Formal Methods for Java
Lecture 2: Operational Semantics

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

April 28, 2017
The Java Language Specification (JLS) SE 8 edition gives semantics for Java

- The document has 788 pages.
- 150 pages to define semantics of expression.
- 31 pages to define semantics of method invocation.

Semantics are only defined by prosa text.

**How can we give the semantics formally?**

Need a mathematical model for computations.
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.
Example: State of a Java Program

What is the state after executing this code?

```java
List myList = new LinkedList();
myList.add(new Integer(1));
```

<table>
<thead>
<tr>
<th>heap</th>
<th>lcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>7: LinkedList 1 8 1</td>
<td>myList: 7</td>
</tr>
<tr>
<td>8: LinkedList.Entry 0 9 9</td>
<td></td>
</tr>
<tr>
<td>9: LinkedList.Entry 10 8 8</td>
<td></td>
</tr>
<tr>
<td>10: Integer 1</td>
<td></td>
</tr>
</tbody>
</table>
The state of a Java program gives valuations to local and global (heap) variables.

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

A state is denoted as \((\text{heap}, \text{lcl})\), where \(\text{heap} : \text{Heap}\) and \(\text{lcl} : \text{Local}\).
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.
Example: Actions of a Java Program

Post-increment expression:

\[(heap, lcl \cup \{x \mapsto 5\}) \xrightarrow{x++ \uparrow 5} (heap, lcl \cup \{x \mapsto 6\})\]

Pre-increment expression:

\[(heap, lcl \cup \{x \mapsto 5\}) \xrightarrow{++x \uparrow 6} (heap, lcl \cup \{x \mapsto 6\})\]

Assignment expression:

\[(heap, lcl \cup \{x \mapsto 5\}) \xrightarrow{x=x*2 \uparrow 10} (heap, lcl \cup \{x \mapsto 10\})\]

Assignment statement:

\[(heap, lcl \cup \{x \mapsto 5\}) \xrightarrow{x=x*2;} (heap, lcl \cup \{x \mapsto 10\})\]
The last slide listed some examples for transitions. We now define rules when a transition is valid.

**Definition (Inference Rules)**

A rule of inference

\[
\frac{F_1 \ldots F_n}{G}
\]

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 (1)

axiom for evaluating local variables:

\[(heap, lcl) \xrightarrow{x \mapsto lcl(x)} (heap, lcl)\]

rule for field access:

\[(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')\]

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

where \(idx\) is the index, of the field \(fld\) in the object \(heap'(v)\)

rule for assignment to local:

\[(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')\]

\[(heap, lcl) \xrightarrow{x = e \triangleright v} (heap', lcl' \oplus \{x \mapsto v\})\]
axiom for evaluating a constant expression $c$:

$$
(\text{heap}, \text{lcl}) \xrightarrow{c \cdot c} (\text{heap}, \text{lcl})
$$

rule for multiplication (similar for other binary operators)

$$
\begin{align*}
(\text{heap}_1, \text{lcl}_1) & \xrightarrow{e_1 \cdot v_1} (\text{heap}_2, \text{lcl}_2) \\
(\text{heap}_2, \text{lcl}_2) & \xrightarrow{e_2 \cdot v_2} (\text{heap}_3, \text{lcl}_3) \\
\end{align*}
$$

$$(\text{heap}_1, \text{lcl}_1) \xrightarrow{e_1 \cdot e_2 \cdot (v_1 \cdot v_2) \mod 2^{32}} (\text{heap}_3, \text{lcl}_3)$$
A derivation for $x = x \times 2$

\[
\begin{align*}
(\text{heap}, lcl \cup \{x \mapsto 5\}) & \xrightarrow{x \mapsto 5} (\text{heap}, lcl \cup \{x \mapsto 5\}) \\
(\text{heap}, lcl \cup \{x \mapsto 5\}) & \xrightarrow{2 \times 2} (\text{heap}, lcl \cup \{x \mapsto 5\}) \\
(\text{heap}, lcl \cup \{x \mapsto 5\}) & \xrightarrow{x \times 2 \mapsto 10} (\text{heap}, lcl \cup \{x \mapsto 10\})
\end{align*}
\]
expression statement (assignment or method call):

\[
(\text{heap}, \text{lcl}) \xrightarrow{e \triangleright v} (\text{heap}', \text{lcl}') \\
(\text{heap}, \text{lcl}) \xrightarrow{e;} (\text{heap}', \text{lcl}')
\]

sequence of statements:

\[
(\text{heap}_1, \text{lcl}_1) \xrightarrow{s_1} (\text{heap}_2, \text{lcl}_2) \quad (\text{heap}_2, \text{lcl}_2) \xrightarrow{s_2} (\text{heap}_3, \text{lcl}_3) \\
(\text{heap}_1, \text{lcl}_1) \xrightarrow{s_1 \cdot s_2} (\text{heap}_3, \text{lcl}_3)
\]
if statement:

\[
\begin{align*}
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{e} > v} (\text{heap}_2, \text{lcl}_2) & \quad (\text{heap}_2, \text{lcl}_2) \xrightarrow{\text{bl}_1} (\text{heap}_3, \text{lcl}_3) \\
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{if}(\text{e}) \text{ bl}_1 \text{ else} \text{bl}_2} (\text{heap}_3, \text{lcl}_3)
\end{align*}
\]

, where \( v \neq 0 \)

\[
\begin{align*}
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{e} > v} (\text{heap}_2, \text{lcl}_2) & \quad (\text{heap}_2, \text{lcl}_2) \xrightarrow{\text{bl}_2} (\text{heap}_3, \text{lcl}_3) \\
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{if}(\text{e}) \text{ bl}_1 \text{ else} \text{bl}_2} (\text{heap}_3, \text{lcl}_3)
\end{align*}
\]

, where \( v = 0 \)