Formal Methods for Java

Lecture 4: Semantics of JML

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

May 8, 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 or expressions.
- $q \xrightarrow{e \triangleright v} q'$ means that in state q the expression e is evaluated to v and the side-effects change the state to q'.
- $q \xrightarrow{st} q'$ means that in state q the statement st is executable and changes the state to q'.

Creating Objects

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

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

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:
  $Q = Flow \times Heap \times Local$
- $Flow ::= Norm | Ret | Exc\langle\langle Address \rangle\rangle$

We use the identifiers $flow \in Flow$, $heap \in Heap$ and $lcl \in Local$ 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:

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

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

Expressions With Exceptions

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

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

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

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

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

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

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?

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

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

No exception caught:

$$\frac{(Norm, h, l) \xrightarrow{s_1} (flow', h', l')}{(Norm, h, l) \xrightarrow{\textbf{try}\ s_1\ \textbf{catch}(Type\ \ ex) s_2} (flow', h', l')}, \text{ where flow' is not } Exc(v) \text{ or } v \text{ is not an instance of } Type$$

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)

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

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

Method Call With Exception

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

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