The Java Virtual Machine

Mark Reynolds

Outline

- Virtual machine
- Properties of the JVM
- Static vs dynamic properties
- Instruction model
- Machine state model

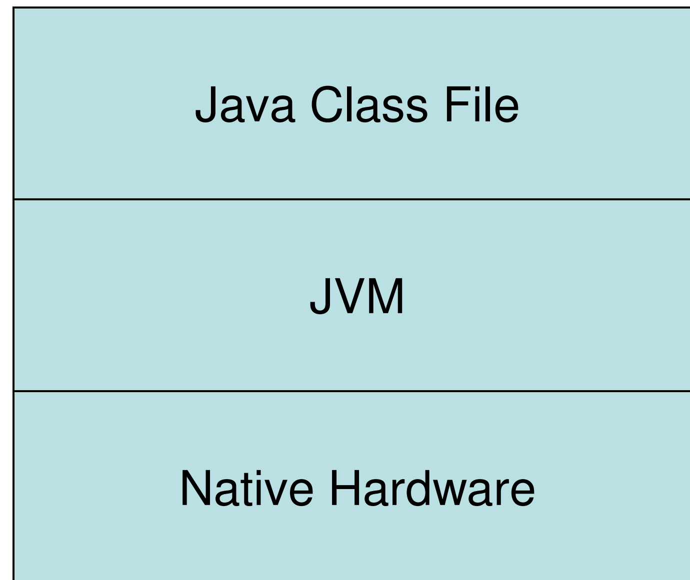
Executables and Class Files

- C program, prog.c
  - Compile with: gcc –o prog prog.c
  - Result: “prog” a *native* executable
  - Runs on the underlying hardware without any helper programs
- Java program, prog.java
  - Compile with: javac prog.java
  - Result: “prog.class” a Java class file
  - Does not run on the underlying hardware
  - Requires a helper program to run: java –cp . prog

The Java Virtual Machine

- Class files are binary files containing machine instructions for the Java Virtual Machine (JVM)
- The JVM is a *virtual* machine
  - Implements an abstract machine model
  - Implements an instruction set that runs on that machine
- Java is “write once, run anywhere” because all Java executables run on the abstract machine, not the native hardware
  - The JVM itself runs on the native hardware
  - Can think of the JVM as a translator between the abstract machine instruction set and the native instruction set

The Java Execution Hierarchy



JVM properties

- Intel x86 architecture
  - About 1400 distinct machine instructions
  - Limited number of registers
  - Extensive use of the stack for passing arguments/return values
  - Weakly typed at the machine code level
- JVM architecture
  - About 240 distinct JVM instructions
  - Up to 32k (virtual) registers per method
  - Each method has its own stack
  - Arguments, return values passed in registers
  - Strongly typed at the machine code level

Example: Add

- x86 add
  - 22 different variants depending on operands
  - Syntax checking of arguments, no type checking
  - Possible to perform semantically invalid actions:
    - register %eax holds an aligned address
    - add %eax, 7
- JVM add
  - 4 variants: iadd, ladd, fadd, dadd
  - All variants pop two items from the stack, type check, add, and then push the result back on the stack

Static properties

- Properties that never change
  - Instruction properties
    - Instruction length (all add instructions are one byte long)
    - Stack utilization (all add instructions pop two, then push one)
    - Register utilization (add instructions do not use any registers)
    - Instruction location within a given method (byte offset to that instruction)

Dynamic Properties

- Properties that change as a method executes
  - The “current” instruction
  - The program counter (pc) or instruction pointer (ip)
  - Stack depth
  - Registers in use
  - Types of items on the stack
  - Types of items in each register

Alloy model of a JVM method

- A signature that describes instructions (static)
- A signature that describes the machine state (dynamic)
- Alloy statements that initialize the model
- State transition rules that move the model from one state to the next as each instruction is synthetically executed

Instruction Model

```
abstract sig Instruction {
   map:   Int                    // instruction offset
   len:   Int                    // instruction length
   s1:    Int                    // initial stack mod
   s2:    Int                    // final stack mod
   …                             // other static props
}
one sig iadd, ladd, fadd, dadd
   extends Instruction { }
```

State model

```
sig State {
   current:  Instruction        // the current ins
   pc:        Int                // program counter
   sdepth:   Int                // stack depth
   …                             // other dyn props
}
```

Initializers

The notation -> is used to compose elements into relations

- a->b means that "a" is mapped to "b" by some relation

Use a fact to capture the complete mapping of a relation

fact {

len = iadd->1 + ladd->1 + fadd->1 + dadd->1

}

The domain of the *len* relation is *Instruction*, the range is *Int*

State transition rules

- Examine the “River Crossing” model in the Alloy tutorial
- Need Alloy rules for transitioning between current state S and next state S’
  - What is the relationship between S.current and S’.current?
  - What is the relationship between S.pc and S’.pc?
  - What is the relationship between S.sdepth and S’.sdepth?