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

by

Leo Freitas[1], Ana Cavalcanti[2], Andy Wellings[2]

[1] Newcastle University
School of Computing Science
Centre for Software Reliability (CSR)

[2] University of York
Department of Computer Science
High Integrity Systems Engineering (HISE) Group

Contents

List of Figures

List of Tables

Declarations summary

Abbreviations

Given sets

Free types

Generic definitions

Axiomatic definitions

Schemas

Z Sections

Z Proof Sections

Theorems summary

Z Toolkit extension

Lemmas

Theorems

Axiomatic consistency lemmas

Z/Eves housekeeping

Assumption Rules

Rewriting Rules

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 upto a *maxpid* value, where the *nullpid* is represented by either zero or anything beyond the maximum. [***Circus tools off***]

theorem basic_defs_axiom1_vc_fsb_axiom
$\exists\, maxpid : \mathbb{N}_1 \mid true \bullet true$

[***Circus tools on***]

| $maxpid : \mathbb{N}_1$

[***Circus tools off***]

theorem rule lMaxPIDPositive
$1 \leq maxpid$

theorem PID_vc_fsb_horiz_def
$\exists\, PID : \mathbb{P}(1 \mathrel{..} maxpid) \mid true \bullet true$

[***Circus tools on***]

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

$PID == 1 \mathrel{..} maxpid$

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***]

theorem rule lPIDNotEmpty
$\neg\, PID = \{\}$

theorem rule lMinPIDValue
$1 \in PID$

theorem grule gMaxpidMaxType
$maxpid \in \mathbb{Z}$

theorem rule lMaxpidIsPID
$maxpid \in PID$

theorem basic_defs_axiom2_vc_fsb_axiom
$\exists\, nullpid : \mathbb{N} \mid true \bullet \forall\, p : PID \bullet p < nullpid$

[***Circus tools on***]

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

$$\begin{array}{|l} nullpid : \mathbb{N} \\ \hline \forall\, p : PID \bullet p < nullpid \end{array}$$

[***Circus tools off***]

theorem rule lNullpidBound
$maxpid < nullpid$

theorem rule lNullPIDDisjoint
$\forall\, p : PID \bullet \neg\ p = nullpid$

theorem GPID_vc_fsb_horiz_def
$\exists\, GPID : \mathbb{P}(PID \cup \{nullpid\}) \mid true \bullet true$

[***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.

$$GPID \;==\; PID \cup \{\, nullpid \,\}$$

[***Circus tools off***]

theorem grule gPIDMaxType
$PID \in \mathbb{P}\ \mathbb{Z}$

theorem grule gGPIDMaxType
$GPID \in \mathbb{P}\ \mathbb{Z}$

theorem rule lNullIsGPID
$nullpid \in GPID$

theorem rule lNullIsNotPID
$\neg\ nullpid \in PID$

theorem rule lPIDIsGPID
$\forall x : PID \bullet x \in GPID$

theorem rule lMinPIDIsGPID
$1 \in GPID$

theorem rule lMaxPIDIsGPID
$maxpid \in GPID$

theorem rule lNonNullGPIDIsPID
$\forall p : GPID \mid \neg\ p = nullpid \bullet p \in PID$

[***Circus tools on***]

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	2	2
Axiomatic definitions	2	2
Generic axiomatic defs.	0	0
Schemas	0	0
Generic schemas	0	0
Theorems	19	19
Proofs	0	0
Total	**23**	**23**

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, addresing 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

$base, size, free : \mathbb{N}$

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

$$VMError ::= VMOkay \mid VMOutOfMemory \mid VMOutOfProcess \mid VMUnknownProcess$$

FIX ME: list of exceptions that are modelled but not present in the code please! Ex. *VMOutOfProcess* is a key one reflecting the available/free resources that isn't caught anywhere in the code (that we could see / find).
Look at javax.scj.util.SCJErrorReporter as well as javax.safetycritical.ScjProcess.ExceptionReporter and vm.ProcessLogic.catchError(Throwable). These places handle exceptions generally, except for Out of memory.
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

$createdVM, usedVM, freeVM : \mathbb{P}\ PID$

$\langle createdVM, usedVM, freeVM \rangle \text{ partition } PID$

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.

$$
\begin{array}{|l}
\textit{VMCreate0} \\
\hline
\Delta\, VM \\
p! : PID \\
\hline
p! \in \mathit{freeVM} \\
\mathit{createdVM}' = \mathit{createdVM} \cup \{\, p! \,\} \\
\mathit{freeVM}' = \mathit{freeVM} \setminus \{\, p! \,\} \\
\mathit{usedVM}' = \mathit{usedVM} \\
\hline
\end{array}
$$

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).

$$
\begin{array}{|l}
\textit{VMRun0} \\
\hline
\Delta\, VM \\
p? : PID \\
\hline
p? \in \mathit{createdVM} \\
\mathit{createdVM}' = \mathit{createdVM} \setminus \{\, p? \,\} \\
\mathit{freeVM}' = \mathit{freeVM} \\
\mathit{usedVM}' = \mathit{usedVM} \cup \{\, p? \,\} \\
\hline
\end{array}
$$

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.

$$
\begin{array}{lcl}
\mathit{VMOkayErr} & == & [\, vmerr! : VMError \mid vmerr! = VMOkay \,] \\
\mathit{OutOfProcessErr} & == & [\, \Xi VM;\ vmerr! : VMError \mid \mathit{freeVM} = \varnothing \\
 & & \quad \wedge\ vmerr! = VMOutOfProcess \,] \\
\mathit{UnknownProcessErr} & == & [\, \Xi VM;\ p? : PID;\ vmerr! : VMError \mid p? \notin \mathit{createdVM} \\
 & & \quad \wedge\ vmerr! = VMUnknownProcess \,] \\
\mathit{VMCreate} & == & ((\mathit{VMCreate0} \wedge \mathit{VMOkayErr}) \vee \mathit{OutOfProcessErr}) \\
\mathit{VMRun} & == & ((\mathit{VMRun0} \wedge \mathit{VMOkayErr}) \vee \mathit{UnknownProcessErr})
\end{array}
$$

2.2.2 VM process operations preconditions

To ensure they are total, we declare signatures with no extra precondition for both operations[1].

$$
\begin{array}{lcl}
\mathit{VMSig} & == & [\, VM \mid true \,] \\
\mathit{VMRunSig} & == & [\, \mathit{VMSig};\ p? : PID \,]
\end{array}
$$

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

theorem VMCreateFSB
$\quad \forall\, \mathit{VMSig} \bullet \mathbf{pre}\ \mathit{VMCreate}$

theorem VMRunFSB
$\quad \forall\, \mathit{VMRunSig} \bullet \mathbf{pre}\ \mathit{VMRun}$

[***Circus tools on***]

FIX ME: Unfortunatley, 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.

[1] This 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).

Z Declarations	**This Chapter**	**Globally**
Unboxed items	10	12
Axiomatic definitions	0	2
Generic axiomatic defs.	0	0
Schemas	4	4
Generic schemas	0	0
Theorems	2	21
Proofs	0	0
Total	**16**	**39**

Table 2.1: Summary of Z declarations for Chapter 2.

Chapter 3

Abstract scheduler

section *ascheduler* **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. $current \in GPID$). The invariant ensures that all valid (non-null) process identifiers are uniquely used across the various schedulable sets used. Moreover, that these *PID*s are all *PID*s known: these schedulable sets form a partition of *PID*s, hence all *PID*s must be uniquely determined as either *readyAS*, *blockedAS*, *freeAS*, or (non-null) *current*.

AScheduler

$current : GPID$
$readyAS, blockedAS, freeAS : \mathbb{P}\ PID$

$\langle \{ current \} \setminus \{ nullpid \}, readyAS, blockedAS, freeAS \rangle \text{ partition } PID$

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

ASchedulerInit

$AScheduler'$

$current' = nullpid$
$readyAS' = \varnothing$
$blockedAS' = \varnothing$
$freeAS' = PID$

3.1.1 Abstract scheduler operations

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

Operation	**Description**	**Precondition**
Create	moves a *freeAS* *PID* into *readyAS*	$freeAS \neq \varnothing$
Dispatch	takes any *readyAS* as *current*	$current = nullpid \land readyAS \neq \varnothing$
Block	puts *current* in *blockedAS* and makes *current* null	$current \neq nullpid$
Timeout	puts *current* in *readyAS* and makes *current* null	$current \neq nullpid$
Wakeup	moves a *blockedAS* *PID* into *ready*	$blockedAS \neq \varnothing$
Destroy	moves either $current, ready, blocked$ *PID* to *free*	$current \neq nullpid \lor readyAS, blockedAS \neq \varnothing$

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.

$$
\begin{array}{l}
\textbf{ACreate0} \\
\hline
\Delta AScheduler \\
p! : PID \\
\hline
freeAS \neq \varnothing \\
current' = current \\
readyAS' = readyAS \cup \{ p! \} \\
blockedAS' = blockedAS \\
freeAS' = freeAS \setminus \{ p! \} \\
p! \in freeAS \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{ADispatch0} \\
\hline
\Delta AScheduler \\
p! : PID \\
\hline
current = nullpid \\
readyAS \neq \varnothing \\
current' \in readyAS \\
readyAS' = readyAS \setminus \{ current' \} \\
blockedAS' = blockedAS \\
freeAS' = freeAS \\
p! = current' \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{ABlock0} \\
\hline
\Delta AScheduler \\
p! : PID \\
\hline
current \neq nullpid \\
current' = nullpid \\
readyAS' = readyAS \\
blockedAS' = blockedAS \cup \{ current \} \\
freeAS' = freeAS \\
p! = current \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{ATimeOut0} \\
\hline
\Delta AScheduler \\
p! : PID \\
\hline
current \neq nullpid \\
current' = nullpid \\
readyAS' = readyAS \cup \{ current \} \\
blockedAS' = blockedAS \\
freeAS' = freeAS \\
p! = current \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{AWakeUp0} \\
\hline
\Delta AScheduler \\
p? : PID \\
\hline
p? \in blockedAS \\
current' = current \\
readyAS' = readyAS \cup \{ p? \} \\
blockedAS' = blockedAS \setminus \{ p? \} \\
freeAS' = freeAS \\
\hline
\end{array}
$$

Destroy is divided in its three valid disjuncts cases.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad ADestroyCurrent \quad}} \\
\Delta AScheduler \\
p? : PID \\
\hline
p? = current \\
current' = nullpid \\
readyAS' = readyAS \\
blockedAS' = blockedAS \\
freeAS' = freeAS \cup \{ p? \} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad ADestroyReady \quad}} \\
\Delta AScheduler \\
p? : PID \\
\hline
p? \in readyAS \\
current' = current \\
readyAS' = readyAS \setminus \{ p? \} \\
blockedAS' = blockedAS \\
freeAS' = freeAS \cup \{ p? \} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad ADestroyBlocked \quad}} \\
\Delta AScheduler \\
p? : PID \\
\hline
p? \in blockedAS \\
current' = current \\
readyAS' = readyAS \\
blockedAS' = blockedAS \setminus \{ p? \} \\
freeAS' = freeAS \cup \{ p? \} \\
\hline
\end{array}
$$

and is composed by disjunction as usual.

$$ADestroy0 \;==\; ADestroyCurrent \lor ADestroyReady \lor ADestroyBlocked$$

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

$$
\begin{aligned}
ASchedMsg \;::=\;& AS\_OKAY \mid AS\_DISPATCH\_ERR \mid AS\_TIMEOUT\_ERR \mid \\
& AS\_BLOCK\_ERR \mid AS\_WAKEUP\_ERR \mid AS\_CREATE\_ERR \mid \\
& AS\_DESTROY\_ERR
\end{aligned}
$$

$$
\begin{aligned}
&ASchedOkay \;==\; [\, aserr! : ASchedMsg \mid aserr! = AS\_OKAY \,] \\
&ASchedErr \;==\; [\, \Xi AScheduler;\; aserr! : ASchedMsg \,] \\
&ACreateErr \;==\; [\, ASchedErr \mid aserr! = AS\_CREATE\_ERR \land freeAS = \varnothing \,] \\
&ADispatchErr \;==\; [\, ASchedErr \mid aserr! = AS\_DISPATCH\_ERR \\
&\qquad \land (current \neq nullpid \lor readyAS = \varnothing) \,] \\
&ABlockErr \;==\; [\, ASchedErr \mid aserr! = AS\_BLOCK\_ERR \land current = nullpid \,] \\
&ATimeOutErr \;==\; [\, ASchedErr \mid aserr! = AS\_TIMEOUT\_ERR \land current = nullpid \,] \\
&AWakeUpErr \;==\; [\, ASchedErr;\; p? : PID \mid aserr! = AS\_WAKEUP\_ERR \land p? \notin blockedAS \,] \\
&ADestroyErr \;==\; [\, ASchedErr;\; p? : PID \mid aserr! = AS\_DESTROY\_ERR \land p? \in freeAS \,]
\end{aligned}
$$

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

$$
\begin{aligned}
&ACreate \;==\; (ACreate0 \land ASchedOkay) \lor ACreateErr \\
&ADispatch \;==\; (ADispatch0 \land ASchedOkay) \lor ADispatchErr \\
&ABlock \;==\; (ABlock0 \land ASchedOkay) \lor ABlockErr \\
&ATimeOut \;==\; (ATimeOut0 \land ASchedOkay) \lor ATimeOutErr \\
&AWakeUp \;==\; (AWakeUp0 \land ASchedOkay) \lor AWakeUpErr \\
&ADestroy \;==\; (ADestroy0 \land ASchedOkay) \lor ADestroyErr
\end{aligned}
$$

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.

theorem disabled rule lScheduledPIDDisjoint
$\forall AScheduler; \ p? : PID \mid \neg \ p? \in blockedAS \land \neg \ p? \in readyAS \land \neg \ p? = current \bullet$
$p? \in freeAS$

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.

$ACreate0Sig == [\, AScheduler \mid freeAS \neq \varnothing \,]$
$ADispatch0Sig == [\, AScheduler \mid current = nullpid \land readyAS \neq \varnothing \,]$
$ATimeOut0Sig == [\, AScheduler \mid current \neq nullpid \,]$
$ABlock0Sig == ATimeOut0Sig$
$AWakeUp0Sig == [\, AScheduler; \ p? : PID \mid p? \in blockedAS \,]$

$ADestroyCurrentSig == [\, AScheduler; \ p? : PID \mid p? = current \,]$
$ADestroyReadySig == [\, AScheduler; \ p? : PID \mid p? \in readyAS \,]$
$ADestroyBlockedSig == [\, AScheduler; \ p? : PID \mid p? \in blockedAS \,]$

$ACreateErrSig == [\, AScheduler \mid freeAS = \varnothing \,]$
$ADispatchErrSig == [\, AScheduler \mid (current \neq nullpid \lor readyAS = \varnothing) \,]$
$ABlockErrSig == [\, AScheduler \mid current = nullpid \,]$
$ATimeOutErrSig == ABlockErrSig$
$AWakeUpErrSig == [\, AScheduler; \ p? : PID \mid p? \notin blockedAS \,]$
$ADestroyErrSig == [\, AScheduler; \ p? : PID \mid p? \in freeAS \,]$

$ASchedTotalSig == [\, AScheduler; \ p? : PID \mid true \,]$

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***]

theorem ASchedulerInitFSB
$\exists AScheduler' \bullet ASchedulerInit$

theorem ADispatch0FSB
$\forall ADispatch0Sig \bullet \mathbf{pre} \ ADispatch0$

theorem ATimeOut0FSB
$\forall ATimeOut0Sig \bullet \mathbf{pre} \ ATimeOut0$

theorem ABlock0FSB
$\forall ABlock0Sig \bullet \mathbf{pre} \ ABlock0$

theorem AWakeUp0FSB
$\forall AWakeUp0Sig \bullet \mathbf{pre} \ AWakeUp0$

theorem ACreate0FSB
$\forall ACreate0Sig \bullet \mathbf{pre}\ ACreate0$

theorem ADestroyCurrentFSB
$\forall ADestroyCurrentSig \bullet \mathbf{pre}\ ADestroyCurrent$

theorem ADestroyReadyFSB
$\forall ADestroyReadySig \bullet \mathbf{pre}\ ADestroyReady$

theorem ADestroyBlockedFSB
$\forall ADestroyBlockedSig \bullet \mathbf{pre}\ ADestroyBlocked$

theorem ADispatchErrFSB
$\forall ADispatchErrSig \bullet \mathbf{pre}\ ADispatchErr$

theorem ATimeOutErrFSB
$\forall ATimeOutErrSig \bullet \mathbf{pre}\ ATimeOutErr$

theorem ABlockErrFSB
$\forall ABlockErrSig \bullet \mathbf{pre}\ ABlockErr$

theorem AWakeUpErrFSB
$\forall AWakeUpErrSig \bullet \mathbf{pre}\ AWakeUpErr$

theorem ACreateErrFSB
$\forall ACreateErrSig \bullet \mathbf{pre}\ ACreateErr$

theorem ADestroyErrFSB
$\forall ADestroyErrSig \bullet \mathbf{pre}\ ADestroyErr$

theorem ADispatchIsTotal
$\forall AScheduler \bullet \mathbf{pre}\ ADispatch$

theorem ATimeOutIsTotal
$\forall AScheduler \bullet \mathbf{pre}\ ATimeOut$

theorem ABlockIsTotal
$\forall AScheduler \bullet \mathbf{pre}\ ABlock$

theorem AWakeUpIsTotal
$\forall ASchedTotalSig \bullet \mathbf{pre}\ AWakeUp$

theorem ACreateIsTotal
$\forall AScheduler \bullet \mathbf{pre}\ ACreate$

theorem ADestroyIsTotal
$\forall ASchedTotalSig \bullet \mathbf{pre}\ ADestroy$

[***Circus tools on***]

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	34	46
Axiomatic definitions	0	2
Generic axiomatic defs.	0	0
Schemas	10	14
Generic schemas	0	0
Theorems	22	43
Proofs	0	0
Total	**66**	**105**

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 implementaiton 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.

HandlerSet0

$$idle : PID$$
$$meh, peh, aeh, oseh : \mathbb{P}\ PID$$

$$idle \in peh$$
$$\langle peh, aeh, oseh \rangle \text{ partition } meh$$

HandlerSet

$$HandlerSet0$$
$$freeHS : \mathbb{P}\ PID$$

$$\langle meh, freeHS \rangle \text{ partition } PID$$

$$IDLE\_PID\_WITNESS \ == \ maxpid$$

[***Circus tools off***]

theorem grule IDLE_PID_WITNESSType
$IDLE\_PID\_WITNESS \in PID$

[***Circus tools on***]

$$
\begin{array}{|l}
\hline
\textit{HandlerSetInit} \\
\textit{HandlerSet}\,' \\
\hline
\textit{meh}' = \{\, \textit{idle}' \,\} \\
\textit{freeHS}' = \textit{PID} \setminus \{\, \textit{idle}' \,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{AddHandler0} \\
\Delta \textit{HandlerSet} \\
p? : \textit{PID} \\
\hline
p? \in \textit{freeHS} \\
\textit{freeHS}' = \textit{freeHS} \setminus \{\, p? \,\} \\
\textit{meh}' = \textit{meh} \cup \{\, p? \,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{ContainsHandler} \\
\Xi \textit{HandlerSet} \\
p? : \textit{PID} \\
\hline
p? \in \textit{meh} \\
\hline
\end{array}
$$

$$\textit{AddHandler} \;\; == \;\; \textit{AddHandler0} \lor \textit{ContainsHandler}$$

$$
\begin{array}{|l}
\hline
\textit{RemoveHandler0} \\
\Delta \textit{HandlerSet} \\
p? : \textit{PID} \\
\hline
p? \in \textit{meh} \land p? \neq \textit{idle} \\
\textit{meh}' = \textit{meh} \setminus \{\, p? \,\} \\
\textit{freeHS}' = \textit{freeHS} \cup \{\, p? \,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UnknownHandler} \\
\Xi \textit{HandlerSet} \\
p? : \textit{PID} \\
\hline
p? \in \textit{freeHS} \lor p? = \textit{idle} \\
\hline
\end{array}
$$

$$\textit{RemoveHandler} \;\; == \;\; \textit{RemoveHandler0} \lor \textit{UnknownHandler}$$

$$
\begin{array}{|l}
\hline
\textit{RemoveAperiodicHandlers} \\
\Delta \textit{HandlerSet} \\
\hline
\textit{meh}' = \textit{meh} \setminus \textit{aeh} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{HandlerSetSig} \\
\textit{HandlerSet} \\
p? : \textit{PID} \\
\hline
\end{array}
$$

[***Circus tools off***]

theorem HandlerSetInitFSB
$\exists HandlerSet' \bullet HandlerSetInit$

theorem AddHandlerFSB
$\forall HandlerSetSig \bullet \mathbf{pre}\ AddHandler$

theorem RemoveHandlerFSB
$\forall HandlerSetSig \bullet \mathbf{pre}\ RemoveHandler$

theorem RemoveAperiodicHandlersFSB
$\forall HandlerSet \bullet \mathbf{pre}\ RemoveAperiodicHandlers$

[***Circus tools on***]

4.2 Priority scheduler of Event handlers

$MIN\_PRIO, MAX\_PRIO : \mathbb{N}_1$

$\langle\!\langle$ disabled rule dMinPrio $\rangle\!\rangle$
$MIN\_PRIO = 1$
$\langle\!\langle$ rule dMinMaxPrio $\rangle\!\rangle$
$MIN\_PRIO \leq MAX\_PRIO$

$PRIORITY == MIN\_PRIO \mathbin{..} MAX\_PRIO$

[***Circus tools off***]

theorem grule MIN_PRIOgeq1
$MIN\_PRIO \geq 1$

theorem grule MAX_PRIOgeq1
$MAX\_PRIO \geq 1$

theorem grule PRIORITYMaxType
$PRIORITY \in \mathbb{P}\ \mathbb{Z}$

theorem grule PRIORITYType
$PRIORITY \in \mathbb{P}\ \mathbb{N}$

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.

$$
\begin{array}{|l}
\multicolumn{1}{l}{PriorityScheduler0} \\
\hline
AScheduler \\
HandlerSet \\
idle : PID \\
prio : \mathrm{seq}_1 (\mathrm{iseq}\ PID) \\
\hline
idle \in peh \\
\mathrm{dom}\ prio = PRIORITY \\
meh = \bigcup (\mathrm{ran} (prio \mathbin{;} \mathrm{ran})) \\
\langle \{current\} \setminus \{nullpid\}, meh, freeHS \rangle \ \mathrm{partition}\ PID \\
\hline
\end{array}
$$

[***Circus tools off***]

$$PRIO\_WITNESS \ == \ (\lambda x : PRIORITY \bullet \langle\rangle) \oplus \langle\langle IDLE\_PID\_WITNESS \rangle\rangle$$

theorem grule PRIO_WITNESSMaxType
$PRIO\_WITNESS \in \mathbb{P} (\mathbb{Z} \times \mathbb{P} (\mathbb{Z} \times \mathbb{Z}))$

theorem grule PRIO_WITNESSRelType
$PRIO\_WITNESS \in \mathbb{Z} \leftrightarrow \mathrm{iseq}\ PID$

theorem grule PRIO_WITNESSPfunType
$PRIO\_WITNESS \in \mathbb{N} \rightarrow\!\!\!\!\!+ \ \mathrm{iseq}\ PID$

theorem rule PRIO_WITNESSElement
$(1, \langle IDLE\_PID\_WITNESS \rangle) \in PRIO\_WITNESS$

theorem rule PRIO_WITNESSdom
$\mathrm{dom}\ PRIO\_WITNESS = PRIORITY$

theorem grule PRIO_WITNESSType
$PRIO\_WITNESS \in \mathrm{seq}_1 (\mathrm{iseq}\ PID)$

theorem rule PRIO_WITNESS_IDLE_WITNESSEquiv
$\bigcup (\mathrm{ran} (PRIO\_WITNESS \mathbin{;} \mathrm{ran})) = \{IDLE\_PID\_WITNESS\}$

[***Circus tools on***]

$$
\begin{array}{|l}
\multicolumn{1}{l}{PriorityScheduler0Init} \\
\hline
PriorityScheduler0\,' \\
AScheduler\,' \\
\hline
current' = nullpid \\
readyAS' = \{idle'\} \\
blockedAS' = \varnothing \\
HandlerSetInit \\
prio' = (\lambda x : PRIORITY \bullet \langle\rangle) \oplus \langle\langle idle' \rangle\rangle \\
\hline
\end{array}
$$

[***Circus tools off***]

theorem PriorityScheduler0InitFSB
$\exists PriorityScheduler0' \bullet PriorityScheduler0Init$

[***Circus tools on***]

$$collectPIDS : (PRIORITY \pfun \operatorname{iseq} PID) \fun (PRIORITY \pfun \power PID)$$
$$\langle\langle \text{disabled rule dCollectPIDS} \rangle\rangle$$
$$\forall prio : PRIORITY \pfun \operatorname{iseq} PID \bullet collectPIDS\ prio = (\lambda x : \operatorname{dom} prio \bullet \operatorname{ran}(prio\ x))$$

[***Circus tools off***]

theorem rule lCollectPIDSIsTotal
$\forall prio : PRIORITY \pfun \operatorname{iseq} PID \bullet prio \in \operatorname{dom} collectPIDS$

[***Circus tools on***]

$$flattenPIDS : (PRIORITY \pfun \operatorname{iseq} PID) \fun \power PID$$
$$\langle\langle \text{disabled rule dFlattenPIDS} \rangle\rangle$$
$$\forall prio : PRIORITY \pfun \operatorname{iseq} PID \bullet flattenPIDS\ prio = \bigcup(\operatorname{ran}(collectPIDS\ prio))$$

[***Circus tools off***]

theorem rule lFlattenPIDSIsTotal
$\forall prio : PRIORITY \pfun \operatorname{iseq} PID \bullet prio \in \operatorname{dom} flattenPIDS$

[***Circus tools on***]

The priority scheduler combines the abstract scheduler, the managed event handlers set, and the VM. This is so that all known *PID*s share a common partitioning invariant, namely: their *free* *PID*s are the same; and each has their own policy for partitioning of *PID*s[1].

PrioSched0Comp
$$AScheduler;\ HandlerSet;\ VM$$
$$free : \power PID$$

PrioSched1ASHS
$$PrioSched0Comp$$
$$current \neq nullpid \implies current \in meh$$

PrioSched2Prio
$$PrioSched1ASHS$$
$$schedPIDs : \operatorname{iseq} PID$$
$$prio : PRIORITY \pfun \operatorname{iseq} PID$$
$$\operatorname{ran} schedPIDs = flattenPIDS\ prio$$
$$meh = usedVM \cup createdVM = flattenPIDS\ prio$$

[1]This separation of concepts is also useful for taming proof expansion and equality substitution.

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad PriorityScheduler \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
PrioSched2Prio \\
\hline
free = freeAS = freeHS = freeVM \\
\hline
\end{array}$$

[***Circus tools off***]

theorem rule fPRIOResMaxRelType
$\forall prio : PRIORITY \pinj \mathrm{iseq}\ PID;\ pr : PRIORITY \mid pr \in \mathrm{dom}\ prio \bullet (prio\ pr) \in \mathbb{Z} \leftrightarrow \mathbb{Z}$

theorem rule fPRIOResMaxPfunType
$\forall prio : PRIORITY \pinj \mathrm{iseq}\ PID;\ pr : PRIORITY \mid pr \in \mathrm{dom}\ prio \bullet (prio\ pr) \in \mathbb{Z} \pfun \mathbb{Z}$

theorem rule fPSPrioResSeqMaxType
$\forall prio : PRIORITY \pinj \mathrm{iseq}\ PID;\ pr : PRIORITY \mid pr \in \mathrm{dom}\ prio \bullet prio\ pr \in \mathrm{seq}\ \mathbb{Z}$

theorem rule fPSPrioResSeqType
$\forall prio : PRIORITY \pinj \mathrm{iseq}\ PID;\ pr : PRIORITY \mid pr \in \mathrm{dom}\ prio \bullet prio\ pr \in \mathrm{seq}\ PID$

theorem rule fPSPrioResISeqType
$\forall prio : PRIORITY \pinj \mathrm{iseq}\ PID;\ pr : PRIORITY \mid pr \in \mathrm{dom}\ prio \bullet prio\ pr \in \mathrm{iseq}\ PID$

[***Circus tools on***]

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad PSAdd0 \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\Delta PriorityScheduler \\
p? : PID \\
pr? : PRIORITY \\
\hline
p? \in free \\
schedPIDs' = schedPIDs \frown \langle p? \rangle \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad PSAddKnownPrio \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
PSAdd0 \\
\hline
pr? \in \mathrm{dom}\ prio \\
prio' = prio \oplus \{ pr? \mapsto prio\ pr? \frown \langle p? \rangle \} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad PSAddNewPrio \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
PSAdd0 \\
\hline
pr? \notin \mathrm{dom}\ prio \\
prio' = prio \oplus \{ pr? \mapsto \langle p? \rangle \} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad PSAddErr \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\Xi PriorityScheduler;\ p? : PID \\
vmerr! : VMError;\ aserr! : ASchedMsg \\
\hline
p? \notin free \\
vmerr! = VMOutOfProcess \\
aserr! = AS_CREATE_ERR \\
\hline
\end{array}$$

$$PSAddPrio0 \;==\; (PSAddNewPrio \lor PSAddKnownPrio)$$

$$PSAddPrio \;==\; PSAddPrio0 \land ACreate \land VMCreate \land AddHandler[aeh/aeh']$$
$$PSAdd \;==\; PSAddPrio \lor PSAddErr$$

$$\begin{array}{|l}
\multicolumn{1}{l}{PSDispatch0} \\
\hline
\Delta PriorityScheduler \\
p! : PID \\
\hline
p! \in \mathrm{ran}\ schedPIDs \\
\hline
\end{array}$$

$$PSDispatch \;==\; PSDispatch0$$

4.3 Priority scheduler operations preconditions

$$\begin{array}{|l}
\multicolumn{1}{l}{PSAddSig} \\
\hline
PriorityScheduler \\
p? : PID \\
pr? : PRIORITY \\
\hline
true \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{PSDispatchSig} \\
\hline
PriorityScheduler \\
\hline
true \\
\hline
\end{array}$$

[***Circus tools off***]

theorem rule lVMAllAlocated
$\forall VM;\ p? : PID \mid freeVM = \{\} \land \lnot p? \in createdVM \bullet p? \in usedVM$

theorem rule lHSFreeIsUnknown
$\forall HandlerSet;\ p? : PID \mid p? \in freeHS \bullet \lnot p? \in meh$

theorem frule fVMInvariant
$\forall VM \bullet \langle createdVM, usedVM, freeVM \rangle \text{ partition } PID$

theorem frule fPSFreeInvariant
$\forall PriorityScheduler \bullet free = freeAS = freeVM = freeHS$

theorem frule fASInvariant
$\forall AScheduler \bullet$
$\langle \{ current \} \setminus \{ nullpid \}, readyAS, blockedAS, freeAS \rangle \text{ partition } PID$

theorem rule fASInvariantFreeNullDisjoint
$\forall AScheduler;\ p? : PID \mid current = nullpid \land p? \in freeAS \bullet$
$(\mathrm{disjoint}\langle\{\}\rangle \frown (\langle readyAS \cup \{p?\}\rangle \frown (\langle blockedAS\rangle \frown \langle(freeAS \setminus \{p?\})\rangle)))$

theorem rule fASInvariantFreeNonNullDisjoint
$\forall AScheduler;\ p? : PID \mid \neg\ current = nullpid \land p? \in freeAS \bullet$
$(\mathrm{disjoint}\langle\{current\}\rangle \frown (\langle readyAS \cup \{p?\}\rangle \frown (\langle blockedAS\rangle \frown \langle(freeAS \setminus \{p?\})\rangle)))$

theorem rule fASInvariantRewrittenNullPartition
$\forall AScheduler \mid current = nullpid \bullet blockedAS \cup (freeAS \cup readyAS) = PID$

theorem rule fASInvariantRewrittenNonNullPartition
$\forall AScheduler;\ p? : PID \mid \neg\ current = nullpid \bullet$
$blockedAS \cup (freeAS \cup (readyAS \cup \{current\})) = PID$

theorem frule fHS0Invariant
$\forall HandlerSet0 \bullet \langle peh, aeh, oseh\rangle \ \mathrm{partition}\ meh$

theorem frule fHSInvariant
$\forall HandlerSet \bullet \langle meh, freeHS\rangle \ \mathrm{partition}\ PID$

theorem lPSAddPropFreeIsUnknown
$\forall PSAddSig \mid p? \in free \bullet \neg\ p? \in aeh \land \neg\ p? \in oseh \land$
$\neg\ p? \in blockedAS \land \neg\ p? \in usedVM \land$
$\neg\ p? \in peh$

theorem rule lCollectPIDSDom
$\forall prio : PRIORITY \nrightarrow \mathrm{iseq}\ PID \bullet \mathrm{dom}\,(collectPIDS\ prio) = \mathrm{dom}\ prio$

theorem rule lCollectPIDSApply
$\forall prio : PRIORITY \nrightarrow \mathrm{iseq}\ PID;\ pr : PRIORITY \mid$
$pr \in \mathrm{dom}\ prio \bullet (collectPIDS\ prio)\ pr = \mathrm{ran}\,(prio\ pr)$

theorem rule lRanInCollectPIDS
$\forall prio : PRIORITY \nrightarrow \mathrm{iseq}\ PID;\ pr : PRIORITY \mid$
$pr \in \mathrm{dom}\ prio \bullet \mathrm{ran}\,(prio\ pr) \in \mathrm{ran}\,(collectPIDS\ prio)$

theorem rule lCollectPIDSOverrideKnownDom
$\forall prio : PRIORITY \nrightarrow \mathrm{iseq}\ PID;\ pr? : PRIORITY;\ p? : PID \mid$
$\neg\ p? \in \mathrm{ran}\,(prio\ pr?) \land pr? \in \mathrm{dom}\ prio \bullet$
$collectPIDS\,(prio \oplus \{(pr?, (prio\ pr? \frown \langle p?\rangle))\}) =$
$collectPIDS\ prio \oplus \{\,(pr?, \mathrm{ran}\,(prio\ pr?) \cup \{\,p?\,\})\,\}$

theorem rule lCollectPIDSOverrideUnknownDom

$$\forall\, prio : PRIORITY \pfun \mathrm{iseq}\ PID;\ pr? : PRIORITY;\ p? : PID \mid \\ \lnot\ pr? \in \mathrm{dom}\ prio \bullet \\ collectPIDS\,(prio \oplus \{(pr?, (\langle p? \rangle))\}) = \\ collectPIDS\ prio \oplus \{\,(pr?, \{\, p? \,\})\,\}$$

theorem rule lRanFlattenPIDSKnownDom

$$\forall\, prio : PRIORITY \pfun \mathrm{iseq}\ PID;\ pr? : PRIORITY;\ p? : PID \mid \\ pr? \in \mathrm{dom}\ prio \bullet \mathrm{ran}\,(prio\ pr?) \in \mathbb{P}\,(flattenPIDS\ prio)$$

theorem rule lFlattenPIDSOverrideKnownDom

$$\forall\, prio : PRIORITY \pfun \mathrm{iseq}\ PID;\ pr? : PRIORITY;\ p? : PID \mid \\ pr? \in \mathrm{dom}\ prio \land \lnot\ p? \in \mathrm{ran}\,(prio\ pr?) \bullet \\ flattenPIDS\,(prio \oplus \{(pr?, (prio\ pr? \frown \langle p? \rangle))\}) = \\ flattenPIDS\ prio \cup \{\, p? \,\} \cup \mathrm{ran}\,(prio\ pr?)$$

theorem rule lFlattenPIDSOverrideUnknownDom

$$\forall\, prio : PRIORITY \pfun \mathrm{iseq}\ PID;\ pr? : PRIORITY;\ p? : PID \mid \\ \lnot\ pr? \in \mathrm{dom}\ prio \bullet \\ flattenPIDS\,(prio \oplus \{(pr?, (\langle p? \rangle))\}) = \\ flattenPIDS\ prio \cup \{\, p? \,\}$$

theorem lPSAddPropUniquePrio

$$\forall\, PSAddSig \mid pr? \in \mathrm{dom}\ prio \land p? \in free \bullet \lnot\ p? \in \mathrm{ran}\,(prio\ pr?)$$

theorem PSAddFSB

$$\forall\, PSAddSig \bullet \mathbf{pre}\ PSAdd$$

theorem PSDispatchFSB

$$\forall\, PSDispatchSig \bullet \mathbf{pre}\ PSDispatch$$

[***Circus tools on***]

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	9	55
Axiomatic definitions	10	12
Generic axiomatic defs.	0	0
Schemas	22	36
Generic schemas	0	0
Theorems	48	91
Proofs	0	0
Total	**89**	**194**

Table 4.1: Summary of Z declarations for Chapter 4.

Chapter 5

SCJ priority scheduler implementation in *Circus*

$\textbf{section}\ ic\_sched\_impl\ \textbf{parents}\ circus\_toolkit, scj\_process$

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

$\textbf{channel}\ VMthrow, PCIHcatchError : EXCEPTION$

$$SCHID : \mathbb{P}_1\ SCHID0$$
$$CLOCKID : \mathbb{P}_1\ CLOCKID0$$

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 $\mathbb{N}$, and *now* to correspond to what javax.realtime.AbsoluteTime.getTime() returns. *model that fully later?*

$$TIME == \mathbb{N}$$

$$now : TIME$$

$$SCJPID : \mathbb{P}_1\ PID$$

$$nullsch : SCHID$$
$$nullclock : CLOCKID$$

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 $1 \mathinner{\ldotp\ldotp} 150$ (see javax.scj.util.Priorities). *how to reflect that?*

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 $PVMcreateProcess : PID$
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 $PCIHregisterCall, PCIHregisterRet, PCIHyieldCall, PCIHyieldRet, PCIHenableCall, PCIHenableRet, P$
channel $PCIHinstance : CLOCKID$
channel $PCIHstartClockHandlerCall : PID$
channel $PCIHstartClockHandlerRet$

channel $FIXME$

5.3 SCJ priority scheduler bridge

$\textbf{channel}\ PPSIgetNextProcessCall$
$\textbf{channel}\ PPSIgetNextProcessRet : PID$

$\textbf{process}\ PrioSchedulerImpl \mathrel{\widehat=} \textbf{begin}$

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.

$$
\begin{array}{l}
\textit{PSIState} \\
\hline
instance : SCHID \\
current : GPID \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{InitPSIState} \\
\hline
PSIState' \\
b? : SCHID \\
\hline
instance' = b? \\
current' = nullpid \\
\hline
\end{array}
$$

$\textbf{state}\ PSIState$

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.

$$
\begin{array}{l}
GetNextProcess \mathrel{\widehat=} PCIHdisableCall \longrightarrow PCIHdisableRet \longrightarrow \\
\qquad\qquad KPSmoveCall \longrightarrow KPSmoveRet?p \longrightarrow current := p \\
\quad ; \\
\textbf{if}(current = nullpid) \longrightarrow \\
\quad KPSgetCurrentProcess?current \longrightarrow KPSstopCall!current \longrightarrow KPSstopRet \longrightarrow \textbf{Skip} \\
[\!]\,(current \neq nullpid) \longrightarrow \\
\quad PCIHenableCall \longrightarrow PCIHenableRet \longrightarrow \textbf{Skip} \\
\textbf{fi}
\end{array}
$$

$$InitSt \mathrel{\widehat=} ENVcreatePSBridge?b : (b \neq nullsch) \longrightarrow \big(InitPSIState\big)$$

$$InitCIH \mathrel{\widehat=} PCIHinstance?c : (c \neq nullclock) \longrightarrow \textbf{Skip}$$

$$InitPSISt \mathrel{\widehat=} InitSt$$

$$PSIRun \mathrel{\widehat=} \mu X \bullet PPSIgetNextProcessCall \longrightarrow GetNextProcess \,;\; PPSIgetNextProcessRet!current \longrightarrow X$$

$\bullet\ InitPSISt \,;\; PSIRun$

$\textbf{end}$

5.4 SCJ priority scheduler process

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

$$\textbf{process}\ \mathit{PrioScheduler} \mathrel{\widehat=} \textbf{begin}$$

5.4.1 SCJ process abstraction

SCJ process is an SCJ abstraction used to encapsulate a vm.Process within the various javax.safetycritical.ManagedSchedu 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.

$$\begin{aligned}&\mathit{SCJSTATE} ::= \mathit{NEW} \mid \mathit{READY} \mid \mathit{EXECUTING} \mid \mathit{BLOCKED} \mid\\ &\mathit{SLEEPING} \mid \mathit{HANDLED} \mid \mathit{TERMINATED}\end{aligned}$$

ScjProcess

$$\begin{aligned}&\mathit{state} : \mathit{SCJSTATE}\\ &\mathit{nextActivation} : \mathit{TIME}\\ &\mathit{target} : \mathit{GPID}\end{aligned}$$

InitScjProcess

$$\begin{aligned}&\mathit{ScjProcess}'\\ &h? : \mathit{GPID}\\ \hline &h? \neq \mathit{nullpid}\\ &\mathit{target}' = h?\\ &\mathit{state}' = \mathit{NEW}\end{aligned}$$

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**.

Queue0

$$\begin{aligned}&\mathit{heapSize} : \mathbb{N}\\ &\mathit{tree} : \mathrm{seq}\ \mathit{PID}\end{aligned}$$

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 $0 \mathrel{..} \mathit{capacity}$ or $\mathit{capacity} + 1$ elements; in Z, we have a sequence from $1 \mathrel{..} \mathit{capacity} + 1$).

InitQueue0

$$\begin{aligned}&\mathit{Queue0}'\\ &\mathit{capacity}? : \mathbb{N}\\ \hline &\mathit{heapSize}' = 0\\ &\mathit{tree}' = (\lambda\, x : 1 \mathrel{..} (\mathit{capacity}? + 1) \bullet \mathit{nullpid})\end{aligned}$$

This is convoluted and will lead to unecessarily complex proofs. I will use a partial sequence instead of a total sequence with null pointers inside.

relation$(\_ <_{sleep} \_)$
relation$(\_ >_{prio} \_)$

$$\begin{array}{|l}
_ >_{prio} _ : PID \leftrightarrow PID \\
_ <_{sleep} _ : PID \leftrightarrow PID \\
prio : PID \rightarrow \mathbb{N}_1 \\
sleep : PID \rightarrow TIME \\
\hline
\forall s1, s2 : PID \bullet s1 <_{sleep} s2 \Leftrightarrow sleep\ s1 < sleep\ s2 \\
\forall p1, p2 : PID \bullet p1 >_{prio} p2 \Leftrightarrow prio\ p1 > prio\ p2
\end{array}$$

Queue

$$\begin{array}{|l}
heapSize, heapCapacity : \mathbb{N} \\
tree : \mathrm{seq}\ PID \\
\hline
heapSize \leq heapCapacity \\
\mathrm{dom}\ tree \subseteq 1 \mathinner{..} heapSize
\end{array}$$

PQueue

$$\begin{array}{|l}
Queue \\
\hline
\forall i, j : \mathrm{dom}\ tree \mid i < j \bullet tree\ i >_{prio} tree\ j
\end{array}$$

SQueue

$$\begin{array}{|l}
Queue \\
\hline
\forall i, j : \mathrm{dom}\ tree \mid i < j \bullet tree\ i <_{sleep} tree\ j
\end{array}$$

InitQueue

$$\begin{array}{|l}
Queue' \\
capacity? : \mathbb{N} \\
\hline
heapSize' = 0 \\
heapCapacity' = capacity? \\
tree' = \langle\rangle
\end{array}$$

QueueEmpty

$$\begin{array}{|l}
\Xi Queue \\
p! : PID \\
\hline
heapSize \leq 0 \\
p! = nullpid
\end{array}$$

QueuePeek0

$$\begin{array}{|l}
\Xi Queue \\
p! : PID \\
\hline
heapSize > 0 \\
p! = head\ tree
\end{array}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{FixedCapacityQueue}}} \\
\Delta Queue \\
\hline
heapCapacity' = heapCapacity \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{QueueExtract0}}} \\
FixedCapacityQueue \\
p! : PID \\
\hline
heapSize > 0 \\
p! = head\ tree \\
tree' = tail\ tree \\
heapSize' = heapSize - 1 \\
\hline
\end{array}
$$

$$
\begin{array}{l}
QueuePeek \ == \ QueuePeek0 \lor QueueEmpty \\
QueueExtract \ == \ QueueExtract0 \lor QueueEmpty
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\textit{QueueInsert}}} \\
FixedCapacityQueue \\
p? : PID \\
\hline
heapSize < heapCapacity \\
heapSize' = heapSize + 1 \\
tree' = tree \frown \langle p? \rangle \\
\hline
\end{array}
$$

$$
\begin{array}{l}
PQueueInit \ == \ (InitQueue \land PQueue') \\
SQueueInit \ == \ (InitQueue \land SQueue') \\
PQueueExtractMax \ == \ (QueueExtract \land \Delta PQueue) \\
SQueueExtractMin \ == \ (QueueExtract \land \Delta SQueue) \\
PQueueInsert \ == \ (QueueInsert \land \Delta PQueue) \\
SQueueInsert \ == \ (QueueInsert \land \Delta SQueue) \\
SQueuePeek \ == \ (QueuePeek \land \Delta SQueue)
\end{array}
$$

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...

$$
\begin{array}{l}
PrioQueue \ == \ PQueue[ptree/tree, pheapSize/heapSize, pheapCapacity/heapCapacity] \\
SleepQueue \ == \ SQueue[stree/tree, sheapSize/heapSize, sheapCapacity/heapCapacity] \\
PQInit \ == \ PQueueInit[ptree'/tree', pheapSize'/heapSize', pheapCapacity'/heapCapacity'] \\
SQInit \ == \ SQueueInit[stree'/tree', sheapSize'/heapSize', sheapCapacity'/heapCapacity'] \\
PQExtractMax \ == \ PQueueExtractMax[ptree/tree, pheapSize/heapSize, pheapCapacity/heapCapacity, ptree'/tr \\
SQExtractMin \ == \ SQueueExtractMin[stree/tree, sheapSize/heapSize, sheapCapacity/heapCapacity, stree'/tree' \\
PQInsert \ == \ PQueueInsert[ptree/tree, pheapSize/heapSize, pheapCapacity/heapCapacity, ptree'/tree', pheapSi \\
SQInsert \ == \ SQueueInsert[ptree/tree, pheapSize/heapSize, pheapCapacity/heapCapacity, stree'/tree', sheapSiz
\end{array}
$$

[***Circus tools off***]

theorem test
$PQExtractMax$

proof[*test*]
invoke PQExtractMax;
invoke PQueueExtractMax;
invoke QueueExtract;
invoke Δ *PQueue*;
disjunctive;
cases;
*invoke QueueExtract*0;
invoke FixedCapacityQueue;
invoke Δ *Queue*;
invoke PQueue;
prove;
rewrite;
■

[***Circus* tools on**]

$$
\begin{array}{|l}
\textit{PriorityFrame} \\
\hline
\textit{PrioQueue} \\
\textit{SleepQueue} \\
\hline
\operatorname{ran} \textit{ptree} \cap \operatorname{ran} \textit{stree} = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{InitPrioFrame} \\
\hline
\textit{PriorityFrame}' \\
\textit{PQInit} \\
\textit{SQInit} \\
\hline
\end{array}
$$

5.4.3 Priority scheduler state

The state is composed of the various declared fields.

$$
\begin{array}{|l}
\textit{SchInfo} \\
\hline
\textit{cih} : \textit{CLOCKID} \\
\textit{rtc} : \textit{TIME} \\
\textit{instance}, \textit{bridge} : \textit{SCHID} \\
\hline
\textit{instance} = \textit{bridge} \Rightarrow \textit{instance} = \textit{nullsch} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{ProcInfo} \\
\hline
\textit{current}, \textit{mainProcess}, \textit{outerMostSeqProcess} : \textit{SCJPID} \\
\textit{seh} : \mathbb{P}_1\ \textit{SCJPID} \\
\textit{procs} : \textit{PID} \pinj \textit{ScjProcess} \\
\hline
\textit{seh} = \{\, \textit{current}, \textit{mainProcess}, \textit{outerMostSeqProcess} \,\} \\
\forall\, i : \operatorname{dom} \textit{procs} \bullet i = (\textit{procs}\ i).\textit{target} \\
\hline
\end{array}
$$

The *scheduler* singleton static field is used to access the scheduler object instance from other classes. Here we model this with an $\textit{instance} \in \textit{SCHID}$ (in Java, this is through the instance() method). The bridge class javax.safetycritical.PrioritySchedulerImpl is used to link the underlying vm.Scheduler interface, with the SCJ run-time environment (RTE) scheduler interface (javax.realtime.PriorityScheduler). This is captured here with another scheduler id named *bridge*. Where appropriate, communication using these ids will represent which scheduler is being refered to. This is to both represent the *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 vm.Process representing the various SCJ processes (javax.safetycritical.ScjProcess), which themselves are encapsulated by the various SCJ infrastructure objects (i.e. javax.safetycritical.PeriodicEventHandler, javax.safetycritical.Mission, javax.safetycritical.MissionSequencer, etc.). *have a picture with this Java process abstract encapsulation?*

$PSState$
$SchInfo$
$ProcInfo$
$PriorityFrame$
$HandlerSet$

$meh \cap seh = \varnothing$
$\mathrm{dom}\ procs = meh \cup seh \setminus \{ nullpid \}$

The scheduler maintains a reference to three javax.safetycritical.ScjProcess objects, which wrap a corresponding vm.Process. Here these are represented simply by $SCJPID$ references, which are a subset of PID to correspond to each class instances respectively. The $mainProcess$ represents the line of execution of the scheduler itself. It is to this ID that the clock interrupt handler returns control after scheduling. The $current$ 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 (getCurrentProcess()). The only relevant place it is used is to call the stop(vm.Process) method from the priority scheduler bridge. Arguably you could call it with a different PID, yet that would mean asking the scheduler to stop a process it does not manage, hence there should be no parameter to the stop() method. The $outerMostSeqProcess$ 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 scaterred around various classes aparently for convenience access. For instance, the handler that each $ScjProcess$ represents is put inside javax.safetycritical.Mission indirectly via another class (javax.safetycritical.ManagedSchedulableSet). We keep it here using the $HandlerSet$ abstraction defined before: it separates what kind of handler instantiated $ScjProcess$es represent, effectively bringing the javax.safetycritical.ManagedSchedulableSet from javax.safetycritical.Mission to the 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 javax.safetycritical.ScjProcess. Thus, removing this structure here brings no discrepancy as far as scheduling is concerned.

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

The state is initialised with the $Init$ 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 $PSState$

$InitSchInfo$
$SchInfo'$

$cih' = nullclock$
$rtc' = 0$
$instance' = bridge' = nullsch$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitProcInfo}} \\
\hline
ProcInfo' \\
\hline
current' = mainProcess' = outerMostSeqProcess' = nullpid \\
seh' = \{ current', mainProcess', outerMostSeqProcess' \} \\
procs' = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InstantiatePS}} \\
\hline
PSState' \\
InitSchInfo \\
InitProcInfo \\
InitPrioFrame \\
HandlerSetInit \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InitSchIds}} \\
\hline
\Delta PSState \\
\Xi ProcInfo \\
\Xi PriorityFrame \\
\Xi HandlerSet \\
i?, b? : SCHID \\
\hline
i? \neq nullsch \land b? \neq nullsch \\
i? \neq b? \\
instance' = i? \\
bridge' = b? \\
cih' = cih \land rtc' = rtc \\
\hline
\end{array}
$$

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

$$
\begin{aligned}
STACK\_UNIT &== 1024 \\
PS\_STACK\_SIZE &== 1 * STACK\_UNIT \\
DEFAULT\_PQUEUE\_SIZE &== 20
\end{aligned}
$$

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.

$$
\begin{aligned}
InitSt \mathrel{\widehat=} {} & ENVinstancePS?i : (i \neq nullsch) \longrightarrow ENVcreatePFrame \longrightarrow \\
& ENVcreatePSBridge?b : (b \neq nullsch) \longrightarrow \big(InitSchIds\big)
\end{aligned}
$$

$$
InitClock \mathrel{\widehat=} ENVinstanceRTC?clock \longrightarrow rtc := clock
$$

$$
Init \mathrel{\widehat=} InitSt \; ; \; InitClock
$$

5.4.4 Various state operations

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{InsertReadyQueueOp}} \\
\hline
\Delta PSState \\
\Xi SchInfo \\
\Xi ProcInfo \\
\Xi HandlerSet \\
PQInsert \\
\hline
\end{array}
$$

[***Circus tools off***]

$\textbf{theorem}$ test2
$InsRQueueOp$

proof[*test*]
$invoke\ InsRQueueOp;$
$invoke\ \Delta\ PriorityFramse;$
$invoke\ PQInsert;$
$invoke\ PQueueInsert;$
$invoke\ QueueInsert;$
$invoke\ \Delta\ PQueue;$
$disjunctive;$
$cases;$
$invoke\ QueueInsert0;$
$invoke\ FixedCapacityQueue;$
$invoke\ \Delta\ Queue;$
$invoke\ PQueue;$
$prove;$
$rewrite;$
$\blacksquare$

[***Circus tools on***]

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.

$$Move \mathrel{\widehat=} \mathbf{var}\ next : PID \bullet KPSmoveCall \longrightarrow FIXME \longrightarrow KPSmoveRet!next \longrightarrow \mathbf{Skip}$$

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.

$$\begin{aligned} SCJStop \mathrel{\widehat=}\ & KPSstopCall?curr : (curr \neq nullpid) \longrightarrow \\ & PVMtransferTo!curr!mainProcess \longrightarrow KPSstopRet \longrightarrow \mathbf{Skip} \end{aligned}$$

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

$$\begin{aligned} &\mathbf{channelset}\ csMove == \{\!|\ KPSmoveCall, KPSmoveRet\ |\!\} \\ &\mathbf{channelset}\ csSCJStop == \{\!|\ KPSstopCall, KPSstopRet\ |\!\} \\ &\mathbf{channelset}\ csCIHApi == csMove \cup csSCJStop \end{aligned}$$

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.

$$AddOuterMostSeq \mathrel{\widehat=} \mathbf{Skip}$$

$$\begin{aligned} Start \mathrel{\widehat=} \mathbf{var}\ p : PID \bullet KPSstartCall \longrightarrow \big(PQExtractMax\big)\ ;\ & current := p; \\ & PVMcreateProcess?m : (m \neq nullpid) \longrightarrow mainProcess := m; \\ & PCIHregisterCall \longrightarrow PCIHregisterRet \longrightarrow \\ & PCIHenableCall \longrightarrow PCIHenableRet \longrightarrow \\ & PCIHstartClockHandlerCall!mainProcess \longrightarrow \\ & PCIHstartClockHandlerRet \longrightarrow \\ & PCIHyieldCall \longrightarrow PCIHyieldRet \longrightarrow KPSstartRet \longrightarrow \mathbf{Skip} \end{aligned}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FixedProcInfo}} \\
\Delta ProcInfo \\
\hline
current' = current \\
mainProcess' = mainProcess \\
outerMostSeqProcess' = outerMostSeqProcess \\
seh' = seh \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ReleaseHandler}} \\
\Delta PSState \\
FixedProcInfo \\
\Xi SchInfo \\
\Xi PriorityFrame \\
\Xi HandlerSet \\
apeh? : PID \\
\hline
apeh? \in \operatorname{dom} procs \\
(procs\ apeh?).state = BLOCKED \Rightarrow PQInsert[apeh?/p?] \\
\hline
\end{array}
$$

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)?

$$ReleaseHandlerNo \mathrel{\widehat=} apeh : PID \bullet \left(apeh \in \operatorname{dom} procs\right) \mathbin{\&} \mathbf{if}((procs\ apeh).state = EXECUTING) \longrightarrow FIXME -$$

$$
\begin{array}{l}
Release \mathrel{\widehat=} KPSreleaseCall?apeh \longrightarrow PCIHdisableCall \longrightarrow PCIHdisableRet \longrightarrow \\
\qquad \left(apeh \in \operatorname{dom}\right) \mathbin{\&} \left(ReleaseHandler\right); \\
\qquad\qquad PCIHenableCall \longrightarrow PCIHenableRet \longrightarrow KPSreleaseRet \longrightarrow \mathbf{Skip}
\end{array}
$$

$$GetCurrentProc \mathrel{\widehat=} KPSgetCurrentProcess!current \longrightarrow \mathbf{Skip}$$

$$
\begin{array}{l}
InsertReadyQueue \mathrel{\widehat=} KPSinsertReadyQueueCall?p \longrightarrow \left(InsertReadyQueueOp\right); \\
\qquad\qquad KPSinsertReadyQueueRet \longrightarrow \mathbf{Skip}
\end{array}
$$

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

$$
\begin{array}{l}
\mathbf{channelset}\ csStart == \{\!|\ KPSstartCall, KPSstartRet\ |\!\} \\
\mathbf{channelset}\ csAddOuterMostSeq == \{\!|\ KPSaddOuterMostSeqCall, KPSaddOuterMostSeqRet\ |\!\} \\
\mathbf{channelset}\ csGetCurrentProcess == \{\!|\ KPSgetCurrentProcess\ |\!\} \\
\mathbf{channelset}\ csInsertReadyQueue == \{\!|\ KPSinsertReadyQueueCall, KPSinsertReadyQueueRet\ |\!\} \\
\mathbf{channelset}\ csRelease == \{\!|\ KPSreleaseCall, KPSreleaseRet\ |\!\} \\
\mathbf{channelset}\ csSCJRTE == csStart \cup csAddOuterMostSeq \cup csGetCurrentProcess \cup \\
\qquad\qquad csInsertReadyQueue \cup csRelease
\end{array}
$$

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 javax.scj.util.Priorities as final static contansts.

$$
\begin{array}{|l}
SCJMIN\_PRIO == 1 \\
SCJMAX\_PRIO == 100 \\
\\
MIN\_HW\_PRIO == 101 \\
MAX\_HW\_PRIO == 150
\end{array}
$$

The actions just offer these constants through the appropriate channel accordingly.

$$GetHWPrio \mathrel{\widehat=} PPSgetMinHWPrio!MIN\_HW\_PRIO \longrightarrow \mathbf{Skip} \\ \Box \\ PPSgetMaxHWPrio!MAX\_HW\_PRIO \longrightarrow \mathbf{Skip}$$

$$GetPrio \mathrel{\widehat=} PPSgetMinPrio!MIN\_PRIO \longrightarrow \mathbf{Skip} \\ \Box \\ PPSgetMaxPrio!MAX\_PRIO \longrightarrow \mathbf{Skip}$$

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

$$\begin{array}{l} \mathbf{channelset}\ csGetHWPrio\ == \ \{\!|\ PPSgetMinHWPrio, PPSgetMaxHWPrio\ |\!\} \\ \mathbf{channelset}\ csGetPrio\ == \ \{\!|\ PPSgetMinPrio, PPSgetMaxPrio\ |\!\} \\ \mathbf{channelset}\ csSCJApi\ == \ csGetHWPrio \cup csGetPrio \end{array}$$

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 inteleaved. This reflects the fact some APIs are called within the user, the RTE, or the clock interrupt handler (CIH) line of execution (see vm.ProcessLogic and vm.Process).

$$CIHApi \mathrel{\widehat=} Move \Box SCJStop$$

$$SCJRTE \mathrel{\widehat=} Start \Box AddOuterMostSeq \Box Release \Box GetCurrentProc \Box InsertReadyQueue$$

$$SCJApi \mathrel{\widehat=} GetHWPrio \Box GetPrio$$

$$Run \mathrel{\widehat=} SCJApi \interleave SCJRTE \interleave CIHApi$$

$$Execute \mathrel{\widehat=} Run \bigtriangleup Catch$$

$$Catch \mathrel{\widehat=} PCIHcatchError?e \longrightarrow \mathbf{Skip}$$

$$\bullet\ Init\ ;\ Execute$$

end

$$\mathbf{process}\ ClockInterruptHandler \mathrel{\widehat=} \mathbf{begin}$$

The vm.ClockInterruptHandler class has **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.

$$
\begin{array}{|l}
\hline
\textit{CIHState} \\
\quad current : GPID \\
\quad handler : GPID \\
\quad psi : SCHID \\
\quad disableCount : \mathbb{N} \\
\quad hvmClockReady : \mathbb{B} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{FixedCIHState} \\
\quad \Delta CIHState \\
\hline
\quad current' = current \\
\quad handler' = handler \\
\quad psi' = psi \\
\hline
\end{array}
$$

state *CIHState*

$$
\begin{array}{|l}
\hline
\textit{InitCIHSt} \\
\quad CIHState' \\
\hline
\quad handler' = current' = nullpid \\
\quad psi' = nullsch \\
\quad disableCount' = 1 \\
\quad hvmClockReady' = \mathbf{False} \\
\hline
\end{array}
$$

$$InitCIH \mathrel{\widehat=} \left(InitCIHSt\right) \; ; \; PCIHinitialise?sch?size \longrightarrow psi := sch$$

$$StartClockHandler \mathrel{\widehat=} PCIHstartClockHandlerCall?c \longrightarrow current := c \; ; \; PCIHstartClockHandlerRet \longrightarrow \mathbf{Skip}$$

$$Yield \mathrel{\widehat=} PCIHyieldCall \longrightarrow Handle \; ; \; PCIHyieldRet \longrightarrow \mathbf{Skip}$$

$$Handle \mathrel{\widehat=} PCIHhandleCall \longrightarrow Disable \; ; \; PCIHhandleRet \longrightarrow \mathbf{Skip}$$

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.

$$Register \mathrel{\widehat=} PCIHregisterCall \longrightarrow VMcihRegister \longrightarrow PCIHregisterRet \longrightarrow \mathbf{Skip}$$

there is an implicit precondition (or guard?) here about $disableCount \geq 0$. Will add as precondition

$$
\begin{array}{|l}
\hline
\textit{CIHEnable} \\
\quad FixedCIHState \\
\hline
\quad disableCount > 0 \\
\quad disableCount' = disableCount - 1 \\
\quad hvmClockReady' = (\mathbf{if}\ disableCount' = 0\ \mathbf{then}\ \mathbf{True}\ \mathbf{else}\ hvmClockReady) \\
\hline
\end{array}
$$

$$Enable \mathrel{\widehat=} \left(disableCount > 0\right) \mathbin{\&} PCIHenableCall \longrightarrow \left(CIHEnable\right) \mathbin{;} \ PCIHenableRet \longrightarrow \mathbf{Skip}$$

$$Disable \mathrel{\widehat=} \left(disableCount \geq 0\right) \mathbin{\&} PCIHdisableCall \longrightarrow disableCount, hvmClockReady := disableCount + 1, \mathbf{False}$$

$$\begin{array}{l} CIHApi \mathrel{\widehat=} Disable \mathbin{\Box} Enable \\ \qquad\qquad \Box \\ \qquad Handle \mathbin{\Box} Yield \\ \qquad\qquad \Box \\ \qquad Register \mathbin{\Box} StartClockHandler \end{array}$$

$$\begin{array}{l} Loop \mathrel{\widehat=} \mu X \bullet PPSIgetNextProcessCall \longrightarrow PPSIgetNextProcessRet?p \longrightarrow current := p; \\ \qquad Enable \mathbin{;} \ PVMtransferTo.handler.current \longrightarrow X \end{array}$$

$$Catch \mathrel{\widehat=} VMthrow?e \longrightarrow PCIHcatchError!e \longrightarrow \mathbf{Skip}$$

$$Run \mathrel{\widehat=} Loop \mathbin{\triangle} Catch$$

$$Execute \mathrel{\widehat=} Run \interleave CIHApi$$

$$\bullet \ (InitCIH \mathbin{;} \ Execute) \setminus \{\!| VMthrow, VMcihRegister |\!\}$$

end

channelset $CihPsInterface \ == \ \{\!| \ |\!\}$

process $IcecapScheduler \mathrel{\widehat=}$
$ClockInterruptHandler \llbracket CihPsInterface \rrbracket PrioScheduler \setminus csSCJRTE$

2 processes: CIH + PS: - unique identifier synch on the channel on the actual values (like mission ids, etc).

1 action per (active) used method (package/public) - call/return per active method - main method is external

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	26	81
Axiomatic definitions	14	26
Generic axiomatic defs.	0	0
Schemas	31	67
Generic schemas	0	0
Theorems	2	93
Proofs	2	2
Total	**75**	**269**

Table 5.1: Summary of Z declarations for Chapter 5.

Z Declarations	**This Chapter**	**Globally**
Unboxed items	26	81
Axiomatic definitions	14	26
Generic axiomatic defs.	0	0
Schemas	31	67
Generic schemas	0	0
Theorems	2	93
Proofs	2	2
Total	**75**	**269**
Circus **Declarations**	**This Chapter**	**Total**
Channel decls.	26	26
Channel set decls.	13	13
Process decls.	4	4
Process ref. assertions	0	0
Name sets	0	0
Actions	0	42
Action ref. assertions	0	0
Total	1	**354**

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 $LauncherL1 \mathrel{\widehat=}$ **begin**

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

DISCUSS: interface with memory areas

$$MemAlloc \mathrel{\widehat=} allocate\_backing\_store \longrightarrow allocate\_immortal\_memory \longrightarrow \mathbf{Skip}$$

$$StartMissionSeq \mathrel{\widehat=} start\_sequencer \longrightarrow done\_sequencer \longrightarrow \mathbf{Skip}$$

$$StartPrioSched \mathrel{\widehat=} start\_prio\_scheduler \longrightarrow \mathbf{Skip}$$

$\bullet$ $MemAlloc \; ; \; StartMissionSeq \; ; \; StartPrioSched$

end

6.2 Clock interrupt handler

$[Clock]$

FIX ME: These channels are initially for the clock handler interface as used within the launching mechanism of the priority scheduler L1

channel $clock\_handler\_init : Clock$
channel $clock\_handler\_register : Clock$
channel $clock\_handler\_enable : Clock$
channel $clock\_handler\_disable : Clock$
channel $clock\_handler\_start : Clock \times PID$
channel $clock\_handler\_yield : Clock$

channel $proc\_transfer : PID \times PID$
channel $getNextProcessCall$
channel $getNextProcessRet : PID$

process $ClockHandler \mathrel{\widehat=}$ **begin**

$State$

$clock : Clock$
$current : PID$
$handler : PID$
$clockready : \mathbb{B}$
$disablecnt : \mathbb{N}$

$current \neq handler$

$InitSt$

$State'$

$clockready' = \mathbf{False}$
$disablecnt' = 1$

This is not total as *disablecnt* could go negative

$EnReady$

$\Delta State$

$disablecnt' = disablecnt - 1$
$disablecnt = 0 \Rightarrow clockready' = \mathbf{True}$
$disablecnt \neq 0 \Rightarrow clockready = clockready' = \mathbf{False}$

state $State$

$Init \mathrel{\widehat=} \big(InitSt\big) \mathbin{;} \; clock\_handler\_init.clock \longrightarrow clock\_handler\_register.clock \longrightarrow \mathbf{Skip}$

$Disable \mathrel{\widehat=} clock\_handler\_disable.clock \longrightarrow disablecnt, clockready := disablecnt + 1, \mathbf{False}$

$Enable \mathrel{\widehat=} clock\_handler\_enable.clock \longrightarrow \big(EnReady\big)$

$Start \mathrel{\widehat=} clock\_handler\_start.clock?p \longrightarrow current := p$

$Yield \mathrel{\widehat=} clock\_handler\_yield.clock \longrightarrow Handle$

$Handle \mathrel{\widehat=} Disable \mathrel{;}\ proc\_transfer.current.handler \longrightarrow \mathbf{Skip}$

$$\begin{array}{l} Run \mathrel{\widehat=} \mu X \bullet getNextProcessCall \longrightarrow \\ \qquad getNextProcessRet?p \longrightarrow \\ \qquad current := p \mathrel{;}\ Enable; \\ \qquad proc\_transfer.handler.current \longrightarrow X \end{array}$$

$Execute \mathrel{\widehat=} (Disable \Box Enable \Box Start \Box Yield \Box Handle) \interleave Run$

$\bullet\ Init \mathrel{;}\ Execute$

end

6.3 Priority Scheduler (Level1)

process $Scheduler \mathrel{\widehat=}$ **begin**

$State$

$PriorityScheduler$
$clock : Clock$
$mainProcess : PID$

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.

state $State$

TODO: decide whether to have the creation as part of the state or not

$$CreateProcess \mathrel{\widehat=} \mathbf{var}\ aserr : ASchedMsg;\ vmerr : VMError \bullet \big((ACreate \land VMCreate)[idle/p!]\big); (\mathbf{if}(aserr = AS\_OKAY \land vmerr = VMOkay) \longrightarrow \mathbf{Skip} \mathrel{[\!]} (aserr \neq AS\_OKAY \lor vmerr \neq VMOkay) \longrightarrow \mathbf{Stop}\ \mathbf{fi})$$

$$CreateIdle \mathrel{\widehat=} CreateProcess\ ;\ \mathbf{Skip}$$

Clock interface to discuss

$$StartClockHandler \mathrel{\widehat=} clock\_handler\_register!clock \longrightarrow clock\_handler\_enable!clock \longrightarrow clock\_handler\_start!clock!mainProcess \longrightarrow clock\_handler\_yield!clock \longrightarrow \mathbf{Skip}$$

$$StartScheduler \mathrel{\widehat=} \mathbf{var}\ vmerr : VMError \bullet \big(VMCreate[mainProcess/p!]\big); (\mathbf{if}(vmerr = VMOkay) \longrightarrow \mathbf{Skip} \mathrel{[\!]} (vmerr \neq VMOkay) \longrightarrow \mathbf{Stop}\ \mathbf{fi})$$

$$Init \mathrel{\widehat=} clock\_handler\_init?clock \longrightarrow CreateIdle\ ;\ StartScheduler\ ;\ StartClockHandler$$

$$NextProc \mathrel{\widehat=} getNextProcessCall \longrightarrow clock\_handler\_disable!clock \longrightarrow getNextProcessRet!current \longrightarrow \mathbf{Skip}$$

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

$$Execute \mathrel{\widehat=} \mu X \bullet NextProc; (\mathbf{if}\ current = nullpid \longrightarrow proc\_transfer!current!mainProcess \longrightarrow \mathbf{Skip}\ \mathbf{fi})\ ;\ clock\_handler\_enable!clock \longrightarrow X$$

$\bullet\ Init\ ;\ Execute$

end

6.4 Mission

$[MissionId]$

$\textbf{channel } getNextMissionCall$
$\textbf{channel } getNextMissionRet : MissionId$
$\textbf{channel } start\_mission : MissionId$
$\textbf{channel } done\_mission : MissionId$
$\textbf{channel } end\_sequence\_app, end\_mission\_fw$

$nullmid : MissionId$

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 ManagedSchedulableSet in the mission sequencer?

$\textbf{process } MissionFW \mathrel{\widehat=} \textbf{begin}$

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;
ManagedSchedulableSet 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.

$State$
$HandlerSet$
$mission : MissionId$
$missionSeq : PID$
$reqTerm : \mathbb{B}$

$missionSeq \in oseh$

$InitSt$
$State'$

$reqTerm' = \textbf{False}$

state *State*

• **Skip**

end

6.5 Event Handlers

$\boxed{\text{FIX ME: Perhaps use OhCircus here given Mission sequencer extends managed event handler?}}$

6.6 Mission Sequencer

$[MissionMem]$

$nullmmem : MissionMem$

channel $req\_terminate\_mission : MissionId \times \mathbb{B}$
channel $set\_missionseq : MissionId \times PID$
channel $enter\_missionmem\_init : MissionId \times MissionMem$
channel $enter\_missionmem\_exec : MissionId \times MissionMem$
channel $enter\_missionmem\_cleanup : MissionId \times MissionMem$
channel $notify\_mission : MissionId \times PID$
channel $missionmem\_remove : MissionMem$
channel $reg\_event\_handler : PID$

process $MissionSequencerFW \mathrel{\widehat=}$ **begin**

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.

FIX ME: TODO: discuss!

State

$HandlerSet_1$
$proc : PID$
$current : MissionId$
$reqTermination : \mathbb{B}$
$missionmem : MissionMem$
$missions : MissionId \pfun HandlerSet$

$nullmid \notin \mathrm{dom}\ missions$
$current \neq nullmid \Rightarrow current \in \mathrm{dom}\ missions$

InitSt

$State'$

$reqTermination' = \mathbf{False}$
$proc' \neq nullpid$
$\exists HandlerSetInit \bullet$
$\quad missions' = \{ current' \mapsto \theta\, HandlerSet' \} \land$
$\quad \theta\, HandlerSet_1{}' = \theta\, HandlerSet'$

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

state *State*

$Start \mathrel{\widehat=} start\_sequencer \longrightarrow \mathbf{Skip}$

Mission protocol

$$StartMission \mathrel{\widehat=} (current \neq nullmid \land missionmem \neq nullmmem) \&$$
$$start\_mission.current \longrightarrow$$
$$req\_terminate\_mission.current.\mathbf{False} \longrightarrow$$
$$set\_missionseq.current.proc \longrightarrow$$
$$enter\_missionmem\_init.current.missionmem \longrightarrow$$
$$enter\_missionmem\_exec.current.missionmem \longrightarrow \mathbf{Skip}$$

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

$$MissionWait \mathrel{\widehat=} (current \neq nullmid) \; \& \; notify\_mission.current.proc \longrightarrow \mathbf{Skip}$$

$$ExecuteMission \mathrel{\widehat=} (current \neq nullmid) \&$$
$$\mu X \bullet req\_terminate\_mission.current?v \longrightarrow;$$
$$\mathbf{if}\ v = \mathbf{False} \land meh_1 \neq \varnothing \longrightarrow$$
$$MissionWait \; ; \; X$$
$$[\!]\, v = \mathbf{True} \lor meh_1 = \varnothing \longrightarrow$$
$$\mathbf{Skip}$$
$$\mathbf{fi}$$
$$done\_mission.current \longrightarrow \mathbf{Skip}$$

$$CleanupMission \mathrel{\widehat=} (current \neq nullmid) \&$$
$$enter\_missionmem\_cleanup.current.missionmem \longrightarrow \mathbf{Skip}$$

$$Execute \mathrel{\widehat=} \mu X \bullet getNextMissionCall \longrightarrow getNextMissionRet?next \longrightarrow$$
$$current := next;$$
$$\mathbf{if}\ current = nullmid \lor reqTermination = \mathbf{True} \longrightarrow$$
$$reqTermination := \mathbf{True}$$
$$[\!]\, current \neq nullmid \longrightarrow$$
$$StartMission \; ; \; ExecuteMission \; ; \; CleanupMission \; ; \; X$$
$$\mathbf{fi}$$

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

$$ReqSeqTerm \mathrel{\widehat=} (current \neq nullmid) \&$$
$$reqTermination := \mathbf{True} \; ; \; req\_terminate\_mission.current.\mathbf{True} \longrightarrow \mathbf{Skip}$$

May be called by PrioScheduler.move() when sleeping queue is empty. TODO

$$Cleanup \mathrel{\widehat=} (missionmem \neq nullmmem) \&$$
$$missionmem\_remove.missionmem \longrightarrow missionmem := nullmmem$$

$$Register \mathrel{\widehat=} reg\_event\_handler.proc \longrightarrow \mathbf{Skip}$$

$$Finish \mathrel{\widehat=} end\_sequence\_app \longrightarrow end\_mission\_fw \longrightarrow done\_sequencer \longrightarrow \mathbf{Skip}$$

$$ExecuteMissionSeq \mathrel{\widehat=} Start \; ; \; Execute \; ; \; Finish$$

$$External \mathrel{\widehat=} ReqSeqTerm \interleave Cleanup \interleave Register$$

$$\bullet \; ExecuteMissionSeq \interleave External$$

end

6.7 NEW SECTION

process $PROCNAME \mathrel{\widehat=}$ **begin**

$State$

$InitSt$

$State'$

$true$

---

state $State$

$\bullet$ **Skip**

end

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	3	84
Axiomatic definitions	2	28
Generic axiomatic defs.	0	0
Schemas	10	77
Generic schemas	0	0
Theorems	0	93
Proofs	0	2
Total	**15**	**284**

Table 6.1: Summary of Z declarations for Chapter 6.

Z Declarations	**This Chapter**	**Globally**
Unboxed items	3	84
Axiomatic definitions	2	28
Generic axiomatic defs.	0	0
Schemas	10	77
Generic schemas	0	0
Theorems	0	93
Proofs	0	2
Total	**15**	**284**
Circus **Declarations**	**This Chapter**	**Total**
Channel decls.	25	51
Channel set decls.	0	13
Process decls.	6	10
Process ref. assertions	0	0
Name sets	0	0
Actions	2	83
Action ref. assertions	0	0
Total	5	**441**

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

$$[XX]$$

$$bigU : \mathbb{P}(\mathbb{P}\ XX) \rightarrow \mathbb{P}\ XX$$

$$\langle\!\langle \text{disabled rule dBigU} \rangle\!\rangle$$
$$\forall SS : \mathbb{P}\,(\mathbb{P}\ XX) \bullet bigU\ SS = \{v : XX \mid \exists S : \mathbb{P}\ XX \mid S \in SS \bullet v \in S\}$$

A.1.2 Lemmas on sets

theorem disabled rule dlBigCupAsBigU $[XX]$
$\forall SS : \mathbb{P}\,(\mathbb{P}\ XX) \bullet \bigcup SS = bigU\ SS$

theorem disabled rule dlBigUDistCup $[XX]$
$\forall A, B : \mathbb{P}\,(\mathbb{P}\ XX) \bullet bigU\ (A \cup B) = bigU\ A \cup bigU\ B$

theorem disabled rule dlBigUBigU $[XX]$
$\forall C : \mathbb{P}\,(\mathbb{P}\ XX) \bullet bigU\ (\{bigU\ C\}) = bigU\ C$

An easy lemma to have BigU just like $\bigcup$

theorem disabled rule dlInBigU $[XX]$
$\forall SS : \mathbb{P}\,(\mathbb{P}\ XX) \bullet x \in bigU\ SS \Leftrightarrow (\exists ss : SS \bullet x \in ss)$

theorem disabled rule dlInPowerBigU $[XX]$
$\forall SS : \mathbb{P}\,(\mathbb{P}\ XX) \mid x \in SS \bullet x \in \mathbb{P}\,(bigU\ SS)$

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 $[X]$
$\forall S, T, U : \mathbb{P}\ X \bullet S \setminus (T \setminus U) = S \setminus T \cup S \cap U$

Toolkit extension A.2 (Set difference equivalence modulo set intersection)

theorem disabled rule lCapEquivWeakensDiffEquiv $[X]$
$\forall S, R, T : \mathbb{P}\ X \mid S \cap T = S \cap R \bullet S \setminus T = S \setminus R$

Similar to `computeDiff1/2`

theorem disabled rule lElemDiffAbsorbption $[X]$
$\forall x : X;\ S : \mathbb{P}\ X \mid \neg\ x \in S \bullet S \setminus \{ x \} = S$

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

theorem disabled rule lElemUnionAbsorbDiffRight $[X]$
$\forall x : X;\ S : \mathbb{P}\ X \mid x \in S \bullet \{ x \} \cup (S \setminus \{ x \}) = S$

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

theorem disabled rule lUnionAbsorbInter $[X]$
$\forall S, T : \mathbb{P}\ X \bullet S \cup T \cap S = S$

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

theorem disabled rule lUnionExchangeDiffLeft $[X]$
$\forall S, T, Q : \mathbb{P}\ X \mid Q \cap T = \{\} \bullet (S \setminus T) \cup Q = (S \cup Q) \setminus T$

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)

theorem disabled rule lFlipEquiv
$x = y \Leftrightarrow y = x$

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

theorem disabled rule lElemSetNonEmpty $[X]$
$\forall set : \mathbb{P}\ X \bullet \forall elem : set \bullet \neg\ set = \{\}$

Trivial lemmas named with a "`Backwards`" (at the end) are dual versions of original toolkit lemmas. For instance, **Z/Eves** provides the toolkit rule `cupAssociates`:

$$\forall S, T, V : \mathbb{P} X \bullet (S \cup T) \cup V = S \cup (T \cup V)$$

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

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

theorem disabled rule lCupAssociatesBackwards $[X]$
$\forall S, T, V : \mathbb{P} X \bullet S \cup (T \cup V) = (S \cup T) \cup V$

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

theorem disabled rule lDistDiffOverUnionLeftBackwards $[X]$
$\forall A, B, C : \mathbb{P}\ X \bullet (A \setminus C) \cup (B \setminus C) = (A \cup B) \setminus C$

Weakening rule A.1 (Expanding $\bigcup$ without knowing the type of x)

theorem disabled rule lBigCupElemType $[X]$
$\forall S : \mathbb{P}(\mathbb{P}\ X) \mid x \in \bigcup S \bullet x \in X$

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.
[***Circus* tools off**]

theorem rule lDiffAbsorb $[X]$
$\forall S : \mathbb{P}\ X \bullet S \setminus S = \{\}$

[***Circus* tools on**]

theorem disabled rule lInterAbsorbDiffRight $[X]$
$\forall S, T, R : \mathbb{P}\ X \mid S \cap T = \{\} \bullet S \cap (T \setminus R) = \{\}$

A.2 Relations

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

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

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

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

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

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

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

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

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

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

[***Circus* tools off**]
Leave enabled. Maybe add A, B subtypes for X, Y.

Weakening rule A.7 (Extracting dom R subtype)

theorem rule lRelDomSubType $[X, Y]$
$\forall R : X \leftrightarrow Y \bullet \mathrm{dom}\ R \in \mathbb{P}\ X$

Leave enabled

Weakening rule A.8 (Extracting dom R subtype)

theorem rule lRelRanSubType $[X, Y]$
$\forall R : X \leftrightarrow Y \bullet \mathrm{ran}\ R \in \mathbb{P}\ Y$

[***Circus* tools on**]
This one is useful when one knows fact between (x, y) and R. Needs to be used with `with disabled (ranCup) XXXX;` otherwise the prover will follow a undesired path.

Trivial lemma A.4 (ran absorbs ∪-unit to the left)

> **theorem** disabled rule lRanAbsorbsCupUnitLeft $[X, Y]$
> $\forall x : X;\ y : Y;\ R : X \leftrightarrow Y \bullet \{ y \} \cup \operatorname{ran} R = \operatorname{ran} (\{ (x, y) \} \cup R)$

Toolkit extension A.7 (Expose ↔ property)

> **theorem** disabled rule lRelWeakening $[X, Y]$
> $R \in X \leftrightarrow Y$
> $\Leftrightarrow$
> $R \in \mathbb{P}(X \times Y) \land$
> $(\forall elem : R \bullet elem \in X \times Y)$

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

> **theorem** disabled rule lRelWithinDom $[X, Y]$
> $\forall R : X \leftrightarrow Y \bullet R \in \operatorname{dom} R \leftrightarrow Y$

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

> **theorem** disabled rule lRelWithinRan $[X, Y]$
> $\forall R : X \leftrightarrow Y \bullet R \in X \leftrightarrow \operatorname{ran} R$

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

> **theorem** disabled rule lRelSelfContained $[X, Y]$
> $\forall R : X \leftrightarrow Y \bullet R \in \operatorname{dom} R \leftrightarrow \operatorname{ran} R$

Just like `domSingleton`

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

> **theorem** disabled rule lDomSingletonBackwards $[X, Y]$
> $\forall x : X;\ y : Y \bullet \{ x \} = \operatorname{dom} \{ (x, y) \}$

Just like `ranSingleton`

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

> **theorem** disabled rule lRanSingletonBackwards $[X, Y]$
> $\forall x : X;\ y : Y \bullet \{ y \} = \operatorname{ran} \{ (x, y) \}$

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

> **theorem** disabled rule lNonMaxDom $[X, Y]$
> $\forall A : \mathbb{P}\ X;\ B : \mathbb{P}\ Y \bullet \forall R : A \leftrightarrow B \bullet \operatorname{dom}[X, Y]\, R = \operatorname{dom}[A, B]\, R$

□

A.3 Function spaces

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

> **theorem** disabled rule lPfunPointIsPfunElem $[X, Y]$
> $\forall x : X;\ y : Y;\ f : X \rightarrow\!\!\!\!\!\!+\ Y \mid x \in \operatorname{dom} f \land y = f\, x \bullet (x, y) \in f$

Toolkit extension A.12 (Expose ⇸ property)

> **theorem** disabled rule lPFunWeakening $[X, Y]$
> $f \in X \rightarrow\!\!\!\!\!\!+\ Y$
> $\Leftrightarrow$
> $f \in X \leftrightarrow Y \land$
> $(\forall d : X;\ r1, r2 : Y \mid (d, r1) \in f \land (d, r2) \in f \bullet r1 = r2)$

Trivial lemma A.8 (Expose $\pfun$ property with fresh names)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lPFunWeakeningFresh } [X, Y] \\
\quad f \in X \pfun Y \\
\qquad \Leftrightarrow \\
\quad f \in X \rel Y \land \\
\quad (\forall df : X; \ rf1, rf2 : Y \mid (df, rf1) \in f \land (df, rf2) \in f \bullet rf1 = rf2)
\end{array}$$

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

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

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lEmptyDomIsEmptyPFun } [X, Y] \\
\quad \forall f : X \pfun Y \mid \lnot f = \{\} \bullet \lnot \operatorname{dom} f = \{\}
\end{array}$$

Toolkit extension A.14 (dom element is specific $\pfun$ element)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lDomElemIsElemPFun } [X, Y] \\
\quad \forall x : X; \ f : X \pfun Y \mid x \in \operatorname{dom} f \bullet (x, f\ x) \in f
\end{array}$$

Toolkit extension A.15 (Member of homogeneous $\pfun$ forms no loop)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lHomogeneousMemberNoLoop } [X] \\
\quad \forall x : X; \ f : X \pfun X \mid x \in \operatorname{dom} f \land \lnot x \in \operatorname{ran} f \bullet \lnot f\ x = x
\end{array}$$

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

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lNotImmediateMemberPFun } [X, Y] \\
\quad \forall f : X \pfun Y; \ x : X; \ y : Y \mid x \in \operatorname{dom} f \land \lnot f\ x = y \bullet \lnot (x, y) \in f
\end{array}$$

Toolkit extension A.17 ($\pfun$-$\oplus$ containment)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lPFunSubsetOplus } [X, Y] \\
\quad \forall f, g : X \pfun Y \mid g \subseteq f \bullet f \oplus g = f \oplus (\operatorname{dom} g \lhd f)
\end{array}$$

□

Toolkit extension A.18 ($\pfun$-$\oplus$ pointwise equivalence)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lPFunElemAbsorbsUnitOplusRight } [X, Y] \\
\quad \forall x : X; \ y : Y; \ f : X \pfun Y \mid (x, y) \in f \bullet f \oplus \{(x, y)\} = f
\end{array}$$

Toolkit extension A.19 ($\pfun$-dom partitions over $\lhd$)

$$\begin{array}{l}
\textbf{theorem} \text{ disabled rule lPFunDomPartitionsPFunDRes } [X, Y] \\
\quad \forall f : X \pfun Y; \ s, t : \mathbb{P}\, X \mid \langle s, t \rangle \text{ partition } \operatorname{dom} f \bullet \langle s \lhd f, t \lhd f \rangle \text{ partition } f
\end{array}$$

A.4 Injective functions

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

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

$$\begin{array}{l}
\textbf{theorem} \text{ disabled lPairInvInPInj } [X, Y] \\
\quad \forall A : \mathbb{P}\ X; \ B : \mathbb{P}\ Y \bullet \forall f : A \pinj B \mid (x, y) \in f \bullet x \in A \land y \in B \land x = (f^{\sim})\ y \land (y, x) \in (f^{\sim})
\end{array}$$

Toolkit extension A.20 (Expose $\pinj$ property)

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lPInjWeakening}\ [X, Y] \\
\quad f \in X \pinj Y \\
\qquad \Leftrightarrow \\
\quad f \in X \pfun Y \land \\
\quad (\forall r : Y;\ d1, d2 : X \mid (d1, r) \in f \land (d2, r) \in f \bullet d1 = d2)
\end{array}$$

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

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lPInjWeakeningFresh}\ [X, Y] \\
\quad f \in X \pinj Y \\
\qquad \Leftrightarrow \\
\quad f \in X \pfun Y \land \\
\quad (\forall rf : Y;\ df1, df2 : X \mid (df1, rf) \in f \land (df2, rf) \in f \bullet df1 = df2)
\end{array}$$

Lemma A.2 ($\pinj$ result uniqueness weakening rule)

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lPInjUniqueResWeakening}\ [X, Y] \\
\quad f \in X \pinj Y \\
\qquad \Leftrightarrow \\
\quad f \in X \pfun Y \land \\
\quad (\forall a, b : X \mid a \in \mathrm{dom}[X, Y] f \land b \in \mathrm{dom}[X, Y] f \land a \neq b \bullet (f\ (a), f\ (b)) \in (_ \neq _)[Y])
\end{array}$$

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

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled lPInjPairIsRanPoint}\ [X, Y] \\
\quad \forall A : \mathbb{P}\ X;\ B : \mathbb{P}\ Y \bullet \forall f : A \pinj B \mid \\
\qquad (x, y) \in f \bullet x \in A \land y \in B \land x \in \mathrm{dom}\ f \\
\qquad \land\ y \in \mathrm{ran}\ f \land y = f\ x \land x = (f\ ^{\sim})\ y
\end{array}$$

Lemma A.4 ($\pinj$ ran element $\_^{\sim}$ in $\pinj$)

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled lApplyInvInDomPInj}\ [X, Y] \\
\quad \forall A : \mathbb{P}\ X;\ B : \mathbb{P}\ Y \bullet \forall f : A \pinj B \bullet \forall a : \mathrm{ran}\ f \bullet (f\ ^{\sim})\ a \in A \land (f\ ^{\sim})\ a \in \mathrm{dom}\ f
\end{array}$$

Toolkit extension A.21 ($\pinj$-dom element ran-$\ndres$ equivalence)

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lHomogeneousElemRanNDresPInj}\ [X] \\
\quad \forall x : X;\ f : X \pinj X \mid x \in \mathrm{dom}\ f \bullet \mathrm{ran}\ (\{ x \} \ndres f) = \mathrm{ran}\ f \setminus \{ f\ x \}
\end{array}$$

Toolkit extension A.22 ($\pinj$-ran element dom-$\nrres$ equivalence)

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lHomogeneousElemDomNRresPInj}\ [X] \\
\quad \forall y : X;\ f : X \pinj X \mid y \in \mathrm{ran}\ f \bullet \mathrm{dom}\ (f \nrres \{ y \}) = \mathrm{dom} f \setminus \{ (f\ ^{\sim})\ y \}
\end{array}$$

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

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lInDomInjection}\ [X, Y] \\
\quad \forall f : X \pinj Y \bullet x \in \mathrm{dom}\ f \Leftrightarrow (\exists y : \mathrm{ran}\ f \bullet x = (f\ ^{\sim})\ y)
\end{array}$$

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

$$\begin{array}{l}
\textbf{theorem}\ \text{disabled rule lInRanInjection}\ [X, Y] \\
\quad \forall f : X \pinj Y \bullet y \in \mathrm{ran}\ f \Leftrightarrow (\exists x : X \mid (x, y) \in f \bullet x = (f\ ^{\sim})\ y)
\end{array}$$

This lemmas is similar (yet complementary) to **Z/Eves** toolkit rule `applyInverse`.

Toolkit extension A.25 ($\pinj$-$\_^{\sim}$ **application)**

theorem disabled rule lApplyInverse2 $[X, Y]$
$\forall A : \mathbb{P}\ X;\ B : \mathbb{P}\ Y \bullet \forall f : A \pinj B \bullet \forall y : \mathrm{ran}\ f \bullet f((f^{\sim})\ y) = y$

Toolkit extension A.26 ($\pinj$ **point is** $\pinj$ **element)**

theorem disabled rule lPInjInvPointIsPInjElem $[X, Y]$
$\forall f : X \pinj Y \mid y \in \mathrm{ran}\ f \land (f^{\sim})\ y = x \bullet (x, y) \in f$

Toolkit extension A.27 (Distinct $\pinj$ **point is not shared in** $\pinj$**)**

theorem disabled rule lPInjPointIsNotShared $[X, Y]$
$\forall x1, x2 : X;\ f : X \pinj Y \mid x1 \in \mathrm{dom}\ f \land \lnot\ x1 = x2 \bullet \lnot\ (x2, f\ x1) \in f$

Toolkit extension A.28 ($\pinj$ **following application is not** $\pinj$ **member)**

theorem disabled rule lFollowingApplicationNoLoopHomogeneousPInj $[X]$
$\forall x : X;\ f : X \pinj X \mid x \in \mathrm{dom}\ f \land f\ x \in \mathrm{dom}\ f \land \lnot\ f\ x = x \bullet \lnot\ (f\ x, f\ x) \in f$

A.5 Finiteness

[***Circus tools off***]
Leave enabled!

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

theorem rule lCrossFinite2
$A \times B \in \mathbb{F}\,(C \times D) \Leftrightarrow$
$\quad A = \{\,\} \lor B = \{\,\} \lor (A \in \mathbb{F}\ C \land B \in \mathbb{F}\ D)$

$\square$

[***Circus tools on***]

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

theorem disabled lFinsetSubset
$X \in \mathbb{P}\ Y \land Y \in \mathbb{F}\ Z \Rightarrow X \in \mathbb{F}\ Z$

$\square$

[***Circus tools off***]
Leave enabled!

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

theorem rule lIsFinite
$x \in \mathbb{F}\ X \Rightarrow x \in \mathbb{P}\ X$

$\square$

[***Circus tools on***]

Toolkit extension A.29 (Finite bijection subset measurement)

theorem disabled lBijectionFinite $[X, Y]$
$\forall A : \mathbb{F}\ X;\ B : \mathbb{P}\ Y \bullet \forall f : A \rightarrowtail B \bullet f \in A \twoheadrightarrow B \land B \in \mathbb{F}\ Y \land \#\ A = \#\ B = \#\ f$

$\square$

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

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

$\square$

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

theorem disabled rule lNonMaxDomEquiv $[X, Y]$
$\forall A : \mathbb{P}\ X;\ B : \mathbb{P}\ Y \bullet \forall R : A \leftrightarrow B \bullet \mathrm{dom}[X, Y]\ R = \mathrm{dom}[A, B]\ R$

$\square$

A.6 Sequeces

A.6.1 Lemmas on sequences

theorem disabled rule lDisjointCatMinusUnion $[X]$
$\forall S, T, R : \mathbb{P}\ X \mid \text{disjoint}\langle S\rangle \frown \langle R\rangle \bullet \text{disjoint}\langle S \setminus T\rangle \frown \langle (R \cup T)\rangle$

theorem disabled rule lDisjointCatUnionMinus $[X]$
$\forall S, T, R : \mathbb{P}\ X \mid \text{disjoint}\langle S\rangle \frown \langle R\rangle \bullet \text{disjoint}\langle S \cup T\rangle \frown \langle (R \setminus T)\rangle$

theorem disabled rule lDisjointCatMinus $[X]$
$\forall S, T, R : \mathbb{P}\ X \mid \text{disjoint}\langle S\rangle \frown \langle R\rangle \bullet \text{disjoint}\langle S \setminus T\rangle \frown \langle R\rangle$

Toolkit extension A.32 (Expose seq property)

theorem disabled rule lSeqWeakening $[X]$
$s \in \text{seq}\ X$
$\Leftrightarrow$
$s \in \mathbb{N} \pfun X \land$
$(\exists n : \mathbb{N} \bullet \text{dom}[\mathbb{Z}, X]\ s = 1 \mathinner{..} n)$

□

Toolkit extension A.33 (Expose iseq property)

theorem disabled rule lISeqWeakening $[X]$
$s \in \text{iseq}\ X$
$\Leftrightarrow$
$s \in \text{seq}\ X \land s \in \mathbb{N} \pinj X$

□

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

theorem disabled rule lSeqElemType $[X]$
$\forall A : \mathbb{P}\ X \bullet \forall s : \text{seq}\ A \mid e \in s \bullet e \in \mathbb{N}_1 \times A$

□

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

theorem disabled rule lSeqWithElemIsNonEmpty $[X]$
$\forall s : \text{seq}\ X \mid e \in s \bullet \neg\ s = \langle\rangle$

□

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

theorem disabled lNonEmptySeqHasElem $[X]$
$\forall A : \mathbb{P}\ X \bullet \forall s : \text{seq}_1\ A \bullet \exists e : \mathbb{N}_1 \times A \bullet e \in s$

□

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

theorem disabled rule lSeqRanElemType $[X]$
$\forall A : \mathbb{P}\,X \bullet \forall s : \text{seq}\,A \mid (i, y) \in s \bullet y \in A$

Toolkit extension A.34 (Non-empty sequence has strictly positive size)

theorem disabled rule lSeqNonEmptySize $[X]$
$\forall s : \operatorname{seq} X \mid \neg s = \langle\rangle \bullet 1 \leq \# s$

□

Toolkit extension A.35 (Alternative for dom s when involving contraction)

theorem disabled rule lSeqDomEquiv $[X]$
$\forall s : \operatorname{seq} X \bullet \operatorname{dom} s \setminus \{\# s\} = 1 \mathrel{..} -1 + \# s$

□

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

theorem disabled rule lNonMaxDomSeq $[X]$
$\forall A : \mathbb{P}\, X \bullet \forall s : \operatorname{seq} A \bullet \operatorname{dom}[\mathbb{Z}, A]\, s = 1 \mathrel{..} (\# \_)[(\mathbb{Z} \times A)]\, s$

□

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

theorem disabled rule lNonMaxDomSeqEquiv $[X]$
$\forall A : \mathbb{P}\, X \bullet \forall s : \operatorname{seq} A \bullet \operatorname{dom}[\mathbb{Z}, X]\, s = \operatorname{dom}[\mathbb{Z}, A]\, s$

□

Toolkit extension A.38 (Sequence domain non-maximal card. equivalence)

theorem disabled rule lNonMaxDomSeqNatEquiv $[X]$
$\forall A : \mathbb{P}\, X \bullet \forall s : \operatorname{seq} A \bullet \operatorname{dom}[\mathbb{Z}, X]\, s = \operatorname{dom}[\mathbb{N}, A]\, s$

□

A.6.2 Lemmas on Sequence concatenation

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

theorem disabled rule lSeqCatLastLeftApply $[X]$
$\forall s, t : \operatorname{seq} X \mid \neg s = \langle\rangle \land \neg t = \langle\rangle \bullet (s \frown t)(\# s) = \mathit{last}\, s$

□

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

theorem disabled rule lSeqCatHeadRightApply $[X]$
$\forall s, t : \operatorname{seq} X \mid \neg s = \langle\rangle \land \neg t = \langle\rangle \bullet (s \frown t)(1 + \# s) = \mathit{head}\, t$

□

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*-(_ ⁀ _) **left side equivalence)**

theorem disabled rule lCatLeftLastEquiv $[X]$
$\forall r, s, t : \mathrm{seq}\ X \mid r = s \frown t \land \neg\, s = \langle\rangle \bullet last\ s = r\,(\#\ s)$

□

Toolkit extension A.44 (*head*-(_ ⁀ _) **right side equivalence)**

theorem disabled rule lCatRightHeadEquiv $[X]$
$\forall r, s, t : \mathrm{seq}\ X \mid r = s \frown t \land \neg\, t = \langle\rangle \bullet head\ t = r\,(1 + \#\ s)$

□

Toolkit extension A.45 (Concatenation size equivalence)

theorem disabled rule lSizeCatEquals $[X]$
$\forall s, t : \mathrm{seq}\ X \mid r = s \frown t \bullet \#\ s + \#\ t = \#\ r$

□

Toolkit extension A.46 (*dom* **of sequence concatenation)**

theorem disabled rule lDomCat $[X]$
$\forall s, t : \mathrm{seq}\, X \bullet \mathrm{dom}(s \frown t) = 1\,..\,\#\,s + \#\,t$

□

[***Circus tools off***]

Weakening rule A.12 (⁀ **max type)**

theorem rule lSeqInCatMaxType $[X]$
$\forall s : \mathrm{seq}\ X;\ x : X \bullet s \frown \langle x\rangle \in \mathbb{P}\,(\mathbb{Z} \times X)$

[***Circus tools on***]
[***Circus tools off***]

Weakening rule A.13 (⁀ **injective sequence to right)**

theorem rule lInISeqCatRight $[X]$
$\forall A : \mathbb{P}\ X \bullet \forall s : \mathrm{iseq}\ A;\ a : A \mid \neg\, a \in \mathrm{ran}\ s \bullet s \frown \langle a\rangle \in \mathrm{iseq}\ A$

[***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

$$(i, a) \in tail\ s \Rightarrow (1 + i, a) \in s$$

are trivial and encouraged (TODO: add later)

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

theorem disabled rule lTailElemInDom $[X]$
$\forall Y : \mathbb{P}\ X \bullet \forall s : \mathrm{seq}\ Y \mid \neg\, s = \langle\rangle \land (i, a) \in tail\ s \bullet 1 + i \in \mathrm{dom}\ s$

□

[***Circus tools off***]
Leave enabled!

Weakening rule A.14 (*tail* preserves sequence injectiveness)

$$\textbf{theorem}\ \text{rule lTailISeqIsISeq}\ [X]$$
$$\forall\, Y : \mathbb{P}\ X \bullet \forall\, s : \text{iseq}\ Y \mid \neg\ s = \langle\rangle \bullet \mathit{tail}\ s \in \text{iseq}\ Y$$

□

[***Circus* tools on**]

Toolkit extension A.48 (Domain of non-empty sequence's *tail*)

$$\textbf{theorem}\ \text{disabled rule lTailDom}\ [X]$$
$$\forall\, s : \text{seq}\, X \mid \neg\ s = \langle\rangle \bullet \text{dom}\,(\mathit{tail}\ s) = 1 \mathinner{..} -1 + \#\ s$$

□

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

Toolkit extension A.49 (Sequence *tail* element in range of *tail*)

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

□

Toolkit extension A.50 (Sequence *tail* is closed under seq range)

$$\textbf{theorem}\ \text{disabled rule lTailRanSubset}\ [X]$$
$$\forall\, s : \text{seq}\, X \mid \neg\ s = \langle\rangle \land y \in \text{ran}\,(\mathit{tail}\ s) \bullet y \in \text{ran}\ s$$

□

Toolkit extension A.51 (Non singleton sequence cardinality)

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

□

Non-singleton sequence property

$$\mathit{NonSingletonSeqInducProperty}[X]$$
$$A : \mathbb{P}\, X$$
$$\text{seq}\, A \subseteq \{\, r : \text{seq}\ A \mid \#\ r > 1 \Rightarrow (\exists\, i, j : \text{seq}_1\ A \bullet r = i \frown j)\,\}$$

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

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

□

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

$$\textbf{theorem}\ \text{disabled lNonSingletonSeqMidPoint}\ [X]$$
$$\forall\, A : \mathbb{P}\ X \bullet \forall\, s : \text{seq}\ A \mid \#\ s > 1 \bullet \exists\, a, b : \text{seq}_1\ A \bullet s = a \frown b$$

□

[***Circus* tools off**]

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

$$\textbf{theorem}\ \text{lSeqRightInducDecompZEvesInduc}\ [X]$$
$$\text{seq}_1\ X \subseteq \{\, t : \text{seq}\ X;\ x : X \bullet t \frown \langle x \rangle \,\}$$

[***Circus* tools on**]

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

$$\textbf{theorem}\ \text{disabled rule lSeqRightInducDecomp}\ [X]$$
$$s \in \text{seq}_1\ X \Leftrightarrow (\exists\, t : \text{seq}\ X;\ x : X \bullet s = t \frown \langle x \rangle)$$

A.6.4 Sequence manipulation

[***Circus tools off***]
Leave enabled

Toolkit extension A.55 (Range of sequence filtering)

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

$\square$

[***Circus tools on***]

A.6.5 Sequence mapping

[***Circus tools off***]
Toolkit extension A.56 (Sequence mapping remains sequence when within the map)

$$\textbf{theorem}\ \text{rule lSeqMapIsSeq}\ [X, Y]$$
$$\forall s : \operatorname{seq} X;\ f : X \twoheadrightarrow Y \mid \operatorname{ran} s \in \mathbb{P}(\operatorname{dom} f) \bullet s \mathbin{;} f \in \operatorname{seq} Y$$

$\square$

[***Circus tools on***]

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	0	84
Axiomatic definitions	0	28
Generic axiomatic defs.	2	2
Schemas	0	77
Generic schemas	1	1
Theorems	99	192
Proofs	0	2
Total	**102**	**386**

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_vc_fsb_axiom*]
instantiate maxpid == 1;
prove;
■

proof[*basic_defs_axiom2_vc_fsb_axiom*]
invoke PID;
instantiate nullpid == *maxpid* + 1;
prove;
use lMaxPIDPositive;
simplify;
■

proof[*PID_vc_fsb_horiz_def*]
instantiate PID == {};
prove;
■

proof[*GPID_vc_fsb_horiz_def*]
instantiate GPID == {};
prove;
■

proof[*lMaxPIDPositive*]
use maxpid$declaration;
apply inNat1;
simplify;
■

proof[*lPIDNotEmpty*]
apply extensionality;
invoke PID;
prove by rewrite;
instantiate $x == 1$;
rewrite;
■

proof[*lMinPIDValue*]
with enabled (*PID*) *prove by reduce*;
■

proof[*gMaxpidMaxType*]
rewrite;
■

proof[*lMaxpidIsPID*]
with enabled (*PID*) *prove by reduce*;
■

proof[*lNullpidBound*]
use dNullPID[p := *maxpid*];
invoke PID;
rewrite;
■

proof[*lNullPIDDisjoint*]
use dNullPID;
rearrange;
rewrite;
■

proof[*gPIDMaxType*]
invoke PID;
rewrite;
■

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

proof[*lNullIsGPID*]
invoke GPID;
rewrite;
■

proof[$lNullIsNotPID$]
$use\ lNullpidBound;$
$invoke\ PID;$
$apply\ inRange;$
$simplify;$
■

proof[$lPIDIsGPID$]
$with\ enabled\ (GPID)\ prove\ by\ reduce;$
■

proof[$lMinPIDIsGPID$]
$rewrite;$
■

proof[$lMaxPIDIsGPID$]
$rewrite;$
■

proof[$lNonNullGPIDIsPID$]
$invoke\ GPID;$
$prove\ by\ rewrite;$
■

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[$VMCreateFSB$]
$split\ freeVM = \emptyset;$
$prove\ by\ reduce;$
$cases;$
$instantiate\ p! == nullpid;$
$rewrite;$
$next;$
$apply\ extensionality\ to\ predicate\ freeVM = \{\};$
$prenex;$
$instantiate\ p! == x;$
$with\ enabled\ (disjointCat,\ distributeCapOverCupLeft,$
$distributeCapOverCupRight,\ lElemUnionAbsorbDiffRight,\ lInterAbsorbDiffRight)\ prove;$
$apply\ extensionality;$
$instantiate\ x\_0 == x;$
$prove;$
$next;$
■

proof[$VMRunFSB$]
split $p? \in createdVM$;
with enabled ($disjointCat$, $distributeCapOverCupLeft$, $distributeCapOverCupRight$, $lElemUnionAbsorbDiffRight$, $lInterAbsorbDiffRight$) *prove by reduce*;
apply extensionality;
instantiate $x\_\_0 == p?$;
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[$lScheduledPIDDisjoint$]
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;
rearrange;
rewrite;
next;
split $blockedAS \cup (freeAS \cup (readyAS \cup \{current\})) = PID$;
simplify;
equality substitute PID;
rearrange;
rewrite;
rearrange;
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} ∪ (freeAS \ {x});
        with enabled (disjointCat) rewrite;
        apply extensionality to predicate blockedAS ∩ freeAS = {};
        apply extensionality to predicate blockedAS ∩ (freeAS \ {x}) = {};
        prenex;
        rewrite;
        instantiate x__1 == x__0;
        rewrite;
        apply distributeCapOverCupRight to expression (readyAS ∪ {x}) ∩ (blockedAS ∪ (freeAS \ {x}))
        apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
        rewrite;
        apply distributeCapOverCupRight to expression blockedAS ∩ (readyAS ∪ {x});
        rewrite;
        apply distributeCapOverCupLeft to expression (readyAS ∪ {x}) ∩ (freeAS \ {x});
        rewrite;
        apply extensionality to predicate freeAS ∩ readyAS = {};
        apply extensionality to predicate readyAS ∩ (freeAS \ {x}) = {};
        prenex;
        rewrite;
        split x ∈ 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} ∪ (freeAS \ {x});
        rewrite;
        apply distributeCapOverCupRight to expression (readyAS ∪ {x}) ∩ (blockedAS ∪ (freeAS \ {x}))
        apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
        rewrite;
        apply extensionality to predicate blockedAS ∩ (freeAS \ {x}) = {};
        apply extensionality to predicate blockedAS ∩ freeAS = {};
        prenex;
        rewrite;
        instantiate x__1 == x__0;
        rearrange;
        rewrite;
        apply distributeCapOverCupRight to expression blockedAS ∩ (readyAS ∪ {x});
        rewrite;
        apply distributeCapOverCupLeft to expression (readyAS ∪ {x}) ∩ (freeAS \ {x});
        rewrite;
        split x ∈ blockedAS;
        rewrite;
        cases;
            instantiate x__1 == x;
            prove by rewrite;
        next;
            apply extensionality to predicate freeAS ∩ readyAS = {};
            apply extensionality to predicate readyAS ∩ (freeAS \ {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 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*;
apply distributeCapOverCupRight;
rewrite;
apply distributeCapOverCupRight to expression $readyAS \cap (freeAS \cup \{current\})$;
rewrite;

■

proof$[ATimeOut0FSB]$

with disabled (PID) *with enabled* $(disjointCat)$ *prove by reduce*;
apply 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 $\{p?\} \cup (blockedAS \setminus \{p?\})$;
rewrite;
with normalization rewrite;
cases;
with enabled $(disjointCat)$ *prove by rewrite*;
apply distributeCapOverCupRight to expression $(readyAS \cup \{p?\}) \cap (freeAS \cup (blockedAS \setminus \{p?$
apply distributeCapOverCupRight to expression $readyAS \cap (blockedAS \cup freeAS)$;
rewrite;
cases;
apply extensionality to predicate $freeAS \cap (blockedAS \setminus \{p?\}) = \{\}$;
apply extensionality to predicate $blockedAS \cap freeAS = \{\}$;
prove by rewrite;
instantiate $x\_\_0 == x$;
rewrite;
next;
apply distributeCapOverCupRight to expression $freeAS \cap (readyAS \cup \{p?\})$;
apply extensionality to predicate $blockedAS \cap freeAS = \{\}$;
prove by rewrite;
instantiate $x == p?$;
prove by rewrite;
next;
apply distributeCapOverCupLeft to expression $(readyAS \cup \{p?\}) \cap (blockedAS \setminus \{p?\})$;
rewrite;
apply extensionality to predicate $readyAS \cap (blockedAS \setminus \{p?\}) = \{\}$;
apply extensionality to predicate $blockedAS \cap readyAS = \{\}$;
prove by rewrite;
instantiate $x\_\_0 == x$;
prove by rewrite;
next;
cases;
apply extensionality to predicate $freeAS \cap (blockedAS \setminus \{p?\}) = \{\}$;
apply extensionality to predicate $blockedAS \cap freeAS = \{\}$;
prove by rewrite;
instantiate $x\_\_0 == x$;
prove by rewrite;
next;
apply distributeCapOverCupLeft to expression $(readyAS \cup \{p?\}) \cap (freeAS \cup (blockedAS \setminus \{p$
rewrite;
cases;
apply extensionality to predicate $blockedAS \cap freeAS = \{\}$;
instantiate $x == p?$;
prove by rewrite;
next;
apply extensionality to predicate $readyAS \cap (freeAS \cup (blockedAS \setminus \{p?\})) = \{\}$;
apply extensionality to predicate $readyAS \cap (blockedAS \cup freeAS) = \{\}$;
prove by rewrite;
instantiate $x\_\_0 == x$;
prove by rewrite;
next;

$\blacksquare$

proof[$ADestroyCurrentFSB$]

$with\ disabled\ (PID)\ with\ enabled\ (disjointCat)\ prove\ by\ reduce;$

$cases;$

$apply\ distributeCapOverCupRight;$

$rewrite;$

$next;$

$apply\ distributeCapOverCupRight;$

$rewrite;$

$apply\ extensionality\ to\ predicate\ readyAS\ \cap\ (freeAS\ \cup\ \{p?\})\ =\ \{\};$

$apply\ extensionality\ to\ predicate\ freeAS\ \cap\ readyAS\ =\ \{\};$

$prove\ by\ rewrite;$

$instantiate\ x\_\_0\ ==\ x;$

$prove\ by\ rewrite;$

$next;$

∎

proof[*ADestroyReadyFSB*]

```
with disabled (PID) with enabled (disjointCat) prove by reduce;
apply lCupAssociatesBackwards to expression freeAS ∪ ({p?} ∪ (readyAS \ {p?}));
apply distributeCapOverCupLeft to expression (blockedAS ∪ (freeAS ∪ {p?})) ∩ (readyAS \ {p?});
rewrite;
apply distributeCapOverCupLeft to expression (freeAS ∪ {p?}) ∩ (readyAS \ {p?});
rewrite;
apply lElemUnionAbsorbDiffRight to expression {p?} ∪ (readyAS \ {p?});
rewrite;
with normalization rewrite;
cases;
  with enabled (disjointCat) prove by rewrite;
  cases;
    apply distributeCapOverCupRight to expression blockedAS ∩ (freeAS ∪ {p?});
    apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
    rewrite;
    apply extensionality to predicate blockedAS ∩ readyAS = {};
    instantiate x == p?;
    rewrite;
  next;
    apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
    rewrite;
    apply extensionality to predicate blockedAS ∩ readyAS = {};
    apply extensionality to predicate blockedAS ∩ (readyAS \ {p?}) = {};
    prenex;
    rewrite;
    instantiate x__0 == x;
    rewrite;
  next;
    apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
    rewrite;
    apply extensionality to predicate freeAS ∩ (readyAS \ {p?}) = {};
    apply extensionality to predicate freeAS ∩ readyAS = {};
    prenex;
    rewrite;
    instantiate x__0 == x;
    rewrite;
next;
      cases;
    apply distributeCapOverCupRight to expression blockedAS ∩ (freeAS ∪ {p?});
    apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
    rewrite;
    apply extensionality to predicate blockedAS ∩ readyAS = {};
    instantiate x == p?;
    rewrite;
  next;
    apply extensionality to predicate readyAS ∩ (blockedAS ∪ freeAS) = {};
    apply extensionality to predicate blockedAS ∩ (readyAS \ {p?}) = {};
    prenex;
    instantiate x__0 == x;
    rewrite;
  next;
    apply distributeCapOverCupRight to expression readyAS ∩ (blockedAS ∪ freeAS);
    rewrite;
    apply extensionality to predicate freeAS ∩ (readyAS \ {p?}) = {};
    apply extensionality to predicate freeAS ∩ readyAS = {};
    prenex;
    instantiate x__0 == x;
    rewrite;
next;
```

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

proof[$ADispatchErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$ATimeOutErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$ABlockErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$AWakeUpErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$ACreateErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$ADestroyErrFSB$]
with disabled $(AScheduler, PID)$ *prove by reduce*;
■

proof[$ADispatchIsTotal$]

use $ADispatch0FSB$;
use $ADispatchErrFSB$;
with disabled ($AScheduler$, PID) *prove by reduce*;
split $current = nullpid \Rightarrow readyAS = \{\}$;
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 $p? \in blockedAS$;
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[$ADestroyIsTotal$]
 $invoke\ ADestroy;$
 $invoke\ ADestroyErr;$
 $split\ p? \in freeAS;$
 $cases;$
  $use\ ADestroyErrFSB;$
  $with\ disabled\ (AScheduler,\ ADestroy0,\ PID)\ prove\ by\ reduce;$
  $instantiate\ aserr\_\_0! == aserr!,\ blockedAS\_\_0' == blockedAS',\ current\_\_0' == current',$
  $freeAS\_\_0' == freeAS',\ readyAS\_\_0' == readyAS';$
  $rewrite;$
 $next;$
  $invoke\ ASchedOkay;$
  $rewrite;$
  $invoke\ ADestroy0;$
  $split\ p? = current;$
  $cases;$
   $use\ ADestroyCurrentFSB;$
   $with\ disabled\ (ADestroyReady,\ ADestroyBlocked,\ AScheduler,\ PID)\ prove\ by\ reduce;$
   $instantiate\ blockedAS\_\_0' == blockedAS',\ current\_\_0' == current',$
   $freeAS\_\_0' == freeAS',\ readyAS\_\_0' == readyAS';$
   $rewrite;$
  $next;$
   $split\ p? \in readyAS;$
   $cases;$
    $use\ ADestroyReadyFSB;$
    $with\ disabled\ (ADestroyCurrent,\ ADestroyBlocked,\ AScheduler,\ PID)\ prove\ by\ reduce;$
    $instantiate\ blockedAS\_\_0' == blockedAS',\ current\_\_0' == current',$
    $freeAS\_\_0' == freeAS',\ readyAS\_\_0' == readyAS';$
    $rewrite;$
   $next;$
    $split\ p? \in blockedAS;$
    $cases;$
     $use\ ADestroyBlockedFSB;$
     $with\ disabled\ (ADestroyReady,\ ADestroyCurrent,\ AScheduler,\ PID)\ prove\ by\ reduce;$
     $instantiate\ blockedAS\_\_0' == blockedAS',\ current\_\_0' == current',$
     $freeAS\_\_0' == freeAS',\ readyAS\_\_0' == readyAS';$
     $rewrite;$
    $next;$
       $use\ lScheduledPIDDisjoint;$
     $invoke\ ASchedTotalSig;$
     $prove\ by\ rewrite;$
 $next;$
■

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[$\mathit{HandlerSetInitFSB}$]

$\mathit{instantiate}\ \mathit{meh}' == \{ \mathit{IDLE\_PID\_WITNESS} \},\ \mathit{peh}' == \{ \mathit{IDLE\_PID\_WITNESS} \},$
$\mathit{aeh}' == \varnothing,\ \mathit{oseh}' == \varnothing,\ \mathit{freeHS}' == \mathit{PID} \setminus \{ \mathit{IDLE\_PID\_WITNESS} \},\ \mathit{idle}' == \mathit{IDLE\_PID\_WITNE}$
$\mathit{with}\ \mathit{enabled}\ (\mathit{disjointCat},\ \mathit{HandlerSet}0,\ \mathit{lElemUnionAbsorbDiffRight})\ \mathit{prove}\ \mathit{by}\ \mathit{reduce};$
$\blacksquare$

proof[$\mathit{AddHandlerFSB}$]

$\mathit{prove}\ \mathit{by}\ \mathit{reduce};$
$\mathit{split}\ p? \in \mathit{meh};$
$\mathit{rewrite};$
$\mathit{cases};$
$\mathit{instantiate}\ \mathit{meh}' == \mathit{meh},\ \mathit{peh}' == \mathit{peh},\ \mathit{aeh}' == \mathit{aeh},\ \mathit{oseh}' == \mathit{oseh},$
$\mathit{freeHS}' == \mathit{freeHS},\ \mathit{idle}' == \mathit{idle};$
$\mathit{prove};$
$\mathit{next};$
$\mathit{apply}\ \mathit{lElemUnionAbsorbDiffRight};$
$\mathit{apply}\ \mathit{lDisjointCatUnionMinus};$
$\mathit{with}\ \mathit{enabled}\ (\mathit{disjointCat})\ \mathit{prove};$
$\mathit{invoke}\ \mathit{HandlerSet}0;$
$\mathit{instantiate}\ \mathit{aeh}' == \mathit{aeh},\ \mathit{oseh}' == \mathit{oseh},\ \mathit{peh}' == \mathit{peh} \cup \{ p? \},\ \mathit{idle}' == \mathit{idle};$
$\mathit{with}\ \mathit{enabled}\ (\mathit{disjointCat})\ \mathit{prove};$
$\mathit{apply}\ \mathit{distributeCapOverCupLeft}\ ;$
$\mathit{apply}\ \mathit{distributeCapOverCupRight};$
$\mathit{rewrite};$
$\mathit{apply}\ \mathit{extensionality}\ \mathit{to}\ \mathit{predicate}\ \mathit{aeh} \cup (\mathit{freeHS} \cup (\mathit{oseh} \cup \mathit{peh})) = \mathit{PID};$
$\mathit{instantiate}\ y == p?;$
$\mathit{rewrite};$
$\mathit{next};$
$\blacksquare$

proof[$RemoveHandlerFSB$]
prove by reduce;
split $p? \in freeHS$;
rewrite;
cases;
instantiate $meh' == meh$, $peh' == peh$, $aeh' == aeh$, $oseh' == oseh$,
$freeHS' == freeHS$, $idle' == idle$;
prove;
next;
split $\neg\ p? = idle$;
rewrite;
apply $lElemUnionAbsorbDiffRight$;
apply $lDisjointCatMinusUnion$;
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;
∎

proof[$RemoveAperiodicHandlersFSB$]
prove by reduce;
instantiate $aeh' == \emptyset$, $oseh' == oseh$, $peh' == peh$, $freeHS' == freeHS \cup aeh$, $idle' == idle$;
with enabled ($lDisjointCatMinusUnion$) *with disabled* ($cupCommutes$) *rewrite*; *invoke* $HandlerSet0$;
with enabled ($disjointCat$, $distributeCapOverCupRight$, $distributeCapOverCupLeft$) *prove*;
apply $distributeDiffOverCupLeft$;
rewrite;
apply extensionality to predicate $oseh \cup peh = oseh \cup peh \setminus aeh$;
with normalization prove;
cases;
apply extensionality to predicate $aeh \cap oseh = \{\}$;
instantiate $x\_0 == x$;
prove;
next;
apply extensionality to predicate $aeh \cap peh = \{\}$;
instantiate $x\_0 == x$;
prove;
next;
■

B.7 Z Section *Priority scheduler proofs*

Z Proof Section B.7.1 (*Priority scheduler proofs* (see 4.2, p. 17))

proof[$MIN\_PRIOgeq1$]
use $MIN\_PRIO\$declaration$;
apply $inNat1$;
rewrite;
■

proof[$MAX\_PRIOgeq1$]
use $MAX\_PRIO\$declaration$;
apply $inNat1$;
rewrite;
■

proof[$PRIORITYMaxType$]
invoke $PRIORITY$;
rewrite;
■

proof[$PRIORITYType$]
invoke $PRIORITY$;
rewrite;
■

proof[$lPriorityNonEmpty$]
apply extensionality;
invoke $PRIORITY$;
instantiate $x == MIN\_PRIO$;
rewrite;
■

proof[$PRIO\_WITNESSMaxType$]
invoke $PRIO\_WITNESS$;
apply *overrideInPowerCross*;
prove;
■

proof[$PRIO\_WITNESSRelType$]
invoke $PRIO\_WITNESS$;
apply *overrideInRel*;
prove;
■

proof[$PRIO\_WITNESSPfunType$]
invoke $PRIO\_WITNESS$;
apply *overrideInPfun*;
prove;
■

proof[$PRIO\_WITNESSElement$]
invoke $PRIO\_WITNESS$;
with enabled (*oplusDef*, *unitSeqDef*) *prove*;
■

proof[$PRIO\_WITNESSdom$]
invoke $PRIO\_WITNESS$;
invoke $PRIORITY$;
with enabled (*cupSubsetLeft*, *dMinPrio*) *prove*;
■

proof[$PRIO\_WITNESSType$]
apply $inSeq1$;
invoke (seq _);
prove;
cases;
instantiate $n == MAX\_PRIO$;
use $lNonMaxDom[\mathbb{Z}, \mathbb{P}(\mathbb{Z} \times \mathbb{Z})][A := \mathbb{Z}, B := \mathrm{iseq}\ PID, R := PRIO\_WITNESS]$;
prove;
invoke $PRIORITY$;
with enabled (*dMinPrio*) *prove*;
next;
apply *extensionality*;
instantiate $x == (MIN\_PRIO, \langle\ IDLE\_PID\_WITNESS\ \rangle)$;
with enabled (*nullSeqDef*, *dMinPrio*) *prove*;
next;
■

proof[$PRIO\_WITNESS\_IDLE\_WITNESSEquiv$]
apply extensionality;
with enabled ($dlBigCupAsBigU$, $dlInBigU$) *prove*;
cases;
with enabled ($inRanFunction$) *prove*;
apply applyComp;
prove;
split $x\_\_0 \in PRIORITY \land PRIO\_WITNESS\, x\_\_0 \in \mathbb{Z} \leftrightarrow \mathbb{Z}$;
rewrite;
cases;
apply domComp;
rewrite;
invoke $PRIO\_WITNESS$;
apply applyOverride;
prove;
next;
apply domComp;
rewrite;
next;
instantiate $ss\_\_0 == \{y\}$;
with enabled ($inRanFunction$) *prove*;
instantiate $x == MIN\_PRIO$;
apply applyComp;
prove;
apply domComp;
rewrite;
invoke $PRIO\_WITNESS$;
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 ($HandlerSet0$, $disjointCat$, $lElemUnionAbsorbDiffRight$) *prove by reduce*;
with enabled ($PRIO\_WITNESS$) *reduce*;
■

proof[$lCollectPIDSIsTotal$]
use $collectPIDS\$declaration$;
prove;
■

proof[$lFlattenPIDSIsTotal$]
use $flattenPIDS\$declaration$;
prove;
■

proof[$fPRIOResMaxRelType$]
$use\ applyInRanPfun[\mathbb{Z}, \mathbb{P}(\mathbb{Z} \times \mathbb{Z})][A := PRIORITY, B := \mathrm{seq}\,\mathbb{Z}, f := prio, a := pr]$;
$prove$;
■

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

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

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

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

Now to the lemmas...

proof[$lVMAllAlocated$]
$invoke\ VM$;
$with\ enabled\ (disjointCat)\ rewrite$;
$equality\ substitute\ PID$;
$prove$;
■

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

proof[$fVMInvariant$]
$prove\ by\ reduce$;
■

proof[$fPSFreeInvariant$]
$invoke\ PriorityScheduler$;
$simplify$;
■

proof[$fASInvariant$]
invoke AScheduler;
simplify;
■

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

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

proof[$fASInvariantRewrittenNullPartition$]
invoke AScheduler;
rewrite;
■

proof[$fASInvariantRewrittenNonNullPartition$]
invoke AScheduler;
rewrite;
■

proof[$fHS0Invariant$]
*invoke HandlerSet*0;
rewrite;
■

proof[$fHSInvariant$]
invoke HandlerSet;
rewrite;
■

proof[*lPSAddPropFreeIsUnknown*]
use lHSFreeIsUnknown;
invoke;
rearrange;
simplify;
*invoke HandlerSet*0;
rewrite;
split $aeh \cup (oseh \cup peh) = meh$;
simplify;
equality substitute meh;
with enabled (*disjointCat*, *distributeCapOverCupRight*) *rewrite*;
cases;
split $blockedAS \cap freeAS = \{\}$;
simplify;
apply extensionality to predicate $blockedAS \cap freeAS = \{\}$;
instantiate $x == p?$;
prove;
next;
apply extensionality to predicate $freeVM \cap usedVM = \{\}$;
instantiate $x == p?$;
rewrite;
next;
■

proof[*lCollectPIDSDom*]
apply dCollectPIDS;
rewrite;
apply extensionality;
prove;
cases;
apply inDom to predicate $x \in \mathrm{dom}\,\{x\_\_0 : \mathrm{dom}\, prio \bullet (x\_\_0, (\mathrm{ran}\,(prio\, x\_\_0)))\}$;
with normalization with enabled (*lRelWeakening*) *rewrite*;
next;
apply inDom to predicate $y \in \mathrm{dom}\,\{x : \mathrm{dom}\, prio \bullet (x, (\mathrm{ran}\,(prio\, x)))\}$;
with normalization with enabled (*lRelWeakening*) *rewrite*;
next;
■

proof[*lCollectPIDSApply*]
apply dCollectPIDS;
rewrite;
use pairInFunction$[\mathbb{Z}, \mathbb{P}\,\mathbb{Z}][f := \{x : \mathrm{dom}\, prio \bullet (x, \mathrm{ran}\,(prio\, x))\}, x := pr, y := \mathrm{ran}\,(prio\, pr)]$;
rearrange;
with normalization rewrite;
apply lPFunWeakening to predicate $\{x : \mathrm{dom}\, prio \bullet (x, \mathrm{ran}\,(prio\, x))\} \in \mathbb{Z} \mapsto \mathbb{P}\,\mathbb{Z}$;
with enabled (*lRelWeakening*) *rewrite* ;
■

proof[*lRanInCollectPIDS*]
apply inRanFunction;
instantiate $x == pr$;
prove;
■

proof[$lCollectPIDSOverrideKnownDom$]
apply extensionality;
prove;
cases;
apply oplusDef;
apply dCollectPIDS;
with enabled ($lRelWeakening$) *prove*;
next;
apply oplusDef to expression $collectPIDS\, prio \oplus \{(pr?, (\{p?\} \cup \operatorname{ran} (prio\, pr?)))\}$;
prove;
apply dCollectPIDS to expression $collectPIDS\, (prio \oplus \{(pr?, (prio\, pr? \frown \langle p? \rangle))\})$;
prove;
with normalization prove;
apply applyOverride;
prove;
apply dCollectPIDS;
prove;
next;
■

proof[$lCollectPIDSOverrideUnknownDom$]
apply extensionality;
prove;
cases;
apply oplusDef;
apply dCollectPIDS;
with enabled ($lRelWeakening$) *prove*;
next;
apply oplusDef ; *prove*;
apply dCollectPIDS; *prove*;
with normalization prove;
next;
■

proof[$lRanFlattenPIDSKnownDom$]
apply inPower;
apply dFlattenPIDS;
prove;
instantiate $B == \operatorname{ran} (prio\, pr?)$;
rewrite;
■

proof[*lFlattenPIDSOverrideKnownDom*]
apply dFlattenPIDS;
rewrite;
apply extensionality;
prove;
apply dlBigCupAsBigU;
rewrite;
apply dlInBigU;
prove;
cases;
apply inRanFunction to predicate $ss \in \operatorname{ran}(collectPIDS\ prio \oplus \{(pr?, (\{p?\} \cup \operatorname{ran}(prio\ pr?)))\})$;
with normalization prove;
cases;
instantiate $B == \operatorname{ran}(prio\ pr?)$;
prove;
next;
simplify;
instantiate $B == ss$;
prove;
next;
with normalization rewrite;
cases;
instantiate $ss == \{p?\} \cup \operatorname{ran}(prio\ pr?)$;
rewrite;
apply inRanFunction to predicate $\{p?\} \cup \operatorname{ran}(prio\ pr?) \in \operatorname{ran}(collectPIDS\ prio \oplus \{(pr?, (\{p?\} \cup \operatorname{ran}(p$
prove;
instantiate $x == pr?$;
prove;
next;
simplify;

cases;
instantiate $ss == \{p?\} \cup \operatorname{ran}(prio\ pr?)$;
rewrite;
apply inRan to predicate $\{p?\} \cup \operatorname{ran}(prio\ pr?) \in \operatorname{ran}(collectPIDS\ prio \oplus \{(pr?, (\{p?\} \cup \operatorname{ran}(prio\ pr?))$
with enabled (*oplusDef*) *prove*;
instantiate $x == pr?$;
rewrite;
next;
simplify;
instantiate $ss == ss\_0$;
apply inRanFunction to predicate $ss \in \operatorname{ran}(collectPIDS\ prio \oplus \{(pr?, (\{p?\} \cup \operatorname{ran}(prio\ pr?)))\})$;
apply inRanFunction to predicate $ss \in \operatorname{ran}(collectPIDS\ prio)$;
prove;
instantiate $x\_0 == x$;
apply applyOverride;
rewrite;

next;
■

proof[$lFlattenPIDSOverrideUnknownDom$]
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;
instantiate $B == ss$;
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 $ss \in \operatorname{ran} (collectPIDS\ prio \oplus \{(pr?, \{p?\})\})$;
prove;
apply applyOverride;
rewrite;
instantiate $x\_0 == x$;
rewrite;
next;
∎

proof[$lPSAddPropUniquePrio$]

$invoke\ PSAddSig;$
$invoke\ PriorityScheduler;$
$split\ \neg\ p? \in flattenPIDS\ prio;$
$cases;$
$rearrange;$
$split\ \neg\ p? \in \mathrm{ran}\,(prio\ pr?);$
$rewrite;$
$next;$
$invoke\ PrioSched2Prio;$
$use\ fVMInvariant;$
$apply\ partitionDef\ to\ predicate\ \langle\ createdVM\ \rangle \frown (\langle\ usedVM\ \rangle \frown \langle\ freeVM\ \rangle)\ \mathrm{partition}\ PID;$
$with\ enabled\ (disjointCat,\ distributeCapOverCupRight)\ rewrite;$
$rearrange;$
$apply\ extensionality\ to\ predicate\ createdVM \cup usedVM = flattenPIDS\ prio;$
$apply\ extensionality\ to\ predicate\ freeVM \cap usedVM = \{\};$
$apply\ extensionality\ to\ predicate\ createdVM \cap freeVM = \{\};$
$rewrite;$
$instantiate\ y == p?;$
$instantiate\ x\_\_0 == p?;$
$instantiate\ x\_\_1 == p?;$
$rewrite;$
$next;$
$\blacksquare$

proof[$PSAddFSB$]
use lPSAddPropUniquePrio;
use lPSAddPropFreeIsUnknown;
rearrange;
simplify;

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

invoke $PSAddPrio0$;
invoke $PSAddKnownPrio$;
invoke $PSAddNewPrio$;
invoke $PSAdd0$;
invoke Ξ $PriorityScheduler$;
invoke Δ $PriorityScheduler$;
rewrite;
rearrange;

split $free = \varnothing$;
rewrite;
cases;
invoke $PriorityScheduler$;
invoke $PrioSched2Prio$;
invoke $PrioSched1ASHS$;
invoke $PrioSched0Comp$;
rearrange;
simplify;
rearrange;
instantiate $p! == maxpid$;
rewrite;
next;

invoke $VMCreate$;
invoke $ACreate$;
invoke $AddHandler$;

invoke $OutOfProcessErr$;
invoke $VMOkayErr$;
invoke $ACreateErr$;
invoke $ASchedErr$;
invoke $ASchedOkay$;
invoke Ξ VM;
invoke Ξ $AScheduler$;
rewrite;

split $p? \in free$;
rewrite;
cases;
invoke $ContainsHandler$;
invoke Ξ $HandlerSet$;

invoke $VMCreate0$;
invoke Δ VM;
invoke VM;
rewrite;
rearrange;

invoke $ACreate0$;
invoke Δ $AScheduler$;
invoke $AScheduler$;
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 `:-)`.

section *all_proofs* **parents** *basic_defs_proofs*, *vm_proofs*, *ascheduler_proofs*, *scj_process_proofs*, *scj_toolkit_proofs*

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).

Z Declarations	**This Chapter**	**Globally**
Unboxed items	0	84
Axiomatic definitions	0	28
Generic axiomatic defs.	0	2
Schemas	0	77
Generic schemas	0	1
Theorems	0	192
Proofs	95	97
Total	**95**	**481**

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

Z Declarations	**This Chapter**	**Globally**
Unboxed items	0	84
Axiomatic definitions	0	28
Generic axiomatic defs.	0	2
Schemas	0	77
Generic schemas	0	1
Theorems	0	192
Proofs	95	97
Total	**95**	**481**
Circus **Declarations**	**This Chapter**	**Total**
Channel decls.	0	51
Channel set decls.	0	13
Process decls.	0	10
Process ref. assertions	0	0
Name sets	0	0
Actions	2	83
Action ref. assertions	0	0
Total	202	**638**

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 CSP_m. 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 CSP_m.

C.1 Version withou *SCJProces* explicitly given

This version abstracts *SCJProcess* as simply *PID*, and hence uses a mapping between *PID*s 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_field(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 PPSgetMinHWPrio, PPSgetMaxHWPrio : 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 -----

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) & 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

PSExecute = PSRun /\ PSCatch

--- Main Action ---

Main = InitPSSt ; PSExecute

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 *SCJProcess* as a separate CSP_m 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 *SCJProcess* is meant to manage. Notice we also introduce guards for the *Circus* calculated preconditions.

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

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

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

--- channels

channel PCIHdisableCall, PCIHdisableRet
channel PCIHenableCall, PCIHenableRet
channel PCIHstartClockHandlerCall  : PID
channel PCIHstartClockHandlerRet
channel PCIHinitialise -- SCHID . NAT . CLOCKID?
channel PCIHhandleCall, PCIHhandleRet
channel PCIHyieldCall, PCIHyieldRet
channel PCIHregisterCall, PCIHregisterRet
channel PCIHcatchErrorCall, PCIHcatchErrorRet

channel PPSIgetNextProcessCall
channel PPSIgetNextProcessRet : PID

channel VMthrow, VMcihRegister
channel VMcreateProcess: PID
channel VMtransfer : PID . PID

VMChan = {| VMthrow, VMcihRegister, VMcreateProcess |}

--- State Components ---

--- 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 ; PCIHinitialise -> 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)
-- in the code there is more internal SCJ API managed of missions etc.

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) &
PMEHhandleAsyncEvent ! 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.handleAssyncEvent() 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 PPSgetMinHWPrio, PPSgetMaxHWPrio : HW_PRIO
channel PPSgetMinPrio, PPSgetMaxPrio: 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?

PSExecute = PSRun /\ PSCatch

--- Main Action ---

Main = InitPSSt ; PSExecute

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

[1] Ana Cavalcanti, Augusto Sampaio, and Jim Woodcock. A refinement strategy for circus. *Formal Aspects of Computing*, 15(2-3):146–181, 2003.

[2] Leo Freitas and Jim Woodcock. A chain datatype in Z. *International Journal of Software and Informatics*, 3(2-3):357–374, 2009.

[3] Open Group. Safety-critical java technology specification. JSR 302, The Open Group, April 2014.

[4] Jim Woodcock and Jim Davies. *Using Z*. Prentice Hall International, 1996.