free [FD]]
assert SCJProcess : [livelock free [FD]]
--assert PrioSched : [deadlock free [FD]]
--assert PrioSched : [divergence free [FD]]
Bibliography


