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

\[
heap_1 = heap \cup \{ na \mapsto (\text{Type}, \langle 0, \ldots, 0 \rangle) \}
\]

\[
\frac{(heap_1, lcl)}{(heap, lcl)} \xrightarrow{\text{na.<init>(e_1, \ldots, e_n)\triangleright v}} (heap', lcl')
\]

\[
\frac{(heap, lcl)}{(heap, lcl)} \xrightarrow{\text{new Type(e_1, \ldots, e_n)\triangleright na}} (heap', lcl')
\]

, where \( na \notin \text{dom heap} \)

Here <init> stands for the internal name of the constructor.
To handle exceptions a few changes are necessary:

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

We use the identifiers \( \text{flow} \in \text{Flow}, \text{heap} \in \text{Heap} \) and \( \text{lcl} \in \text{Local} \) 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:

\[
(flow, heap, lcl) \xrightarrow{e\Downarrow v} (flow, heap, lcl), \text{ where } flow \neq \text{Norm}
\]

\[
(flow, heap, lcl) \xrightarrow{s} (flow, heap, lcl), \text{ where } flow \neq \text{Norm}
\]
Expressions With Exceptions

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

\[
\begin{align*}
(Norm, heap, lcl) & \xrightarrow{e_1 \triangleright v_1} q \quad q \xrightarrow{e_2 \triangleright v_2} q' \\
(Norm, heap, lcl) & \xrightarrow{e_1 \ast e_2 \triangleright (v_1 \cdot v_2) \mod 2^{32}} q'
\end{align*}
\]

\[
\begin{align*}
(Norm, heap, lcl) & \xrightarrow{st_1} q \quad q \xrightarrow{st_2} q' \\
(Norm, heap, lcl) & \xrightarrow{st_1 ; st_2} q'
\end{align*}
\]

\[
\begin{align*}
(Norm, heap, lcl) & \xrightarrow{e \triangleright v} q \quad q \xrightarrow{s_1} q' \\
(Norm, heap, lcl) & \xrightarrow{\text{if}(e) s_1 \text{else} s_2} q', \text{ where } v \neq 0
\end{align*}
\]

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

\[
(Norm, heap, lcl) \xrightarrow{e \triangleright v} (flow, heap, lcl), \text{ where } flow \neq Norm
\]
Throwing Exceptions

\[
\begin{align*}
(Norm, heap, lcl) \xrightarrow{e \triangleright v} (Norm, heap', lcl') \\
(Norm, heap, lcl) \xrightarrow{\text{throw } e;} (Exc(v), heap', lcl')
\end{align*}
\]

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

\[
\begin{align*}
(Norm, heap, lcl) \xrightarrow{e \triangleright 0} q' \\
q' \xrightarrow{\text{throw new NullPointerException()}} q'' \\
(Norm, heap, lcl) \xrightarrow{e \cdot \text{fld} \triangleright v} q''
\end{align*}
\]

, where \( v \) is some arbitrary value
Complete Rules for `throw`

\[
\begin{align*}
&\frac{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{e \ni v} (\text{Norm}, \text{heap}', \text{lcl}')}{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{\text{throw } e;} (\text{Exc}(\nu), \text{heap}', \text{lcl}')}} \quad \text{where } \nu \neq 0 \\
&\frac{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{e \ni 0} q'}{q' \xrightarrow{\text{throw new NullPointerException()} } q''} \\
&\frac{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{\text{throw } e;} q''}{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{\text{throw } e;} q''} \\
&\frac{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{e \ni v} (\text{flow}', \text{heap}', \text{lcl}')}{(\text{Norm}, \text{heap}, \text{lcl}) \xrightarrow{\text{throw } e;} (\text{flow}', \text{heap}', \text{lcl}')} \quad \text{where } \text{flow}' \neq \text{Norm}
\end{align*}
\]
Catching Exceptions

Catching an exception:

\[
(Norm, heap, lcl) \overset{s_1}{\rightarrow} (Exc(v), heap', lcl')
\]

\[
(Norm, heap', lcl' \cup \{ex \mapsto v\}) \overset{s_2}{\rightarrow} q''
\]

where \(v\) is an instance of Type

\[
(Norm, heap, lcl) \overset{\text{try}_{s_1} \text{catch}(Type \; ex)_{s_2}}{\rightarrow} q''
\]

No exception caught:

\[
(Norm, h, l) \overset{s_1}{\rightarrow} (flow', h', l')
\]

\[
(Norm, h, l) \overset{\text{try}_{s_1} \text{catch}(Type \; ex)_{s_2}}{\rightarrow} (flow', h', l')
\]

where \(flow'\) is not \(Exc(v)\) or \(v\) is not an instance of Type
Return Statement

Return statement stores the value and signals the \textit{Ret} in flow component:

\[
\begin{align*}
(Norm, \heap, \lcl) & \xrightarrow{\text{e} \triangleright v} (Norm, \heap', \lcl') \\
(Norm, \heap, \lcl) & \xrightarrow{\text{return e}} (\text{Ret}, \heap', \lcl' \oplus \{\text{result} \mapsto v\})
\end{align*}
\]

But evaluating \(e\) can also throw exception:

\[
\begin{align*}
(Norm, \heap, \lcl) & \xrightarrow{\text{e} \triangleright v} (\text{flow}, \heap', \lcl') \\
(Norm, \heap, \lcl) & \xrightarrow{\text{return e}} (\text{flow}, \heap', \lcl')
\end{align*}
\]

where \(\text{flow} \neq \text{Norm}\).
Method Call (Normal Case)

\[
\begin{align*}
(Norm, h_1, l_1) & \xrightarrow{e \triangleright v} q_2 \\
q_2 & \xrightarrow{e_1 \triangleright v_1} q_3 \\
& \vdots \\
q_{n+1} & \xrightarrow{e_n \triangleright v_n} (f_{n+2}, h_{n+2}, l_{n+2}) \\
(f_{n+2}, h_{n+2}, ml) & \xrightarrow{body} (Ret, h_{n+3}, ml') \\
(Norm, h_1, l_1) & \xrightarrow{e.m(e_1, \ldots, e_n) \triangleright ml'(\text{\textbackslash result})} (Norm, heap_{n+3}, l_{n+2})
\end{align*}
\]

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

\[
\begin{align*}
(Norm, h_1, l_1) & \xrightarrow{e \triangleright v} q_2 \\
q_2 & \xrightarrow{e_1 \triangleright v_1} q_3 \\
& \quad \vdots \\
q_{n+1} & \xrightarrow{e_n \triangleright v_n} (f_{n+2}, h_{n+2}, l_{n+2}) \\
(f_{n+2}, h_{n+2}, ml) & \xrightarrow{\text{body}} (\text{Exc}(v_e), h_{n+3}, ml') \\
(Norm, h_1, l_1) & \xrightarrow{e.m(e_1, \ldots, e_n) \triangleright ml'(\\text{\result})} (\text{Exc}(v_e), \text{heap}_{n+3}, l_{n+2})
\end{align*}
\]

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