Formal Methods for Java

Lecture 5: Semantics of JML

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

May 10, 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 (flow, 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'.

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

The following axioms state that in an abnormal state statements are not executed:

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

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

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 $ml = \{this \mapsto v, param_1 \mapsto v_1, \dots, param_n \mapsto v_n\}$

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: $(Norm, h, l)$, where $l(this)$ is an object of class `C`

We show

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

Example: Method Call

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

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

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

Example: Method Call (general proof)

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

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

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

$$(N, h, ml') \xrightarrow{\mathit{if}\ \ (n{=}{=}0)\ \ \ldots} (Ret, h, ml' \oplus \{\backslash result \mapsto ((ml(n) - 1)! \bmod 2^{32})\}) \qquad \text{(IH)}$$

We first show that

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

Proof tree:

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

Example: Method Call (general proof, cont.)

Now we can prove the return statement correct.

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

Finally, prove the whole method body.

$$\dfrac{\dfrac{(N, h, ml) \xrightarrow{n \rhd ml(n)} (N, h, ml) \quad (N, h, ml) \xrightarrow{0 \rhd 0} (N, h, ml)}{(N, h, ml) \xrightarrow{n == 0 \rhd 0} (N, h, ml)} \quad (**)}{(N, h, ml) \xrightarrow{if\ \ (n == 0)\ \ \ldots} (Ret, h, ml \oplus \{\backslash result \mapsto 1\})}$$

Semantics of Specification

```
/*@ requires x >= 0;
  @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
  @*/
public static int isqrt(int x) {
  body
}
```

Whenever the method is called with values that satisfy the requires-formula
and the method terminates normally
then the ensures-formula holds.

For all $heap, heap', lcl, lcl'$ if $lcl(x) \geq 0$
and $(Norm, heap, lcl) \xrightarrow{body} (Ret, heap', lcl')$,
then $lcl'(\backslash result) \leq Math.sqrt(lcl(x)) < lcl'(\backslash result) + 1$ holds.

Hoare Triples

```
/*@ requires x >= 0;
  @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
  @*/
public static int isqrt(int x) {
  body
}
```

The JML code above states partial correctness of the Hoare triple

$$\{x \geq 0\}$$
$$\mathit{body}$$
$$\{\backslash \mathit{result} \leq \mathit{Math.sqrt}(x) < \backslash \mathit{result} + 1\}$$

It also states total correctness, as we will see later.

Post condition and input parameters

Is the following implementation correct?

```
/*@ requires x >= 0;
  @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
  @*/
public static int isqrt(int x) {
  x = 0;
  return 0;
}
```

No, because JML always evaluates input parameters always in the pre-state!

For all $heap, heap', lcl, lcl'$ if $lcl(x) \geq 0$
and $(Norm, heap, lcl) \xrightarrow{body} (Ret, heap', lcl')$,
then $lcl'(\backslash result) \leq Math.sqrt(lcl(x)) < lcl'(\backslash result) + 1$ holds.

What About Exceptions?

```
/*@ requires true;
  @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
  @ signals (IllegalArgumentException) x < 0;
  @ signals_only IllegalArgumentException;
  @*/
public static int isqrt(int x) {
  body
}
```

The `signals_only` specification denotes that for all transitions

$$(Norm, heap, lcl) \xrightarrow{body} (Exc(v), heap', lcl')$$

where *lcl* satisfies the precondition and v is an Exception, v must be of type IllegalArgumentException.
The `signals` specification denotes that in that case *lcl* must satisfy $x < 0$.

The code is still allowed to throw an Error like a OutOfMemoryError or a ClassNotFoundError.

Side-Effects

A method can change the heap in an unpredictable way.
The assignable clause restricts changes:

```
/*@ requires x >= 0;
  @ assignable \nothing;
  @ ensures \result <= Math.sqrt(x) && Math.sqrt(x) < \result + 1;
  @*/
public static int isqrt(int x) {
  body
}
```

For all executions of the method,

$$(\mathit{Norm}, \mathit{heap}, \mathit{lcl}) \xrightarrow{\mathit{body}} (\mathit{Ret}, \mathit{heap}', \mathit{lcl}'),$$

if $\mathit{lcl}(x) >= 0$ then the formula

$$\mathit{lcl}'(\backslash \mathit{result}) <= \mathit{Math.sqrt}(\mathit{lcl}(x)) < \mathit{lcl}'(\backslash \mathit{result} + 1)$$

holds and $\mathit{heap} \subseteq \mathit{heap}'$.

What is the meaning of a formula

A formula like x `>= 0` is a Boolean Java expression. It can be evaluated with the operational semantics.
`x >= 0` holds in state $(heap, lcl)$, iff

$$(Norm, heap, lcl) \xrightarrow{\texttt{x >= 0} \triangleright 1} (Norm, heap', lcl')$$

An assertion may not have side-effects; it may create new objects, though, i.e., $heap \subseteq heap'$ and $lcl = lcl'$.
For the ensures formula both the pre-state and the post-state are necessary to evaluate the formula.