Using *Circus* to Verify Safety-Critical Java Level 2 Programs

Matt Luckcuck

10th of May 2018

Using *Circus* to Verify Safety-Critical Java Level 2 Programs

Matt Luckcuck

10th of May 2018

Supervisors:
Ana Cavalcanti and Andy Wellings

Using *Circus* to Verify Safety-Critical Java Level 2 Programs

Matt Luckcuck

10th of May 2018

Supervisors:
Ana Cavalcanti and Andy Wellings

Outline

Outline

- Java in Safety-Critical Systems
- Safety-Critical Java
  - Safety-Critical Java Level 2
- Modelling and Translation
  - *Circus* Intro
  - Model
  - Translation
- Model Utility
- Summary

Java in Safety-Critical Systems

Java in Safety-Critical Systems

Java

- Java not traditionally associated with safety-critical programs
- More abstraction, less control…
  - Garbage collected memory management
  - Poor scheduling control

Java in Safety-Critical Systems

Java

- Java not traditionally associated with safety-critical programs
- More abstraction, less control…
  - Garbage collected memory management
  - Poor scheduling control

*“The intrinsic safety of the standard language is irrelevant, **it is how safe the use of the language can be made that matters**”* – Hatton, *Safer C (1995)*

Java in Safety-Critical Systems

Java

- Interesting for safety-critical systems:
  - Strong typing
  - Precise definition
  - Widely understood
  - Language features e.g. exception handling
- Long standing effort to improve Java…
  - Java Community Process's **Java Specification Requests (JSR)**

Java in Safety-Critical Systems

Real-Time Specification for Java (RTSJ)

- Java Community Process: JSR 1
- RTSJ addresses some of the Java's problems…
  - Region-based memory
  - Better memory control
  - Better scheduling control
- Complex for safety-critical programs

Safety-Critical Java

Safety-Critical Java

Safety-Critical Java (SCJ)…

- New language for applications that must be certified
  - Aeroplanes
  - Robots
  - Etc.
- Java Community Process: JSR 302
- Builds on the Real-Time Specification for Java (RTSJ)
- Simpler, hierarchical programming paradigm
- (Natural) language specification $\sim$ 112 pages…
  - Defines $\sim$ 36 classes and interfaces
  - Does not cover verification

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Borrows from the RTSJ…
  - Real-time abstractions
  - Memory areas

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Borrows from the RTSJ…
  - Real-time abstractions
  - Memory areas
- Region-based hierarchical memory management

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Borrows from the RTSJ…
  - Real-time abstractions
  - Memory areas
- Region-based hierarchical memory management
- Fixed priority scheduling with Priority Ceiling Emulation

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Borrows from the RTSJ…
  - Real-time abstractions
  - Memory areas
- Region-based hierarchical memory management
- Fixed priority scheduling with Priority Ceiling Emulation
- Three feature sets (compliance levels)
  - Level 0
  - Level 1
  - Level 2

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Borrows from the RTSJ…
  - Real-time abstractions
  - Memory areas
- Region-based hierarchical memory management
- Fixed priority scheduling with Priority Ceiling Emulation
- Three feature sets (compliance levels)
  - Level 0
  - Level 1
  - **Level 2**

Safety-Critical Java

Compliance Levels

- Level 0:
  - Single processor
  - Cyclic executive
- Level 1:
  - Introduce concurrency
  - More release patterns
- Level 2:
  - Highly concurrent
  - Multi-processor
  - Complicated release patterns
  - Suspension

Release Pattern

When a process becomes available for execution

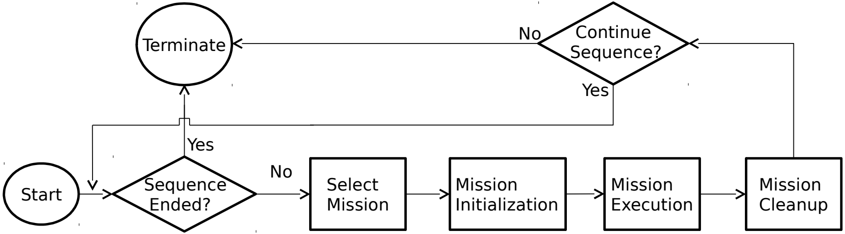
Safety-Critical Java

SCJ API

- `Safelet`: controls the program and starts the *Mission Sequencer*
- `MissionSequencer`: instantiates and starts a sequence of *Missions*
- `Mission`: controls a set of processes, represented by subclasses of *Managed Schedulable*
- `ManagedSchedulable`: super-type of all four process types…
  - `PeriodicEventHandler`
  - `AperiodicEventHandler`
  - `OneShotEventHandler`
  - `ManagedThread`

Safety-Critical Java

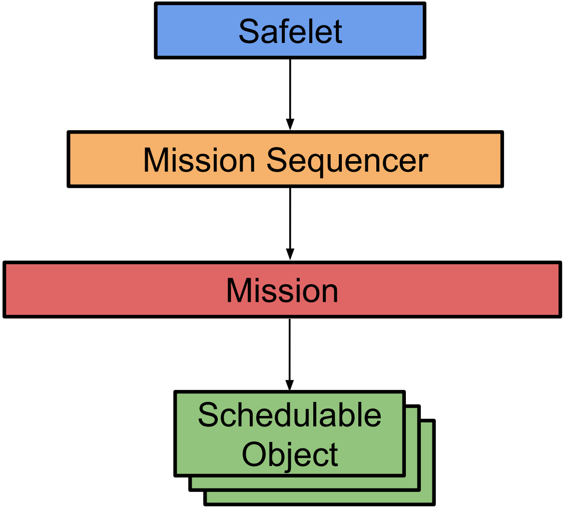
Mission Sequencer



Mission Phases

1. **Initialize**: creates and registers schedulables
2. **Execute**: simultaneously activate mission's schedulables
3. **Cleanup**: reset data structures

Safety-Critical Java



SCJ Level 2

SCJ Level 2 Features

- Access to Java suspension methods
  - `wait()`, `notify()`, etc
- Access to all release patterns:
  - periodic
  - aperiodic
  - run-once after a time offset
  - run-to-completion
- Complex program structures due to more concurrent components
  - Multiple Mission Sequencers enable multiple Missions to be active
  - One active Mission per Mission Sequencer
  - Schedulables from any active Mission may preempt, based on their priorities

SCJ Level 2

SCJ Level 2 Features

- Access to Java suspension methods
  - `wait()`, `notify()`, etc
- Access to all release patterns:
  - periodic
  - aperiodic
  - run-once after a time offset
  - run-to-completion
- Complex program structures due to more concurrent components
  - Multiple Mission Sequencers enable multiple Missions to be active
  - One active Mission per Mission Sequencer
  - Schedulables from any active Mission may preempt, based on their priorities

SCJ Level 2

SCJ Level 2 Features

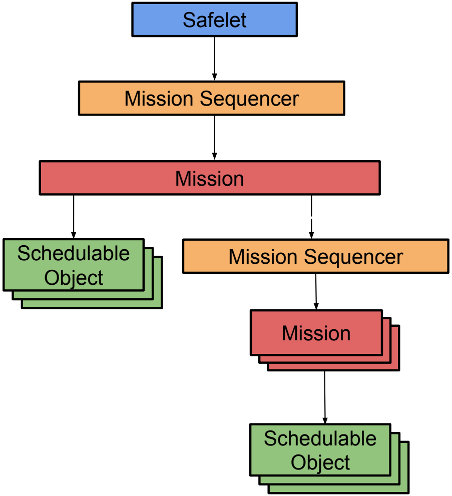
- Access to Java suspension methods
  - `wait()`, `notify()`, etc
- Access to all release patterns:
  - periodic
  - aperiodic
  - run-once after a time offset
  - run-to-completion
- Complex program structures due to more concurrent components
  - Multiple Mission Sequencers enable multiple Missions to be active
  - One active Mission per Mission Sequencer
  - Schedulables from any active Mission may preempt, based on their priorities

SCJ Level 2

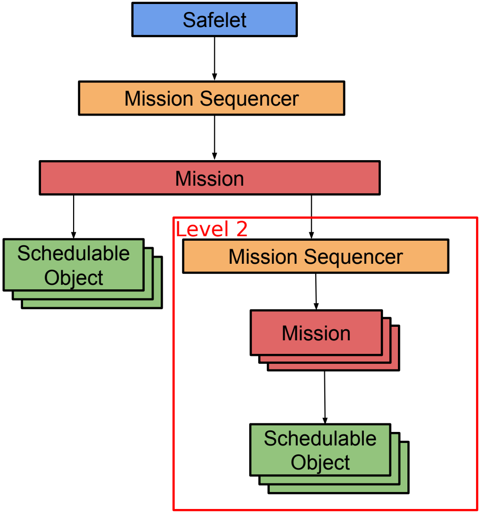
SCJ Level 2 Features

- Access to Java suspension methods
  - `wait()`, `notify()`, etc
- Access to all release patterns:
  - periodic
  - aperiodic
  - run-once after a time offset
  - run-to-completion
- Complex program structures due to more concurrent components
  - Multiple Mission Sequencers enable multiple Missions to be active
  - One active Mission per Mission Sequencer
  - Schedulables from any active Mission may preempt, based on their priorities

SCJ Level 2


Safelet
Mission Sequencer
Mission
Schedulable Object
Mission Sequencer
Mission
Schedulable Object

SCJ Level 2


Safelet
Mission Sequencer
Mission
Schedulable Object
Level 2
Mission Sequencer
Mission
Schedulable Object

Modelling and Translation

Modelling Approach

Circus Language

- Combination of **Z** and **CSP**
  - Captures both State and Behaviour
- Organised around Processes
  - State component (**Z**) to hold variables
  - Actions (**Z** and **CSP**) to perform behaviours
  - Main action specifies overall behaviour
- Communication through **CSP** channels

Modelling Approach

$\mathbf{process}\ P \mathrel{\widehat{=}}$

$State$

$var1 : \mathbb{B}$

$var2 : \mathbb{Z}$

$Init$

$State'$

$var1' = \mathbf{False}$

$var2' = 42$

$Action1 \mathrel{\widehat{=}} chan1 \longrightarrow \mathbf{Skip}$

$Action2 \mathrel{\widehat{=}} chan2 \longrightarrow \mathbf{Skip}$

$\bullet\ Action1 \mathrel{\Box} Action2$

Modelling Approach

Building the Model

- Two Components:
  - SCJ's Application Programming Interface (API)
  - Templates for SCJ Programs
- *Agnostic* of Java
- Combine the two components to capture a program's behaviour
- Enables program verification
  - via CSP's Model-Checker:

Modelling Approach

Building the Model

- Two Components:
  - SCJ’s Application Programming Interface (API)
  - Templates for SCJ Programs
- *Agnostic* of Java
- Combine the two components to capture a program’s behaviour
- Enables program verification
  - via CSP’s Model-Checker: FDR

Modelling Approach

Building the Model

- Two Components:
  - SCJ's Application Programming Interface (API)
  - Templates for SCJ Programs
- *Agnostic* of Java
- Combine the two components to capture a program's behaviour
- Enables program verification
  - via CSP's Model-Checker: FDR

Modelling Approach

Building the Model

- Two Components:
  - SCJ’s Application Programming Interface (API)
  - Templates for SCJ Programs
- *Agnostic* of Java
- Combine the two components to capture a program’s behaviour
- Enables program verification
  - via CSP’s Model-Checker: FDR

FDR4

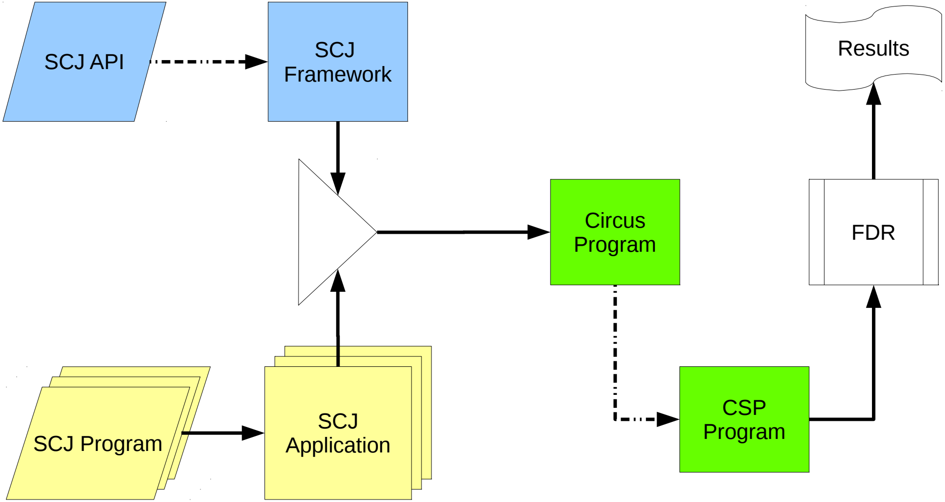
Modelling Approach

Model

- Captures the SCJ Level 2 *paradigm*
  - $\sim$ 3300 lines of *Circus*
- Abstracts away from Java…
  - Scheduling
  - Resources (E.g. Memory)
  - Exceptions
- Expands on a model of SCJ Level 1[1]($\sim$ 700 lines)
  - Level 2 features
  - API changes
  - Also Level 1 features not covered

[1] Zeyda et al. *Circus Models for Safety-Critical Java Programs*. Computer Journal (2014)

Modelling Approach



Translation

Overview

- Extended existing Level 1 translation tool
  - Level 2 Tool: TightRope
  - Automatic generation of models from Level 2 programs

Translation Using TightRope

- Compiles the program to generate Abstract Syntax Trees (ASTs)
  - No annotations needed, unlike Level 1 tool
- Extract program-specific information
- Drop the information into gaps in a template
  - Using FreeMarker template engine

Translation

Translation Using TightRope

- TightRope, produces models from code:

Producers–Consumers	6 classes	∼1.2 seconds
Aircraft	25 classes	∼2.3 seconds

- … Some expression rewriting required
  - Translating arbitrary Java code

Model Utility

Model Utility

Level 2 Problems…

- Modelling Level 2 exposed problems with termination[2] :
  1. Termination of `MissionSequencers`
  2. Termination of waiting threads
- No one had thought hard enough about Level 2

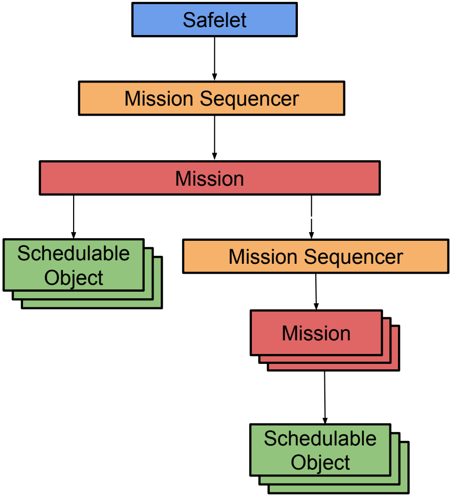
[2] Luckcuck, Wellings, Cavalcanti. *Safety-Critical Java: Level 2 in Practice.* Concurrency and Computation: Practice and Experience. (2017)

Model Utility

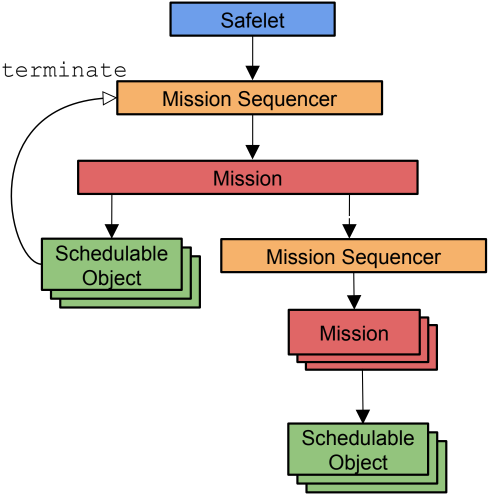
Termination of `Mission Sequencers`

- Originally, any schedulable could terminate any mission sequencer
- Intended to allow a schedulable to terminate the program
- But with Level 2 having multiple active missions…
  - Poor link to startup structure
  - Breaks encapsulation of `Mission`
  - Chaotic

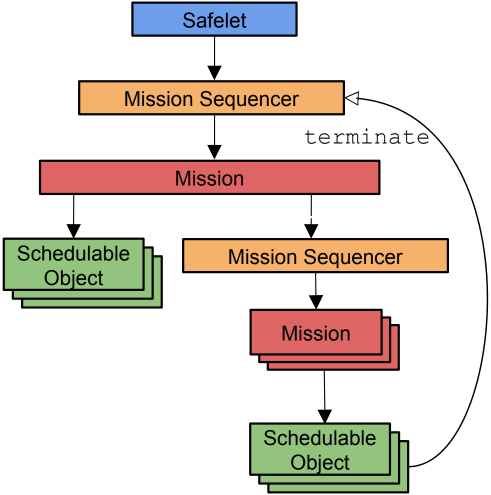
Model Utility



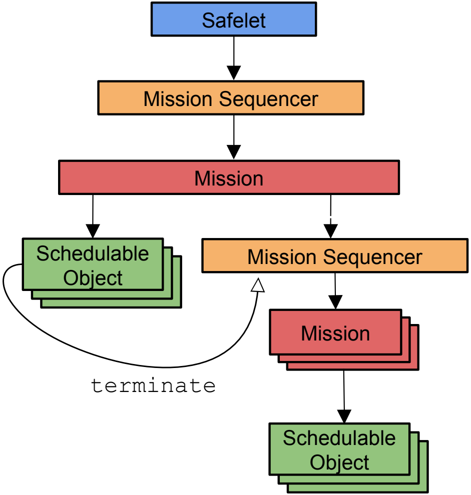
Model Utility



Model Utility



Model Utility



Model Utility

Termination of `Mission Sequencers`

- Proposed that when a a `Mission` terminates it tells its `MissionSequencer` if it should terminate too
  - Mirrors startup
  - Restores encapsulation of `Mission`
- Used my model to compare original and proposed protocol
- Checked that the proposed protocol works
- Showed that it had 94.5% fewer states
- Adopted SCJ v0.96

Model Utility

Termination of waiting threads

- According to the language specification, during Mission termination the infrastructure will…
  - '… wait for all the Managed Schedulable Objects associated with this Mission to terminate'
- If a schedulable is blocked at this point, it will never terminate

Model Utility

Termination of Waiting Threads

- Proposed either:
  a) SCJ Interrupts all schedulables during termination, or;
  b) Add a new method to the schedulable interface that programmers can use to interrupt a blocked schedulable
- SCJ specification now highlights termination in Level 2:
  - Second proposal, but in an existing method
  - Provides a uniform way of handling custom termination behaviour

Model Utility

Top-Down Development

Target for refinement-based development of SCJ programs[3]

- Correct-by-construction approach
- Refines abstract specifications to concrete specifications…
  - That capture the SCJ paradigm
- Enables this, but out of scope

[3] Cavalcanti, Sampaio, and Woodcock. *A Refinement Strategy for Circus*. Formal Aspects of Computing. (2003)

Model Utility

Bottom-Up Development

Translation from SCJ code to model, for program verification

- Model-Checking and Animation
- Catches certain program errors…
    - Deadlock
    - Divergence/Exceptions

Program Verification

Overview

- We want to be able to use this model to verify programs…
- But there is no model checker for *Circus*
- So, we use industry-proven CSP model checker FDR3…
- But, this requires another translation…

Program Verification

Why Use *Circus*?

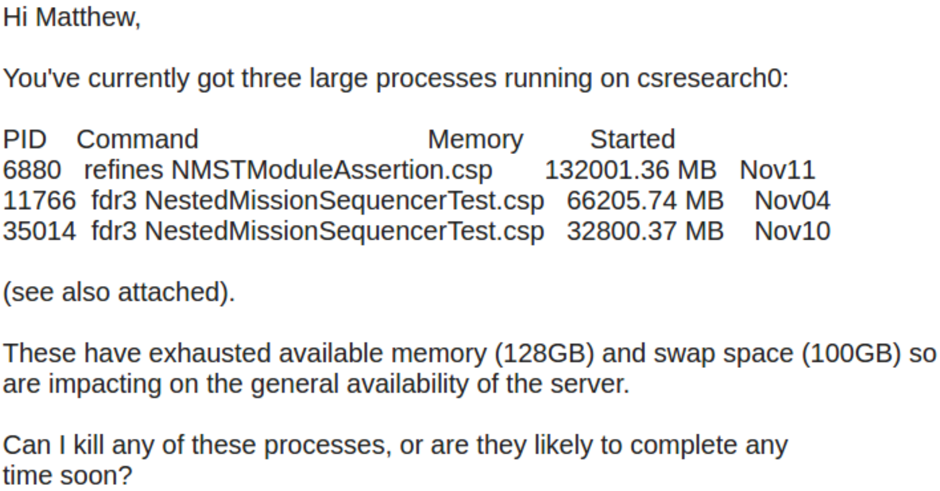
Tight integration of state and behaviour

Program Verification

Circus to `CSPm`

- `CSPm` is the machine-readable version of CSP, used by FDR
- Informal translation from *Circus* to `CSPm`
  - State in *Circus* processes becomes state process in `CSPm`
  - Most behaviour in *Circus* translates straight into `CSPm`
- However, treatment of state in initial translations produced intractable models…

Program Verification



Hi Matthew,

You've currently got three large processes running on csresearch0:

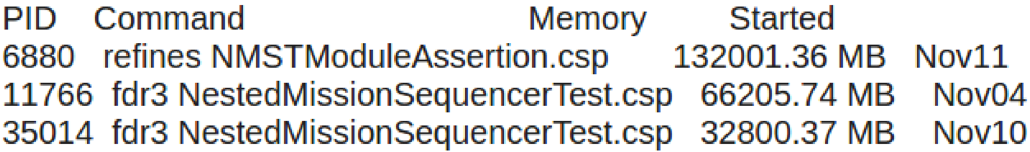
PID	Command	Memory	Started
6880	refines NMSTModuleAssertion.csp	132001.36 MB	Nov11
11766	fdr3 NestedMissionSequencerTest.csp	66205.74 MB	Nov04
35014	fdr3 NestedMissionSequencerTest.csp	32800.37 MB	Nov10

(see also attached).

These have exhausted available memory (128GB) and swap space (100GB) so are impacting on the general availability of the server.

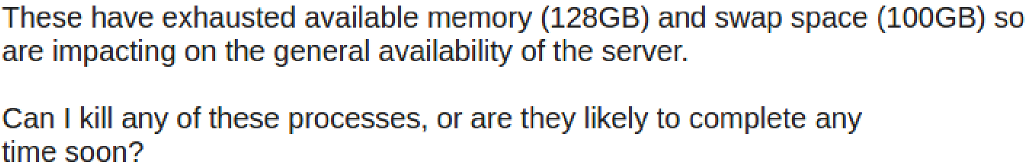
Can I kill any of these processes, or are they likely to complete any time soon?

Program Verification



```
PID    Command                                    Memory         Started
6880   refines NMSTModuleAssertion.csp       132001.36 MB   Nov11
11766  fdr3 NestedMissionSequencerTest.csp   66205.74 MB    Nov04
35014  fdr3 NestedMissionSequencerTest.csp   32800.37 MB    Nov10
```

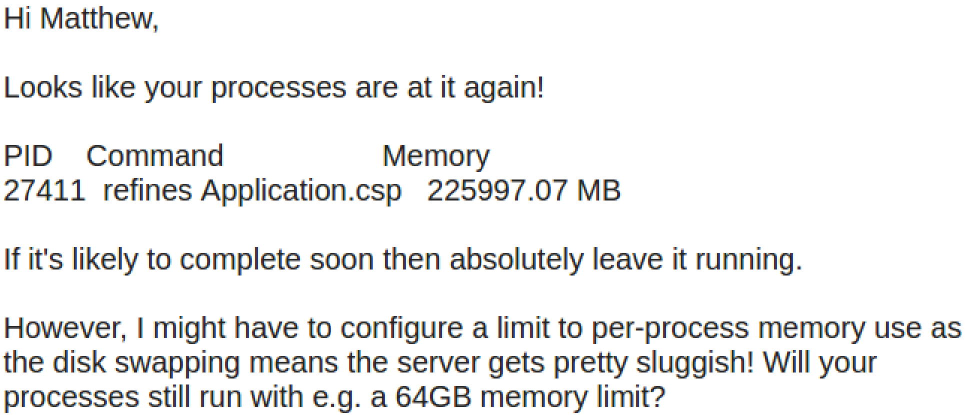
Program Verification



These have exhausted available memory (128GB) and swap space (100GB) so are impacting on the general availability of the server.

Can I kill any of these processes, or are they likely to complete any time soon?

Program Verification



Hi Matthew,

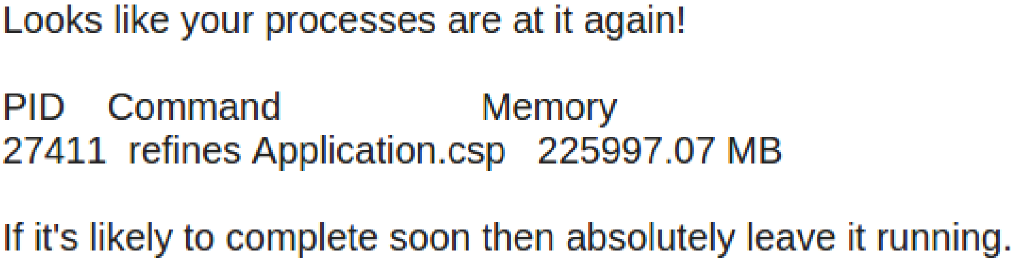
Looks like your processes are at it again!

```
PID    Command                     Memory
27411  refines Application.csp   225997.07 MB
```

If it's likely to complete soon then absolutely leave it running.

However, I might have to configure a limit to per-process memory use as the disk swapping means the server gets pretty sluggish! Will your processes still run with e.g. a 64GB memory limit?

Program Verification



Looks like your processes are at it again!

```
PID    Command                   Memory
27411  refines Application.csp   225997.07 MB
```

If it's likely to complete soon then absolutely leave it running.

Program Verification

Personal Best

226GB on csresearch0 in 1 process

Program Analysis

Circus to CSPm

- Improved the CSPm model with the help of Tom Gibson-Robinson (FDR's maintainer) at Oxford University
- Building distributed CSP processes of 'complex' data structures:
  - Sets
  - Sequences
  - Priority Queue
- Made model-checking tractable…

Process controlling a set of `value`
(Add or remove a number)

		`value=` $\{0..20\}$
P_1	Compiled	7438.52s ($\sim$2hrs)
	Checked	3.84s
P_2	Compiled	0.01s
	Checked	4.41s

Program Verification

Animation and Model Checking in FDR3

- Animation:
  - Step through model to compare to API or running program
- Model-Checking:
  - Deadlock- and divergence-freedom
  - Enables custom checks: exceptions, particular program behaviours, etc

Summary

Model Summary

Model

- Close correspondence with the SCJ API
  - Validated against API
- Extends existing Level 1 model…
  - Validated against API
- Our modelling effort simplified SCJ's termination protocol…
  - Adopted in v0.96

Program Verification Summary

Program Analysis

- Modelling approach enables verification technique
- Translation to CSP, for FDR
  - *Circus* is close to CSP
  - Scope for automation
- Recent tool that automates *Circus* to `CSPm` translation[4]
  - Limitations on input models
  - I suspect similar problems with state explosion

[4] Beg and Butterfield. *Development of a Prototype Translator from Circus to CSPM.* ICOSST (2015)

Future Work

Future Work

- More general translation approach
  - Less expression rewriting
- Automate *Circus* to `CSPm` translation
  - Dealing with data
  - Simplify customised checks

Thank you for listening