Formal Methods for Java

Lecture 3: Operational Semantics (Part 2)

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

May 3, 2017

Operational Semantics for Java

Idea: define transition system for Java

Definition (Transition System)

A transition system (TS) is a structure $TS = (Q, Act, \rightarrow)$, where

- Q is a set of states,
- Act a set of actions,
- $\rightarrow \subseteq Q \times Act \times Q$ the transition relation.

- Q reflects the current dynamic state (heap and local variables).
- Act is the executed code.
- Idea from: D. v. Oheimb, T. Nipkow, Machine-checking the Java specification: Proving type-safety, 1999

State of a Java Program

The state of a Java program gives valuations local and global (heap) variables.

- $Q = Heap \times Local$
- $Heap = Address \rightarrow Class \times \text{seq}\ Value$
- $Local = Identifier \rightarrow Value$
- $Value = \mathbb{Z}, Address \subseteq \mathbb{Z}$

A state is denoted as $(heap, lcl)$, where $heap : Heap$ and $lcl : Local$.

Actions of a Java Program

An action of a Java Program is either

- the evaluation of an expression e to a value v, denoted as $e \triangleright v$, or
- a Java statement, or
- a Java code block.

Note that expressions with side-effects can modify the current state

Rules

Definition (Inference Rules)

A rule of inference

$$\frac{F_1 \ldots F_n}{G}, \text{ where } \ldots$$

is a decidable relation between formulae. The formulae $F_1, \ldots, F_n$ are called the premises of the rule and G is called the conclusion.
If $n = 0$ the rule is called an axiom schema. In this case the bar may be omitted.

The intuition of a rule is that if all premises hold, the conclusion also holds.

Rules for Java Expressions

axiom for evaluating local variables:

$$(heap, lcl) \xrightarrow{x \triangleright lcl(x)} (heap, lcl)$$

axiom for evaluating constants:

$$(heap, lcl) \xrightarrow{c \triangleright c} (heap, lcl)$$

rule for field access:

$$\frac{(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')}{(heap, lcl) \xrightarrow{e.fld \triangleright heap'(v)(idx)} (heap', lcl')},$$

where *idx* is the index of the field *fld* in the object $heap'(v)$

Rules for Assignment Expressions

rule for assignment to local:

$$\frac{(heap, lcl) \xrightarrow{e \rhd v} (heap', lcl')}{(heap, lcl) \xrightarrow{x = e \rhd v} (heap', lcl' \oplus \{x \mapsto v\})}$$

rule for assignment to field:

$$\frac{\begin{array}{c}(heap_1, lcl_1) \xrightarrow{e_1 \rhd v_1} (heap_2, lcl_2) \\ (heap_2, lcl_2) \xrightarrow{e_2 \rhd v_2} (heap_3, lcl_3)\end{array}}{(heap_1, lcl_1) \xrightarrow{e_1.fld = e_2 \rhd v_2} (heap_4, lcl_3)},$$

where $heap_4 = heap_3 \oplus \{(v_1, idx) \mapsto v_2\}$ and idx is the index of the field fld in the object at $heap_3(v_1)$.

Rules for Java Statements

expression statement (assignment or method call):

$$\frac{(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')}{(heap, lcl) \xrightarrow{e;} (heap', lcl')}$$

sequence of statements:

$$\frac{(heap_1, lcl_1) \xrightarrow{s_1} (heap_2, lcl_2) \quad (heap_2, lcl_2) \xrightarrow{s_2} (heap_3, lcl_3)}{(heap_1, lcl_1) \xrightarrow{s_1\ s_2} (heap_3, lcl_3)}$$

Rules for Java Statements

if statement:

$$\frac{(heap_1, lcl_1) \xrightarrow{e \triangleright v} (heap_2, lcl_2) \quad (heap_2, lcl_2) \xrightarrow{s_1} (heap_3, lcl_3)}{(heap_1, lcl_1) \xrightarrow{\textbf{if}(e)\, s_1 \textbf{else} s_2} (heap_3, lcl_3)}, \text{where } v \neq 0$$

$$\frac{(heap_1, lcl_1) \xrightarrow{e \triangleright v} (heap_2, lcl_2) \quad (heap_2, lcl_2) \xrightarrow{s_2} (heap_3, lcl_3)}{(heap_1, lcl_1) \xrightarrow{\textbf{if}(e)\, s_1 \textbf{else} s_2} (heap_3, lcl_3)}, \text{where } v = 0$$

while statement:

$$\frac{(heap_1, lcl_1) \xrightarrow{\textbf{if}(e)\{s\ \textbf{while}(e)\, s\}} (heap_2, lcl_2)}{(heap_1, lcl_1) \xrightarrow{\textbf{while}(e)\, s} (heap_2, lcl_2)}$$

Rule for Java Method Call

$$\frac{\begin{array}{c}(heap_1, lcl_1) \xrightarrow{e \triangleright v} (heap_2, lcl_2) \\ (heap_2, lcl_2) \xrightarrow{e_1 \triangleright v_1} (heap_3, lcl_3) \\ \vdots \\ (heap_{n+1}, lcl_{n+1}) \xrightarrow{e_n \triangleright v_n} (heap_{n+2}, lcl_{n+2}) \\ (heap_{n+2}, mlcl) \xrightarrow{body} (heap_{n+3}, mlcl')\end{array}}{(heap_1, lcl_1) \xrightarrow{e.m(e_1,\dots,e_n) \triangleright mlcl'(\backslash result)} (heap_{n+3}, lcl_{n+2})},$$

where $body$ is the body of the method m in the object $heap_{n+2}(v)$, and $mlcl = \{this \mapsto v, param_1 \mapsto v_1, \dots, param_n \mapsto v_n\}$ where $param_1, \dots, param_n$ are the names of the parameters of m

The value $\backslash result$ is written by the return statement using the rule

$$\frac{(heap_1, lcl_1) \xrightarrow{e \triangleright v} (heap_2, lcl_2)}{(heap_1, lcl_1) \xrightarrow{\textbf{return}\ e} (heap_2, lcl_2 \oplus \{\backslash result \mapsto v\})}$$

Example: Method Call

```
public class C
  public int factorial(int n) {
    if (n == 0)
       return 1;
    else
       return n * this.factorial(n-1);
} }
```

Start state: (h, l), where $l(this)$ is an object of class C

We show

$$(h, l) \xrightarrow{this.factorial(0) \triangleright 1} (h, l)$$

Example: Method Call

Let $ml = \{this \mapsto l(this), n \mapsto 0\}$. Then,

$$\dfrac{\dfrac{\begin{array}{c}(h, ml) \xrightarrow{n \rhd 0} (h, ml) \\ (h, ml) \xrightarrow{0 \rhd 0} (h, ml)\end{array}}{(h, ml) \xrightarrow{n==0 \rhd 1} (h, ml)} \quad \dfrac{(h, ml) \xrightarrow{1 \rhd 1} (h, ml)}{(h, ml) \xrightarrow{return\ 1;} (h, ml \oplus \{\backslash result \mapsto 1\})}}{(h, ml) \xrightarrow{if\ (n==0)\ return\ 1; else \ldots} (h, ml \oplus \{\backslash result \mapsto 1\})}$$

$$\dfrac{\begin{array}{l}(h, l) \xrightarrow{this \rhd l(this)} (h, l) \\ (h, l) \xrightarrow{0 \rhd 0} (h, l) \\ (h, ml) \xrightarrow{if\ (n==0)\ return\ 1; else \ldots} (h, ml)\end{array}}{(h, l) \xrightarrow{this.factorial(0) \rhd 1} (h, l)}$$