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.
The state of a Java program gives valuations 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 \((heap, lcl)\), where \(heap : \text{Heap}\) and \(lcl : \text{Local}\).
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.
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

axiom for evaluating local variables:

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

axiom for evaluating constants:

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

rule for field access:

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

where \(idx\) is the index of the field \(fld\) in the object \(heap'(v)\)
Rules for Assignment Expressions

rule for assignment to local:

\[
\frac{(heap, lcl) \xrightarrow{e \triangleright v} (heap', lcl')}{(heap, lcl) \xrightarrow{x = e \triangleright v} (heap', lcl' \oplus \{ x \mapsto v \})}
\]

rule for assignment to field:

\[
\frac{(heap_1, lcl_1) \xrightarrow{e_1 \triangleright v_1} (heap_2, lcl_2) \quad (heap_2, lcl_2) \xrightarrow{e_2 \triangleright v_2} (heap_3, lcl_3)}{(heap_1, lcl_1) \xrightarrow{e_1.fld = e_2 \triangleright v_2} (heap_4, lcl_3)}
\]

where \( heap_4 = heap_3 \oplus \{ (v_1, idx) \mapsto v_2 \} \) and \( idx \) is the index of the field \( fld \) in the object at \( heap_3(v_1) \).
Rules for Java Statements

expression statement (assignment or method call):

\[
(\text{heap}, \text{lcl}) \xrightarrow{e>\nu} (\text{heap}', \text{lcl}') \\
(\text{heap}, \text{lcl}) \xrightarrow{e;i} (\text{heap}', \text{lcl}')
\]

sequence of statements:

\[
(\text{heap}_1, \text{lcl}_1) \xrightarrow{s_1} (\text{heap}_2, \text{lcl}_2) \quad (\text{heap}_2, \text{lcl}_2) \xrightarrow{s_2} (\text{heap}_3, \text{lcl}_3) \\
(\text{heap}_1, \text{lcl}_1) \xrightarrow{s_1 \cdot s_2} (\text{heap}_3, \text{lcl}_3)
\]
Rules for Java Statements

if statement:

\[
\begin{align*}
(\text{heap}_1, \text{lcl}_1) & \xrightarrow{\text{if}(e) \ s_1 \ \text{else} \ s_2} (\text{heap}_3, \text{lcl}_3), \text{ where } v \neq 0 \\
(\text{heap}_1, \text{lcl}_1) & \xrightarrow{\text{if}(e) \ s_1} (\text{heap}_3, \text{lcl}_3)
\end{align*}
\]

while statement:

\[
\begin{align*}
(\text{heap}_1, \text{lcl}_1) & \xrightarrow{\text{if}(e)\{s \ \text{while}(e) \ s\}} (\text{heap}_2, \text{lcl}_2) \\
(\text{heap}_1, \text{lcl}_1) & \xrightarrow{\text{while}(e) \ s} (\text{heap}_2, \text{lcl}_2)
\end{align*}
\]
Rule for Java Method Call

\[
(\text{heap}_1, \text{lcl}_1) \xrightarrow{e \triangleright v} (\text{heap}_2, \text{lcl}_2)
\]
\[
(\text{heap}_2, \text{lcl}_2) \xrightarrow{e_1 \triangleright v_1} (\text{heap}_3, \text{lcl}_3)
\]
\[
\vdots
\]
\[
(\text{heap}_{n+1}, \text{lcl}_{n+1}) \xrightarrow{e_n \triangleright v_n} (\text{heap}_{n+2}, \text{lcl}_{n+2})
\]
\[
(\text{heap}_{n+2}, \text{mlcl}) \xrightarrow{\text{body}} (\text{heap}_{n+3}, \text{mlcl}')
\]
\[
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{e.m}(e_1, \ldots, e_n) \triangleright \text{mlcl}'(\text{\result})} (\text{heap}_{n+3}, \text{lcl}_{n+2})
\]

where \text{body} is the body of the method \text{m} in the object \text{heap}_{n+2}(\text{v}), and \text{mlcl} = \{\text{this} \mapsto \text{v}, \text{param}_1 \mapsto \text{v}_1, \ldots, \text{param}_n \mapsto \text{v}_n\} where \text{param}_1, \ldots, \text{param}_n are the names of the parameters of \text{m}.

The value \text{\result} is written by the return statement using the rule
\[
(\text{heap}_1, \text{lcl}_1) \xrightarrow{\text{e.m}(e_1, \ldots, e_n) \triangleright \text{mlcl}'(\text{\result})} (\text{heap}_{n+3}, \text{lcl}_{n+2})
\]
Example: Method Call

```java
public class C {
    public int factorial(int n) {
        if (n == 0)
            return 1;
        else
            return n * this.factorial(n-1);
    }
}
```

Start state: \((h, l)\), where \(l(this)\) is an object of class \(C\)

We show

\[
(h, l) \xrightarrow{this.factorial(0)>1} (h, l)
\]
Example: Method Call

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

\[
\begin{align*}
(h, ml) & \xrightarrow{n>0} (h, ml) \\
(h, ml) & \xrightarrow{0>0} (h, ml) \\
(h, ml) & \xrightarrow{n==0>1} (h, ml) \\
(h, ml) & \xrightarrow{\text{return } 1;} (h, ml \oplus \{ \backslash \text{result } \mapsto 1 \}) \\
(h, l) & \xrightarrow{\text{this } \mapsto l(this)} (h, l) \\
(h, l) & \xrightarrow{0>0} (h, l) \\
(h, ml) & \xrightarrow{\text{if } (n==0) \text{ return } 1; \text{else...}} (h, ml \oplus \{ \backslash \text{result } \mapsto 1 \}) \\
(h, l) & \xrightarrow{\text{this.factorial}(0)>1} (h, l) \\
\end{align*}
\]