Operational Semantics II

A4M36TPJ, 2013/2014

Example Language EXPR - Syntax

$$\begin{aligned} Expr ::= \; & Num \mid \\ & \triangle Expr \mid \\ & Expr \odot Expr \end{aligned}$$

Num is a predefined set of integer numbers (a.k.a. Z).

Small-Step Operational Semantics (SOS)

- We need to define a transition relation:

$$\Rightarrow \in Expr \times Expr$$

- The following expression means that the transition e to e' is done in one step.

$$e \Rightarrow e'$$

SOS Rules

$$e, e_1, e_2, e' \in Expr \qquad n, n' \in Num$$

$$(\text{triangle e})\ \frac{e \Rightarrow e'}{\triangle e \Rightarrow \triangle e'}$$

$$(\text{triangle n})\ \frac{}{\triangle n \Rightarrow -n}$$

$$(\text{odot } e_1)\ \frac{e_1 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e' \odot e_2}$$

$$(\text{odot n})\ \frac{}{n \odot n' \Rightarrow n + n'}$$

$$(\text{odot } e_2)\ \frac{e_2 \Rightarrow e'}{e_1 \odot e_2 \Rightarrow e_1 \odot e'}$$

Proof Tree for SOS

$$
\text{(triangle e)}\ \dfrac{\text{(odot } e_1\text{)}\ \dfrac{\text{triangle n}\ \dfrac{}{\triangle 15 \Rightarrow -15}}{(\triangle 15) \odot (\triangle 24) \Rightarrow -15 \odot (\triangle 24)}}{\triangle((\triangle 15) \odot (\triangle 24)) \Rightarrow \triangle(-15 \odot (\triangle 24))}
$$

$$
\text{(triangle e)}\ \dfrac{\text{(odot } e_2\text{)}\ \dfrac{\text{triangle n}\ \dfrac{}{\triangle 24 \Rightarrow -24}}{(\triangle 15) \odot (\triangle 24) \Rightarrow (\triangle 15) \odot -24}}{\triangle((\triangle 15) \odot (\triangle 24)) \Rightarrow \triangle((\triangle 15) \odot -24)}
$$

SOS Formal Definition

$$S = < CF, \Rightarrow, FC, IF, OF >$$

- CF is the **domain of configurations**. The domain variable cf ranges over them.
- =>, the **transition relation**, cf => cf' is a one step transition from the configuration cf to cf'.
- FC is the set of **final configurations.**
- IF is the input function (Prog x Inputs) -> CF.
- OF is the output function FC -> Answer.

Big-Step Operational Semantics (BOS)

- Different Approach
- Program is evaluated in one step.
- We need to define the transition relation:

$$\Longrightarrow \in Expr \times Num$$

BOS Rules for EXPR

$$(\text{Num})\,\frac{}{n \Longrightarrow n}$$

$$(\text{triangle e})\,\frac{e \Longrightarrow n}{\triangle e \Longrightarrow -n}$$

$$(\text{odot e})\,\frac{e_1 \Longrightarrow n_1 \quad e_2 \Longrightarrow n_2}{e_1 \odot e_2 \Longrightarrow n_1 + n_2}$$

Proof Tree for BOS

$$
\text{(triangle e)}\ \dfrac{\text{(odot e)}\ \dfrac{\text{(triangle e)}\ \dfrac{\text{(Num)}\ \dfrac{}{15 \Longrightarrow 15 \quad 24 \Longrightarrow 24}}{\triangle 15 \Longrightarrow -15 \quad \triangle 24 \Longrightarrow -24}}{(\triangle 15) \odot (\triangle 24) \Longrightarrow -15 + -24}}{\triangle((\triangle 15) \odot (\triangle 24)) \Longrightarrow -(-15 + -24)}
$$

Example SOS Language I++

- Simple imperative language
- It supports assignment, conditional, iteration and sequence of commands.

I++ Syntax

$$
\begin{aligned}
Command ::=& Command; Command\\
&|skip\\
&|if\ \ Boolean\ \ then\ \ Command\ \ else\ \ Command\ \ fi\\
&|x := Num\\
&|while\ \ Boolean\ \ do\ \ Command\ \ od\\
Boolean ::=& true|false\\
&|Boolean\ \ and\ \ Boolean\\
&|not\ \ Boolean\\
&|Num = Num\\
Num ::=& Number\\
&|Num + Num\\
&|x
\end{aligned}
$$

Legal Program in I++

```
x:=0;
while x < 5 do
    x:=x+1;
od;
```

SOS for I++

- We need to specify components of the formal definition.
- At first we need to specify what is a configuration.

Configuration of I++

- We have to remember the state of I++ program.
- There is only one variable **x.**
- The configuration of the program I++ is a pair (Program, Num).

Configuration of I++

- More formally: $cf \in Program \times Num$

$$IF(p : Program, \emptyset) = (p, 0)$$

Transition Relation =>

$$\text{(assign)}\ \frac{meaning(n, m) = n'}{(x := n, m) \Rightarrow (skip, n')}$$

$$\text{(if true)}\ \frac{meaning(B, m) = true}{(if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2\ \ fi, m) \Rightarrow (C_1, m)}$$

$$\text{(if false)}\ \frac{meaning(B, m) = false}{(if\ \ B\ \ then\ \ C_1\ \ else\ \ C_2\ \ fi, m) \Rightarrow (C_2, m)}$$

Transition Relation =>

$$\text{(while)}\ \frac{}{(while\ \ B\ \ do\ \ C\ \ od, m) \Rightarrow (if\ \ B\ \ then\ \ C; while\ \ B\ \ do\ \ C\ \ od\ \ else\ \ skip\ \ fi, m)}$$

$$\text{(skip;)}\ \frac{}{(skip; C, m) \Rightarrow (C, m)}$$

$$\text{(skip)}\ \frac{}{(skip, m) \Rightarrow (skip, m)}$$

$$\text{(;)}\ \frac{(C_1, m) \Rightarrow (C_1', m')}{(C_1; C_2, m) \Rightarrow (C_1'; C_2, m')}$$

You can think about...

- Define function *meaning* used in previous definitions.
- How can be language extended to support more variables?