Formal Methods for Java

Lecture 3: Operational Semantics (Part 2)

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

October 30, 2012

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.

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

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

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 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:

$$\frac{(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')}{(heap, lcl) \xrightarrow{e.fld \triangleright heap'(v)(idx)} (heap', lcl')}, \text{ where } idx \text{ is the index of the field } fld \text{ in the object } heap'(v)$$

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

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

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

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

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,

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

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

Example: Method Call (general proof)

We can even show by induction that for $ml(n) \geq 0$

$$(h, ml) \xrightarrow{\mathit{if}\ (n{=}{=}0)\ \dots} (h, ml \oplus \{\backslash result \mapsto (ml(n)! \bmod 2^{32})\})$$

Proof by induction over $ml(n)$. Base case $ml(n) = 0$ was already shown. Assume $n > 0$. Induction hypothesis: if $ml'(n) = ml(n) - 1$, then

$$(h, ml') \xrightarrow{\mathit{if}\ (n{=}{=}0)\ \dots} (h, ml' \oplus \{\backslash result \mapsto ((ml(n) - 1)! \bmod 2^{32})\}) \quad (\mathrm{IH})$$

We first show that

$$(h, ml) \xrightarrow{this.factorial(n-1) \triangleright (ml(n)-1)! \bmod 2^{32}} (h, ml)$$

Proof tree:

$$\frac{(h, ml) \xrightarrow{this \triangleright ml(this)} (h, ml) \qquad \dfrac{\begin{array}{c}(h, ml) \xrightarrow{n \triangleright ml(n)} (h, ml) \\ (h, ml) \xrightarrow{1 \triangleright 1} (h, ml)\end{array}}{(h, ml) \xrightarrow{n-1 \triangleright ml(n)-1} (h, ml)} \quad (IH)}{(h, ml) \xrightarrow{this.factorial(n-1) \triangleright (ml(n)-1)! \bmod 2^{32}} (h, ml) \quad (*)}$$

Example: Method Call (general proof, cont.)

Now we can prove the return statement correct.

$$\dfrac{\dfrac{(h, ml) \xrightarrow{n \triangleright ml(n)} (h, ml) \qquad (*)}{(h, ml) \xrightarrow{n*this.factorial(n-1) \triangleright (ml(n)! \bmod 2^{32})} (h, ml)}}{(h, ml) \xrightarrow{return\ \ n*this.factorial(n-1);} (h, ml \oplus \{\backslash result \mapsto (ml(n)! \bmod 2^{32}\}) \ (**)}$$

Finally, prove the whole method body.

$$\dfrac{\dfrac{(h, ml) \xrightarrow{n \triangleright ml(n)} (h, ml) \quad (h, ml) \xrightarrow{0 \triangleright 0} (h, ml)}{(h, ml) \xrightarrow{n==0 \triangleright 0} (h, ml)} \qquad (**)}{(h, ml) \xrightarrow{if\ \ (n==0)\ \dots} (h, ml \oplus \{\backslash result \mapsto 1\})}$$

Creating Objects

Creating an Object is always combined with the call of a constructor:

$$\frac{\begin{array}{c} heap_1 = heap \cup \{na \mapsto (Type, \langle 0, \dots, 0 \rangle)\} \\ (heap_1, lcl) \xrightarrow{na.\texttt{<init>}(e_1,\dots,e_n) \triangleright v} (heap', lcl') \end{array}}{(heap, lcl) \xrightarrow{\textbf{new}\ Type(e_1,\dots,e_n) \triangleright na} (heap', lcl')}, \text{ where } na \notin \operatorname{dom} heap$$

Here `<init>` stands for the internal name of the constructor.

Exceptions and Control Flow

To handle exceptions a few changes are necessary:

- We extend the state by a flow component:
  $Q = \mathit{Flow} \times \mathit{Heap} \times \mathit{Local}$
- $\mathit{Flow} ::= \mathit{Norm} | \mathit{Ret} | \mathit{Exc}\langle\langle \mathit{Address} \rangle\rangle$

We use the identifiers $\mathit{flow} \in \mathit{Flow}$, $\mathit{heap} \in \mathit{Heap}$ and $\mathit{lcl} \in \mathit{Local}$ in the rules. Also $q \in Q$ stands for an arbitrary state.
The following axioms state that in an abnormal state statements are not executed:

$$(\mathit{flow}, \mathit{heap}, \mathit{lcl}) \xrightarrow{e \triangleright v} (\mathit{flow}, \mathit{heap}, \mathit{lcl}), \text{ where } \mathit{flow} \neq \mathit{Norm}$$

$$(\mathit{flow}, \mathit{heap}, \mathit{lcl}) \xrightarrow{s} (\mathit{flow}, \mathit{heap}, \mathit{lcl}), \text{ where } \mathit{flow} \neq \mathit{Norm}$$

Expressions With Exceptions

The previously defined rules are valid only if the left-hand-state is not an exception state.

$$\frac{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{e_1 \triangleright v_1} q \quad q \xrightarrow{e_2 \triangleright v_2} q'}{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{e_1 * e_2 \triangleright (v_1 \cdot v_2) \bmod 2^{32}} q'}$$

$$\frac{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{st_1} q \quad q \xrightarrow{st_2} q'}{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{st_1; st_2} q'}$$

$$\frac{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{e \triangleright v} q \quad q \xrightarrow{s_1} q'}{(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{\textbf{if}(e)\, s_1 \textbf{else} s_2} q'}, \text{ where } v \neq 0$$

Note that exceptions are propagated using the axiom from the last slide.

$$(\mathit{flow}, \mathit{heap}, \mathit{lcl}) \xrightarrow{e \triangleright v} (\mathit{flow}, \mathit{heap}, \mathit{lcl}), \text{ where } \mathit{flow} \neq \mathit{Norm}$$

Throwing Exceptions

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

What happens if in a field access the object is null?

$$\frac{\begin{array}{c}(Norm, heap, lcl) \xrightarrow{e \triangleright 0} q' \\ q' \xrightarrow{\textbf{throw new}\ NullPointerException()} q''\end{array}}{(Norm, heap, lcl) \xrightarrow{e.fld \triangleright v} q''}, \text{where } v \text{ is some arbitrary value}$$

Complete Rules for **throw**

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} (Norm, heap', lcl')}{(Norm, heap, lcl) \xrightarrow{\textbf{throw}\ e;} (Exc(v), heap', lcl')}, \text{ where } v \neq 0$$

$$\frac{\begin{array}{c}(Norm, heap, lcl) \xrightarrow{e \triangleright 0} q' \\ q' \xrightarrow{\textbf{throw new}\ NullPointerException()} q''\end{array}}{(Norm, heap, lcl) \xrightarrow{\textbf{throw}\ e;} q''}$$

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} (flow', heap', lcl')}{(Norm, heap, lcl) \xrightarrow{\textbf{throw}\ e;} (flow', heap', lcl')}, \text{ where } flow' \neq Norm$$

Catching Exceptions

Catching an exception:

$$\frac{\begin{array}{c}(Norm, heap, lcl) \xrightarrow{s_1} (Exc(v), heap', lcl') \\ (Norm, heap', lcl' \cup \{ex \mapsto v\}) \xrightarrow{s_2} q''\end{array}}{(Norm, heap, lcl) \xrightarrow{\textbf{try}\ s_1 \textbf{catch}(Type\ \ ex) s_2} q''}, \text{ where } v \text{ is an instance of } Type$$

No exception caught:

$$\frac{(Norm, h, l) \xrightarrow{s_1} (flow', h', l')}{(Norm, h, l) \xrightarrow{\textbf{try}\ s_1 \textbf{catch}(Type\ \ ex) s_2} (flow', h', l')},$$

where flow' is not $Exc(v)$ or v is not an instance of $Type$

Return Statement

Return statement stores the value and signals the *Ret* in flow component:

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} (Norm, heap', lcl')}{(Norm, heap, lcl) \xrightarrow{\textbf{return}\ e} (Ret, heap', lcl' \oplus \{\backslash result \mapsto v\})}$$

But evaluating *e* can also throw exception:

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} (flow, heap', lcl')}{(Norm, heap, lcl) \xrightarrow{\textbf{return}\ e} (flow, heap', lcl')}, \text{ where } flow \neq Norm$$

Method Call (Normal Case)

$$\frac{\begin{array}{c}(Norm, h_1, l_1) \xrightarrow{e \triangleright v} q_2 \\ q_2 \xrightarrow{e_1 \triangleright v_1} q_3 \\ \vdots \\ q_{n+1} \xrightarrow{e_n \triangleright v_n} (f_{n+2}, h_{n+2}, l_{n+2}) \\ (f_{n+2}, h_{n+2}, ml) \xrightarrow{body} (Ret, h_{n+3}, ml')\end{array}}{(Norm, h_1, l_1) \xrightarrow{e.m(e_1,\dots,e_n) \triangleright ml'(\backslash result)} (Norm, heap_{n+3}, l_{n+2})},$$

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

Method Call With Exception

$$\frac{\begin{array}{c}(Norm, h_1, l_1) \xrightarrow{e \triangleright v} q_2 \\ q_2 \xrightarrow{e_1 \triangleright v_1} q_3 \\ \vdots \\ q_{n+1} \xrightarrow{e_n \triangleright v_n} (f_{n+2}, h_{n+2}, l_{n+2}) \\ (f_{n+2}, h_{n+2}, ml) \xrightarrow{body} (Exc(v_e), h_{n+3}, ml')\end{array}}{(Norm, h_1, l_1) \xrightarrow{e.m(e_1,\dots,e_n) \triangleright ml'(\backslash result)} (Exc(v_e), heap_{n+3}, l_{n+2})},$$

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