COSE312: Compilers

Lecture 12 — Semantic Analysis (2)

Hakjoo Oh
2017 Spring

Operational Semantics

Operational semantics is concerned about how to execute programs and not merely what the execution results are.

- *Big-step operational semantics* describes how the overall results of executions are obtained.
- *Small-step operational semantics* describes how the individual steps of the computations take place.

In both kinds, the semantics is specified by a transition system $(\mathbb{S}, \rightarrow)$ where $\mathbb{S}$ is the set of states with two types:

- $\langle S, s \rangle$: a nonterminal state (i.e. the statement S is to be executed from the state s)
- s: a terminal state

The transition relation describes how the execution takes place. The difference between the two approaches are in the definitions of transition relation.

Big-step Operational Semantics

The transition relation specifies the relationship between the initial state and the final state:

$$\langle S, s\rangle \to s'$$

Transition relation is defined with inference rules of the form: A rule has the general form

$$\frac{\langle S_1, s_1\rangle \to s_1', \ldots, \langle S_n, s_n\rangle \to s_n'}{\langle S, s\rangle \to s'} \text{ if } \cdots$$

- $S_1, \ldots, S_n$ are statements that constitute S.
- A rule has a number of premises and one conclusion.
- A rule may also have a number of conditions that have to be fulfilled whenever the rule is applied.
- Rules without premises are called axioms.

Big-step Operational Semantics for **While**

$$\frac{}{\langle x := a, s\rangle \rightarrow s[x \mapsto \mathcal{A}[\![a]\!](s)]}$$

$$\frac{}{\langle \texttt{skip}, s\rangle \rightarrow s}$$

$$\frac{\langle S_1, s\rangle \rightarrow s' \qquad \langle S_2, s'\rangle \rightarrow s''}{\langle S_1; S_2, s\rangle \rightarrow s''}$$

$$\frac{\langle S_1, s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \rightarrow s'} \text{ if } \mathcal{B}[\![b]\!](s) = \texttt{true}$$

$$\frac{\langle S_2, s\rangle \rightarrow s'}{\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \rightarrow s'} \text{ if } \mathcal{B}[\![b]\!](s) = \texttt{false}$$

$$\frac{\langle S, s\rangle \rightarrow s' \qquad \langle \texttt{while}\ b\ S, s'\rangle \rightarrow s''}{\langle \texttt{while}\ b\ S, s\rangle \rightarrow s''} \text{ if } \mathcal{B}[\![b]\!](s) = \texttt{true}$$

$$\frac{}{\langle \texttt{while}\ b\ S, s\rangle \rightarrow s} \text{ if } \mathcal{B}[\![b]\!](s) = \texttt{false}$$

Example

Let s be a state with $s(x) = 3$. Then, we have

$$(\texttt{y:=1; while } \neg\texttt{(x=1) do (y:=y}\star\texttt{x; x:=x-1)}, s) \to s[y \mapsto 6][x \mapsto 1]$$

Execution Types

We say the execution of a statement S on a state s

- *terminates* if and only if there is a state s' such that $\langle S, s \rangle \rightarrow s'$ and
- *loops* if and only if there is no state s' such that $\langle S, s \rangle \rightarrow s'$.

We say a statement S always terminates if its execution on a state s terminates for all states s, and always loops if its execution on a state s loops for all states s.

Examples

- `while true do skip`
- `while ¬(x=1) do (y:=y⋆x; x:=x-1)`

Semantic Equivalence

We say S_1 and S_2 are semantically equivalent, denoted $S_1 \equiv S_2$, if the following is true for all states s and s':

$$\langle S_1, s\rangle \to s' \quad \text{if and only if} \quad \langle S_2, s\rangle \to s'$$

Example

$$\texttt{while } b \texttt{ do } S \equiv \texttt{if } b \texttt{ then } (S;\ \texttt{while } b \texttt{ do } S) \texttt{ else skip}$$

Proof.

Semantic Function for Statements

The semantic function for statements is the partial function:

$$\mathcal{S}_b : \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

$$\mathcal{S}_b[\![S]\!](s) = \begin{cases} s' & \text{if } \langle S, s \rangle \to s' \\ \textbf{undef} & \text{otherwise} \end{cases}$$

Examples:

- $\mathcal{S}_b[\![$ `y:=1; while` $\neg$`(x=1) do (y:=y`$\star$`x; x:=x-1)` $]\!](s[x \mapsto 3])$
- $\mathcal{S}_b[\![$ `while true do skip` $]\!]$(s)

Summary of **While**

The syntax is defined by the grammar:

$$
\begin{array}{rcl}
a & \to & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \to & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \le a_2 \mid \neg b \mid b_1 \wedge b_2 \\
c & \to & x := a \mid \texttt{skip} \mid c_1 ; c_2 \mid \texttt{if}\ b\ c_1\ c_2 \mid \texttt{while}\ b\ c
\end{array}
$$

The semantics is defined by the functions:

$$
\begin{array}{rcl}
\mathcal{A}[\![a]\!] & : & \mathbf{State} \to \mathbb{Z} \\
\mathcal{B}[\![b]\!] & : & \mathbf{State} \to \mathbf{T} \\
\mathcal{S}_b[\![c]\!] & : & \mathbf{State} \hookrightarrow \mathbf{State}
\end{array}
$$

cf) Implementation: Syntax

$$
\begin{array}{lcl}
a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
c & \rightarrow & x := a \mid \texttt{skip} \mid c_1; c_2 \mid \texttt{if}\ b\ c_1\ c_2 \mid \texttt{while}\ b\ c
\end{array}
$$

```
type a = Int of int | Var of string | Plus of a * a |
         Mult of a * a | Minus of a * a
type b = True | False | Eq of a * a | Le of a * a |
         Neg of b | Conj of b * b
type c = Assign of string * a | Skip | Seq of c * c |
         If of b * c * c | While of b * c
```

cf) Implementation: State

```
type state = (string * int) list
let empty_state = []
let bind x v s = (x,v)::s
let rec find x s =
  match s with
  | (x',v')::s' -> if x = x' then v' else find x s'
  | [] -> raise (Failure ("Not found " ^ x))
```

cf) Implementation: Arithmetic Expressions

$$
\begin{aligned}
\mathcal{A}[\![\, a \,]\!] &: \textbf{State} \rightarrow \mathbb{Z} \\
\mathcal{A}[\![\, n \,]\!](s) &= n \\
\mathcal{A}[\![\, x \,]\!](s) &= s(x) \\
\mathcal{A}[\![\, a_1 + a_2 \,]\!](s) &= \mathcal{A}[\![\, a_1 \,]\!](s) + \mathcal{A}[\![\, a_2 \,]\!](s) \\
\mathcal{A}[\![\, a_1 \star a_2 \,]\!](s) &= \mathcal{A}[\![\, a_1 \,]\!](s) \times \mathcal{A}[\![\, a_2 \,]\!](s) \\
\mathcal{A}[\![\, a_1 - a_2 \,]\!](s) &= \mathcal{A}[\![\, a_1 \,]\!](s) - \mathcal{A}[\![\, a_2 \,]\!](s)
\end{aligned}
$$

```
let rec eval_a : a -> state -> int
=fun a s ->
  match a with
  | Int n -> n
  | Var x -> find x s
  | Plus (a1,a2) -> (eval_a a1 s) + (eval_a a2 s)
  | Mult (a1,a2) -> (eval_a a1 s) * (eval_a a2 s)
  | Minus (a1,a2) -> (eval_a a1 s) - (eval_a a2 s)
```

cf) Implementation: Boolean Expressions

$$
\begin{array}{rcl}
\mathcal{B}[\![\ b\ ]\!] & : & \mathbf{State} \to \mathbf{T} \\
\mathcal{B}[\![\ \mathtt{true}\ ]\!](s) & = & \mathit{true} \\
\mathcal{B}[\![\ \mathtt{false}\ ]\!](s) & = & \mathit{false} \\
\mathcal{B}[\![\ a_1 = a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) = \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{B}[\![\ a_1 \le a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) \le \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{B}[\![\ \neg b\ ]\!](s) & = & \mathcal{B}[\![\ b\ ]\!](s) = \mathit{false} \\
\mathcal{B}[\![\ b_1 \wedge b_2\ ]\!](s) & = & \mathcal{B}[\![\ b_1\ ]\!](s) \wedge \mathcal{B}[\![\ b_2\ ]\!](s)
\end{array}
$$

```
let rec eval_b : b -> state -> bool
=fun b s ->
  match b with
  | True -> true
  | False -> false
  | Eq (a1,a2) -> eval_a a1 s = eval_a a2 s
  | Le (a1,a2) -> eval_a a1 s <= eval_a a2 s
  | Neg b -> not (eval_b b s)
  | Conj (b1,b2) -> (eval_b b1 s) && (eval_b b2 s)
```

cf) Implementation: Statements

$$\frac{}{\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![\, a \,]\!](s)]} \quad \frac{}{\langle \mathtt{skip}, s\rangle \to s}$$

$$\frac{\langle S_1, s\rangle \to s' \qquad \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''} \quad \frac{\langle S_1, s\rangle \to s'}{\langle \mathtt{if}\ b\ S_1\ S_2, s\rangle \to s'} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathtt{true}$$

$$\frac{\langle S_2, s\rangle \to s'}{\langle \mathtt{if}\ b\ S_1\ S_2, s\rangle \to s'} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathtt{false} \quad \frac{}{\langle \mathtt{while}\ b\ S, s\rangle \to s} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathtt{fal}$$

$$\frac{\langle S, s\rangle \to s' \qquad \langle \mathtt{while}\ b\ S, s'\rangle \to s''}{\langle \mathtt{while}\ b\ S, s\rangle \to s''} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathtt{true}$$

```
let rec eval_c : c -> state -> state
=fun c s -> match c with
  | Assign (x,a) -> bind x (eval_a a s) s
  | Skip -> s
  | Seq (c1,c2) -> eval_c c2 (eval_c c1 s)
  | If (b,c1,c2) -> if eval_b b s then eval_c c1 s else eval_c c2 s
  | While (b,c) ->
    if eval_b b s then eval_c (While (b,c)) (eval_c c s) else s
```

cf) Implementation: Running Factorial

$$y:=1;\ \texttt{while}\ \neg(x=1)\ \texttt{do}\ (y:=y\star x;\ x:=x-1)$$

```
let fact =
    Seq (Assign ("y", Int 1),
       While (Neg (Eq (Var "x", Int 1)),
         Seq (Assign ("y",Mult (Var "y", Var "x")),
           Assign ("x", Minus (Var "x", Int 1)))))

let state = eval_c fact [("x", 3)]
let _ = print_int (find "y" state)
```