Machine Assisted Reasoning for Multi-Threaded Java Bytecode

Mikael Lagerkvist

April 2005

Goal of Project

- Define an operational semantics for an interesting subset of the multi-threaded Java Virtual Machine.
- Embed the semantics in a proof tool for machine assisted reasoning.
- Do some examples to show the formalization in action.

Possible motivation

- Formalize the behaviour of Java threads
- Prove properties of programs
- Evaluate the proof tool used

Operational semantics

- A method for describing the meaning of programs
- Defined as a transition relation $s \xrightarrow{\alpha} s'$ for systems s and s', and action α.
- Usually defined through rules, for example:

$$\text{SeqComp}\ \frac{c_1 \xrightarrow{\alpha} c_1'}{c_1; c_2 \xrightarrow{\alpha} c_1'; c_2}$$

The μ-calculus

- First order logic as the base
- Fixed points of recursive predicates
- Expressive, "one and a half order" logic

VeriCode Proof Tool (VCPT)

- Proof assistant
- Support for operational semantics
  - The transition relation is a predicate of type $\mathit{system} \longrightarrow \mathit{action} \longrightarrow \mathit{system}$
  - $s \xrightarrow{\alpha} s'$ is expressed as $\mathit{transRel}\ s\ \alpha\ s'$
  - Modalities on actions
- Lazy induction

Java

Java is a modern object-oriented, garbage-collected, multi-threaded, distributed, portable, interpreted programming language.

The Java Virtual Machine (JVM)

The JVM is a platform for running compiled Java programs.

- Stacks for computation
- Direct encoding of class hierarchies
- Parallel threads of execution
  Any scheduling policy is valid!

JVM Memory layout

- A set of running threads
- A heap of allocated class instances
- Constant definitions (constant pool)

The `putfield(i)` instruction

The instruction `putfield` is followed in the code stream by an argument i.

The execution takes values *val* and *objref* from the stack.

The result is that field i of instance *objref* is set to value *val*.

Helpful formulae

Some formulae were developed to manipulate lists. For example:

at $at\ List\ Index\ Element$
Ex: $at\ [g, e, c]\ 1\ e$

set $set\ List\ Index\ Element\ List'$
Ex: $at\ [g, e, c]\ 1\ h\ [g, h, c]$

Excluded features

The following features were excluded.

- Exceptions
- Class hierarchies
- Datatypes other than natural numbers
- Distribution

Semantics overview

Close resemblance to the JVM definition.

Semantics in two levels.

- Method level transitions ($\rightarrow_m$)
- System level transitions ($\rightarrow$)

iadd at method-level

$$\text{IAdd}\frac{at\ CS\ PC\ iadd \qquad N_1 + N_2 = N}{\langle CS, PC, [N_1, N_2|VS], LS\rangle \rightarrow_m \langle CS, PC+1, [N|VS], LS\rangle}$$

iadd at system-level

$$at\ \ Ths\ \ I\ \ \langle TId, [F|T]\rangle$$

$$F \rightarrow_m F'$$

$$\text{Compute}\frac{set\ \ Ths\ \ I\ \ \langle TId, [F'|T]\rangle\ \ Ths'}{\langle Ths, Hp, CP\rangle \rightarrow \langle Ths', Hp, CP\rangle}$$

The Semantics in VCPT

- Direct embedding as explicit formula
- Follows the formal semantics closely
- Automation of derivations for concrete systems

Scheduling of threads

- The unconstrained choice of next thread in the semantics corresponds to some legal choice of thread
- Next state is described as the disjunction of the legal choices

A Simple Program

```
 1 class Worker extends Thread {
 2    Container objref;
 3    public Worker(Container objref) {
 4      this.objref = objref;
 5    }
 6    public void run() {
 7      while(true) {
 8        synchronized(objref) {
 9          // do something
10        }
11      }
12    }
13 }
```

One Thread in Bytecode

Code

PC	Instruction
0	goto(1)
1	load(0)
2	getfield(0)
3	dup()
4	store(1)
5	monitorenter()
6	load(1)
7	monitorexit()
8	goto(1)

Data referenced

Local variables:
- 0: Reference to class instance.
- 1: Stored Container reference.

Class variables:
- 0: Reference to Container instance.

Proving properties

We will focus on which thread gets to enter the critical section.

The predicate `t1inCS` (`t2inCS`) is true if thread 1 (thread 2) is in its critical section.

Simple property

$\neg Eventually(t1inCS)$ There is no fairness in the system.

Simple property

$Sometime(\neg t1inCS \wedge Eventually(t1inCS))$ The queue of a mutual exclusion lock is fair.

Slightly more advanced property

$Always(\neg(t1inCS \wedge t2inCS))$ The two threads are never in their critical section at the same time.

Contributions

The contributions of the thesis are the following.

- Clear operational operational semantics of Java Bytecode
- A treatment of multiple threads in the JVM
- Embedding the JVM semantics in a powerful and interesting proof assistant

Conclusions

- There is much additional effort involved in making a toolfor proving properties of actual programs
- The abstract behaviour of Java threads are relatively easy to describe as an operational semantics
- VCPT is an interesting environment for this kind of work

Further work

- Model more of the JVM (exceptions, class hierarchies,...)
- Better treatment of naming issues
- Integrate more security-guarantees of the JVM
- Add rewrite simplification to VCPT.
- Investigate potential for raising the level of abstraction

Questions?