Operational Semantics for Java

Idea: define transition system for Java

**Definition (Transition System)**

A transition system \( \mathcal{TS} \) is a structure \( \mathcal{TS} = (Q, \text{Act}, \rightarrow) \), where

- \( Q \) is a set of states,
- \( \text{Act} \) a set of actions,
- \( \rightarrow \subseteq Q \times \text{Act} \times Q \) the transition relation.

- \( Q \) reflects the current dynamic state (flow, heap and local variables).
- \( \text{Act} \) 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 \( q' \).
- \( q \xrightarrow{\text{st}} q' \) means that in state \( q \) the statement \( \text{st} \) is executable and changes the state to \( q' \).
\[ Q = Flow \times Heap \times Local \]

\[ Flow ::= Norm | Ret | Exc \langle \langle \text{Address} \rangle \rangle \]

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

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

\[ (flow, heap, lcl) \xrightarrow{s} (flow, heap, lcl), \text{ where } flow \neq Norm \]
Return Statement

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

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

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

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

where \( flow \neq Norm \)
Method Call (Normal Case)

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

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 \(ml = \{ \text{this} \mapsto v, param_1 \mapsto v_1, \ldots, param_n \mapsto v_n \}\)
(\text{Norm}, h_1, l_1) \xrightarrow{e^>v} q_2 \quad q_2 \xrightarrow{e_1^>v_1} q_3 \\
\vdots \quad q_{n+1} \xrightarrow{e_{n+1}^>v_{n+1}} (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)^>ml'(\text{\result})} (\text{Exc}(v_e), \text{heap}_{n+3}, l_{n+2})

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 \(ml = \{\text{this} \mapsto v, param_1 \mapsto v_1, \ldots, param_n \mapsto v_n\}\).
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

\[
(Norm, h, l) \xrightarrow{this.factorial(0)\geq 1} (Norm, h, l)
\]
Let $ml = \{this \mapsto l(this), n \mapsto 0\}$. Then,

$$
\begin{align*}
(N, h, ml) &\xrightarrow{n>0} (N, h, ml) \\
(N, h, ml) &\xrightarrow{0>0} (N, h, ml) \\
(N, h, ml) &\xrightarrow{n==0>1} (N, h, ml) \\
(N, h, ml) &\xrightarrow{\text{return 1}; \text{else} \ldots} (Ret, h, ml \oplus \{\text{result} \mapsto 1\}) \\
(N, h, ml) &\xrightarrow{\text{this.factorial}(0)>1} (N, h, l)
\end{align*}
$$
Example: Method Call (general proof)

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

$$(N, h, m\ell) \xrightarrow{\text{if } (n=0)} (\text{Ret}, h, m\ell \oplus \{\text{result} \mapsto (m\ell(n)! \mod 2^{32})\})$$

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

$$(N, h, m\ell') \xrightarrow{\text{if } (n=0)} (\text{Ret}, h, m\ell' \oplus \{\text{result} \mapsto ((m\ell(n) - 1)! \mod 2^{32})\}) \quad \text{(IH)}$$

We first show that

$$(N, h, m\ell) \xrightarrow{\text{this.factorial}(n-1)! \mod 2^{32}} (N, h, m\ell)$$

Proof tree:
Now we can prove the return statement correct.

\[
\begin{align*}
(N, h, ml) & \xrightarrow{n > ml(n)} (N, h, ml) \quad (*) \\
& \xrightarrow{n * \text{this.factorial}(n-1) > (ml(n)! \mod 2^{32})} (N, h, ml) \\
(N, h, ml) & \xrightarrow{\text{return } n * \text{this.factorial}(n-1);} (\text{Ret}, h, ml \oplus \{ \text{result} \mapsto (ml(n)! \mod 2^{32}) \}) \quad (**) \\
\end{align*}
\]

Finally, prove the whole method body.

\[
\begin{align*}
(N, h, ml) & \xrightarrow{n > ml(n)} (N, h, ml) \quad (N, h, ml) \xrightarrow{0 > 0} (N, h, ml) \quad (***) \\
& \xrightarrow{n == 0 > 0} (N, h, ml) \\
(N, h, ml) & \xrightarrow{\text{if } (n == 0) \ldots} (\text{Ret}, h, ml \oplus \{ \text{result} \mapsto 1 \}) \\
\end{align*}
\]
Semantics of Specification

```java
/*@ 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) ≥ 0` and `(Norm, heap, lcl) \xrightarrow{body} (Ret, heap', lcl')`, then `lcl'(`\result`) ≤ Math.sqrt(lcl(x)) < lcl'(`\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\}

body

\{\result \leq Math.sqrt(x) < \result + 1\}

It also states total correctness, as we will see later.
Is the following implementation correct?

```java
/*@ 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 $lcl'(\text{result}) \leq Math.sqrt(lcl(x)) < lcl'(\text{result}) + 1$ holds.
What About Exceptions?

```java
/*@
  requires true;
  ensures result \leq 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 `ClassNotFoundException`.
Side-Effects

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

```java
/*@ 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,

\[(\text{Norm}, \text{heap}, lcl) \xrightarrow{\text{body}} (\text{Ret}, \text{heap}', lcl')\],

if \(lcl(x) \geq 0\) then the formula

\[lcl'(\\text{result}) \leq Math.sqrt(lcl(x)) < lcl'(\text{result} + 1)\]

holds and \(\text{heap} \subseteq \text{heap}'\).
What is the meaning of a formula

A formula like $x \geq 0$ is a Boolean Java expression. It can be evaluated with the operational semantics.

$x \geq 0$ holds in state $(heap, lcl)$, iff

$$ (Norm, heap, lcl) \xrightarrow{x \geq 0 \downarrow} (Norm, heap', lcl') $$

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.