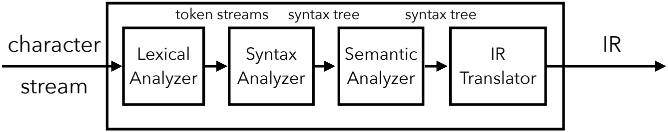
COSE312: Compilers

Lecture 8 — Operational Semantics

Hakjoo Oh
2026 Spring

Semantic Analysis



Semantic analysis aims to statically detect runtime errors, e.g.,

```
int a[10] = {...};
int x = rand();
int y = 1;
if (x > 0) {
  if (x < 15) {
    if (x < 10) a[x] = "hello" + y;
    a[x] = 1;
  }
} else {
  y = y / x;
}
```

Syntax vs. Semantics

A programming language is defined with syntax and semantics.

- The syntax is concerned with the grammatical structure of programs.
  - Context-free grammar
- The semantics is concerned with the meaning of programs. Two approaches to specifying program semantics:
  - Operational semantics: The meaning is specified by the computation steps executed on a machine. Interested in how it is obtained.
  - Denotational semantics: The meaning is modelled by mathematical objects that represent the effect of executing the program. Interested in the effect, not how it is obtained.

The **While** Language: Abstract Syntax

n will range over numerals, **Num**
x will range over variables, **Var**
a will range over arithmetic expressions, **Aexp**
b will range over boolean expressions, **Bexp**
S will range over statements, **Stm**

$$
\begin{array}{rcl}
a & \rightarrow & n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2 \\
b & \rightarrow & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
S & \rightarrow & x := a \mid \texttt{skip} \mid S_1; S_2 \mid \texttt{if}\ b\ S_1\ S_2 \mid \texttt{while}\ b\ S
\end{array}
$$

Example

The factorial program:

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

The abstract syntax tree:

Semantics of Arithmetic Expressions

- The meaning of an expression depends on the values bound to the variables that occur in the expression, e.g., $x + 3$.
- A state is a function from variables to values:

$$s \in \mathbf{State} = \mathbf{Var} \to \mathbb{Z}$$

- The meaning of arithmetic expressions is a function:

$$\mathcal{A} : \mathbf{Aexp} \to \mathbf{State} \to \mathbb{Z}$$

$$\begin{array}{rcl}
\mathcal{A}[\![\ a\ ]\!] & : & \mathbf{State} \to \mathbb{Z} \\
\mathcal{A}[\![\ n\ ]\!](s) & = & n \\
\mathcal{A}[\![\ x\ ]\!](s) & = & s(x) \\
\mathcal{A}[\![\ a_1 + a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) + \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{A}[\![\ a_1 \star a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) \times \mathcal{A}[\![\ a_2\ ]\!](s) \\
\mathcal{A}[\![\ a_1 - a_2\ ]\!](s) & = & \mathcal{A}[\![\ a_1\ ]\!](s) - \mathcal{A}[\![\ a_2\ ]\!](s)
\end{array}$$

Semantics of Boolean Expressions

- The meaning of boolean expressions is a function:

$$\mathcal{B} : \textbf{Bexp} \to \textbf{State} \to \mathbf{T}$$

where $\mathbf{T} = \{true, false\}$.

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

Free Variables

The free variables of an expression are defined to be the set of variables occurring in it:

$$
\begin{array}{rcl}
FV(n) & = & \emptyset \\
FV(x) & = & \{x\} \\
FV(a_1 + a_2) & = & FV(a_1) \cup FV(a_2) \\
FV(a_1 \star a_2) & = & FV(a_1) \cup FV(a_2) \\
FV(a_1 - a_2) & = & FV(a_1) \cup FV(a_2)
\end{array}
$$

$$
\begin{array}{rcl}
FV(\texttt{true}) & = & \emptyset \\
FV(\texttt{false}) & = & \emptyset \\
FV(a_1 = a_2) & = & FV(a_1) \cup FV(a_2) \\
FV(a_1 \leq a_2) & = & FV(a_1) \cup FV(a_2) \\
FV(\neg b) & = & FV(b) \\
FV(b_1 \ \wedge \ b_2) & = & FV(b_1) \cup FV(b_2)
\end{array}
$$

Only the free variables influence the value of an expression.

Lemma

Let s and s' be two states such that $s(x) = s'(x)$ for all $x \in FV(a)$. Then, $\mathcal{A}[\![a]\!](s) = \mathcal{A}[\![a]\!](s')$.

Proof) By structural induction on a.

- n: $\mathcal{A}[\![n]\!](s) = n = \mathcal{A}[\![n]\!](s')$.
- x: $\mathcal{A}[\![x]\!](s) = s(x) = s'(x) = \mathcal{A}[\![x]\!](s')$.
- $a_1 + a_2$:

$$\begin{array}{lll} \mathcal{A}[\![a_1 + a_2]\!](s) & = \mathcal{A}[\![a_1]\!](s) + \mathcal{A}[\![a_2]\!](s) & \cdots \text{def. of } \mathcal{A}[\![a_1 + a_2]\!] \\ & = \mathcal{A}[\![a_1]\!](s') + \mathcal{A}[\![a_2]\!](s') & \cdots \text{Induction Hypothesis (I.H.)} \\ & = \mathcal{A}[\![a_1 + a_2]\!](s') & \cdots \text{def. of } \mathcal{A}[\![a_1 + a_2]\!] \end{array}$$

- $a_1 \star a_2, a_1 - a_2$: Similar.

□

Lemma

Let s and s' be two states such that $s(x) = s'(x)$ for all $x \in FV(b)$. Then, $\mathcal{B}[\![b]\!](s) = \mathcal{B}[\![b]\!](s')$.

Proof) Exercise.

Substitution

- $a[y \mapsto a_0]$: the arithmetic expression that is obtained by replacing each occurrence of y in a by a_0.

$$\begin{array}{rcl} n[y \mapsto a_0] & = & n \\ x[y \mapsto a_0] & = & \begin{cases} a_0 & \text{if } x = y \\ x & \text{if } x \neq y \end{cases} \\ (a_1 + a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) + (a_2[y \mapsto a_0]) \\ (a_1 \star a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) \star (a_2[y \mapsto a_0]) \\ (a_1 - a_2)[y \mapsto a_0] & = & (a_1[y \mapsto a_0]) - (a_2[y \mapsto a_0]) \end{array}$$

- $s[y \mapsto v]$: the state s except that the value bound to y is v.

$$(s[y \mapsto v])(x) = \begin{cases} v & \text{if } x = y \\ s(x) & \text{if } x \neq y \end{cases}$$

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 $(\mathbb{S}, \to)$ where $\mathbb{S}$ is the set of states (configurations) 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 $(\to) \subseteq \mathbb{S} \times \mathbb{S}$ 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:

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

Transition relation is defined with inference rules of the form:

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

- $S_1, \ldots, S_n$ 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**

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

B-Skip
$$\frac{}{\langle \texttt{skip}, s\rangle \to s}$$

B-Seq
$$\frac{\langle S_1, s\rangle \to s'' \qquad \langle S_2, s''\rangle \to s'}{\langle S_1; S_2, s\rangle \to s'}$$

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

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

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

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

Example

Consider the statement:

```
(z:=x; x:=y); y:=z
```

Let s_0 be the state that maps all variables except x and y and has $s_0(\mathrm{x}) = 5$ and $s_0(\mathrm{y}) = 7$. Applying the semantics rules generates the following *derivation tree*:

$$\frac{\dfrac{\langle \mathrm{z} := \mathrm{x}, s_0\rangle \to s_1 \qquad \langle \mathrm{x} := \mathrm{y}, s_1\rangle \to s_2}{\langle \mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}, s_0\rangle \to s_2} \qquad \langle \mathrm{y} := \mathrm{z}, s_2\rangle \to s_3}{\langle (\mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}); \mathrm{y} := \mathrm{z}, s_0\rangle \to s_3}$$

where we have used the abbreviations:

$$\begin{array}{rcl} s_1 & = & s_0[z \mapsto 5] \\ s_2 & = & s_1[x \mapsto 7] = s_0[z \mapsto 5, x \mapsto 7] \\ s_3 & = & s_2[y \mapsto 5] = s_0[z \mapsto 5, x \mapsto 7, y \mapsto 5] \end{array}$$

Exercise

Let s be a state with $s(x) = 3$. Find the derivation of

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

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

Examples:

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

Semantic Equivalence

With formal semantics, we can now rigorously reason about program behavior, e.g., semantic equivalence.

- S_1 and S_2 are syntactically equivalent if $S_1 = S_2$.
- 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

Lemma

For any $b \in \mathbf{Bexp}, S \in \mathbf{Stm}$,

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

Proof) To show:

$$\forall s, s' \in \mathbf{State}.\ \langle \texttt{while } b\ S, s\rangle \to s' \iff \langle \texttt{if } b\ (S; \texttt{while } b\ S)\ \texttt{skip}, s\rangle \to s'$$

$\boxed{\Rightarrow}$ Suppose $\langle \texttt{while } b\ S, s\rangle \to s'$ for states s, s'. Then there must be a derivation of $\langle \texttt{while } b\ S, s\rangle \to s'$, where the final rule is either

$$\frac{}{\langle \texttt{while } b\ S, s\rangle \to s} \text{ if } \mathcal{B}[\![b]\!](s) = \mathit{false} \qquad (1)$$

where $s = s'$ or

$$\frac{\langle S, s\rangle \to s'' \qquad \langle \texttt{while } b\ S, s''\rangle \to s'}{\langle \texttt{while } b\ S, s\rangle \to s'} \text{ if } \mathcal{B}[\![b]\!](s) = \mathit{true} \qquad (2)$$

- In case (1), because $\mathcal{B}[\![b]\!](s) = \mathit{false}$, we can build the following derivation:

$$\dfrac{\overline{\langle \texttt{skip}, s\rangle \to s}}{\langle \texttt{if}\ b\ (S; \texttt{while}\ \ b\ \ S)\ \texttt{skip}, s\rangle \to s} \text{ if } \mathcal{B}[\![b]\!](s) = \mathit{false}$$

- In case (2), the derivation must have the form:

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

  Using this, we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\langle S, s\rangle \to s''} \qquad \dfrac{\vdots}{\langle \texttt{while}\ \ b\ \ S, s''\rangle \to s'}}{\langle S; \texttt{while}\ \ b\ \ S, s\rangle \to s'}}{\langle \texttt{if}\ b\ (S; \texttt{while}\ \ b\ \ S)\ \texttt{skip}, s\rangle \to s'} \text{ if } \mathcal{B}[\![b]\!](s) = \mathit{true}$$

In either case, we obtain a derivation of $\langle \texttt{if}\ b\ (S; \texttt{while}\ \ b\ \ S)\ \texttt{skip}, s\rangle \to s'$. Thus,

$$\forall s, s' \in \mathbf{State}.\ \langle \texttt{while}\ \ b\ \ S, s\rangle \to s' \implies \langle \texttt{if}\ b\ (S; \texttt{while}\ \ b\ \ S)\ \texttt{skip}, s\rangle \to s'$$

$\boxed{\Leftarrow}$ Suppose $\langle \texttt{if } b\ (S; \texttt{while } b\ S)\ \texttt{skip}, s\rangle \to s'$ for states s, s'. Then there are two possibilities:

$$\frac{\langle \texttt{skip}, s\rangle \to s}{\langle \texttt{if } b\ (S; \texttt{while } b\ S)\ \texttt{skip}, s\rangle \to s} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathit{false} \tag{3}$$

$$\frac{\dfrac{\vdots}{\langle (S; \texttt{while } b\ S), s\rangle \to s'}}{\langle \texttt{if } b\ (S; \texttt{while } b\ S)\ \texttt{skip}, s\rangle \to s'} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathit{true} \tag{4}$$

From either derivation, we can construct a derivation of $\langle \texttt{while } b\ S, s\rangle \to s'$. Consider the second case, (4), which has a derivation of $\langle S; \texttt{while } b\ S, s\rangle \to s'$ of the form

$$\frac{\dfrac{\vdots}{\langle S, s\rangle \to s''} \quad \dfrac{\vdots}{\langle \texttt{while } b\ S, s''\rangle \to s'}}{\langle S; \texttt{while } b\ S, s\rangle \to s'}$$

for some state s''. Using the derivations of $\langle S, s\rangle \to s''$ and $\langle \texttt{while } b\ S, s''\rangle \to s'$, we build

$$\frac{\dfrac{\vdots}{\langle S, s\rangle \to s''} \quad \dfrac{\vdots}{\langle \texttt{while } b\ S, s''\rangle \to s'}}{\langle \texttt{while } b\ S, s\rangle \to s'} \text{ if } \mathcal{B}[\![\, b \,]\!](s) = \mathit{true}$$

It is easy to construct a derivation of $\langle \texttt{while } b\ S, s\rangle \to s'$ from (3). Thus,

$$\forall s, s' \in \mathbf{State}.\ \langle \texttt{while } b\ S, s\rangle \to s' \Longleftarrow \langle \texttt{if } b\ (S; \texttt{while } b\ S)\ \texttt{skip}, s\rangle \to s'$$

$\square$

Semantic Function for Statements

The semantics of statements can be defined by the partial function:

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

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

Examples:

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

Implementing Big-Step Interpreter

```
type var = string

type aexp =
  | Int of int
  | Var of var
  | Plus of aexp * aexp
  | Mult of aexp * aexp
  | Minus of aexp * aexp

type bexp =
  | True
  | False
  | Eq of aexp * aexp
  | Le of aexp * aexp
  | Neg of bexp
  | Conj of bexp * bexp

type cmd =
  | Assign of var * aexp
  | Skip
  | Seq of cmd * cmd
  | If of bexp * cmd * cmd
  | While of bexp * cmd
```

Implementing Big-Step Interpreter

```
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)))
       )
  )

module State = struct
  type t = (var * int) list
  let empty = []
  let rec lookup s x =
    match s with
    | [] -> 0
    | (y,v)::s' -> if x = y then v else lookup s' x
  let update s x v = (x,v)::s
end

let init_s = update empty "x" 3
```

Implementing Big-Step Interpreter

```
let rec eval_a : aexp -> State.t -> int
=fun a s ->
  match a with
  | Int n -> n
  | Var x -> State.lookup s x
  | 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)

let rec eval_b : bexp -> State.t -> 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)
```

Implementing Big-Step Interpreter

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

let _ =
    print_int (State.lookup (eval_c fact init_s) "y");
    print_newline ()
```

Small-Step Operational Semantics

The individual computation steps are described by the transition relation of the form:

$$\langle S, s\rangle \Rightarrow \gamma$$

where γ either is non-terminal state $\langle S', s'\rangle$ or terminal state s'. The transition expresses the first step of the execution of S from state s.

- If $\gamma = \langle S', s'\rangle$, then the execution of S from s is not completed and the remaining computation continues with $\langle S', s'\rangle$.
- If $\gamma = s'$, then the execution of S from s has terminated and the final state is s'.

We say $\langle S, s\rangle$ is stuck if there is no γ such that $\langle S, s\rangle \Rightarrow \gamma$ (no stuck state for **While**).

Small-Step Operational Semantics for **While**

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

S-Skip $$\frac{}{\langle \texttt{skip}, s\rangle \Rightarrow s}$$

S-Seq1 $$\frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_1'; S_2, s'\rangle}$$

S-Seq2 $$\frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

S-IfT $$\frac{}{\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle}\ \text{if } \mathcal{B}[\![\, b \,]\!](s) = \mathit{true}$$

S-IfF $$\frac{}{\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \Rightarrow \langle S_2, s\rangle}\ \text{if } \mathcal{B}[\![\, b \,]\!](s) = \mathit{false}$$

S-While $$\frac{}{\langle \texttt{while}\ b\ S, s\rangle \Rightarrow \langle \texttt{if}\ \ b\ \ \texttt{(}S\texttt{;}\ \ \texttt{while}\ \ b\ \ S\texttt{)}\ \ \texttt{skip}, s\rangle}$$

Derivation Sequence

A *derivation sequence* of a statement S starting in state s is either

- A finite sequence

$$\gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_k$$

  which is sometimes written

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots \Rightarrow \gamma_k$$

  consisting of configurations satisfying

$$\gamma_0 = \langle S, s \rangle, \quad \gamma_i \Rightarrow \gamma_{i+1} \text{ for } 0 \leq i < k$$

  where $k \geq 0$ and γ_k is either a terminal or stuck configuration.

- An infinite sequence

$$\gamma_0, \gamma_1, \gamma_2, \cdots$$

  which is sometimes written

$$\gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \cdots$$

  consisting of configurations satisfying $\gamma_0 = \langle S, s \rangle$ and $\gamma_i \Rightarrow \gamma_{i+1}$ for $0 \leq i$.

Example

Consider the statement:

```
(z:=x; x:=y); y:=z
```

Let s_0 be the state that maps all variables except x and y and has $s_0(\mathrm{x}) = 5$ and $s_0(\mathrm{y}) = 7$. We then have the derivation sequence:

$$
\begin{aligned}
&\langle (\mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}); \mathrm{y} := \mathrm{z}, s_0 \rangle \\
\Rightarrow\ &\langle \mathrm{x} := \mathrm{y}; \mathrm{y} := \mathrm{z}, s_0[z \mapsto 5] \rangle \\
\Rightarrow\ &\langle \mathrm{y} := \mathrm{z}, s_0[z \mapsto 5, x \mapsto 7] \rangle \\
\Rightarrow\ &s_0[z \mapsto 5, x \mapsto 7, y \mapsto 5]
\end{aligned}
$$

Each step has a derivation tree explaining why it takes place, e.g.,

$$
\cfrac{\cfrac{\langle \mathrm{z} := \mathrm{x}, s_0 \rangle \Rightarrow s_0[z \mapsto 5]}{\langle \mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}, s_0 \rangle \Rightarrow \langle \mathrm{x} := \mathrm{y}, s_0[z \mapsto 5] \rangle}}{\langle (\mathrm{z} := \mathrm{x}; \mathrm{x} := \mathrm{y}); \mathrm{y} := \mathrm{z}, s_0 \rangle \Rightarrow \langle \mathrm{x} := \mathrm{y}; \mathrm{y} := \mathrm{z}, s_0[z \mapsto 5] \rangle}
$$

Example: Factorial

Assume that $s(x) = 3$.

$\langle$`y:=1; while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s\rangle$
$\Rightarrow \langle$`while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 1]\rangle$
$\Rightarrow \langle$`if ¬(x=1) then ((y:=y⋆x; x:=x-1);while ¬(x=1) do (y:=y⋆x; x:=x-1)) else skip`$, s[y \mapsto 1]\rangle$
$\Rightarrow \langle$`((y:=y⋆x; x:=x-1);while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 1]\rangle$
$\Rightarrow \langle$`x:=x-1;while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 3]\rangle$
$\Rightarrow \langle$`while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 3][x \mapsto 2]\rangle$
$\Rightarrow \langle$`if ¬(x=1) then ((y:=y⋆x; x:=x-1);while ¬(x=1) do (y:=y⋆x; x:=x-1)) else skip`$, s[y \mapsto 3][x \mapsto 2]\rangle$
$\Rightarrow \langle$`((y:=y⋆x; x:=x-1);while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 3][x \mapsto 2]\rangle$
$\Rightarrow \langle$`x:=x-1;while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 6][x \mapsto 2]\rangle$
$\Rightarrow \langle$`while ¬(x=1) do (y:=y⋆x; x:=x-1)`$, s[y \mapsto 6][x \mapsto 1]\rangle$
$\Rightarrow s[y \mapsto 6][x \mapsto 1]$

Other Notations

- We write $\gamma_0 \Rightarrow^k \gamma_k$ to indicate that there are k steps in the execution from γ_0 to γ_k.
- We write $\gamma \Rightarrow^* \gamma'$ to indicate that there are a finite number of steps.
- We say that the execution of a statement S on a state s terminates if and only if there is a finite derivation sequence starting with $\langle S, s \rangle$.
- The execution loops if and only if there is an infinite derivation sequence starting with $\langle S, s \rangle$.

Semantic Function

The semantic function $\mathcal{S}_s$ for small-step semantics:

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

$$\mathcal{S}_s[\![S]\!](s) = \begin{cases} s' & \text{if } \langle S, s\rangle \Rightarrow^* s' \\ \mathbf{undef} & \end{cases}$$

Implementing Small-Step Interpreter

```
type conf =
  | NonTerminated of cmd * State.t
  | Terminated of State.t

let rec next : conf -> conf
=fun conf ->
  match conf with
  | Terminated _ -> raise (Failure "Must not happen")
  | NonTerminated (c, s) ->
    match c with
    | Assign (x, a) -> Terminated (State.update s x (eval_a a s))
    | Skip -> Terminated s
    | Seq (c1, c2) -> (
       match  (next (NonTerminated (c1,s))) with
       | NonTerminated (c', s') -> NonTerminated (Seq (c', c2), s')
       | Terminated s' -> NonTerminated (c2, s')
      )
    | If (b, c1, c2) ->
      if eval_b b s then NonTerminated (c1, s) else NonTerminated (c2, s)
    | While (b, c) -> NonTerminated (If (b, Seq (c, While (b,c)), Skip), s)
```

Implementing Small-Step Interpreter

```
let rec next_trans : conf -> State.t
=fun conf ->
  match conf with
  | Terminated s -> s
  | _ -> next_trans (next conf)

let _ =
  print_int (State.lookup (next_trans (NonTerminated (fact,init_s))) "y");
  print_newline ()
```

Equivalence of Big-Step and Small-Step Semantics

Theorem

For every statement S of **While**, *we have $\mathcal{S}_b[\![S]\!] = \mathcal{S}_s[\![S]\!]$.*

Proof) By Lemma (1) and Lemma (2) below.

Lemma (1)

For every statement S of **While** *and states s and s',*

$$\langle S, s\rangle \to s' \implies \langle S, s\rangle \Rightarrow^* s'.$$

Lemma (2)

For every statement S of **While**, *states s and s' and natural number k,*

$$\langle S, s\rangle \Rightarrow^k s' \implies \langle S, s\rangle \to s'.$$

Auxiliary Lemmas

Lemma (3)

If $\langle S_1, s\rangle \Rightarrow^k s'$ *then* $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$.

Proof) Exercise

Lemma (4)

If $\langle S_1; S_2, s\rangle \Rightarrow^k s'$ *then there exists a state* s'' *and natural numbers* k_1 *and* k_2 *such that* $\langle S_1, s\rangle \Rightarrow^{k_1} s''$ *and* $\langle S_2, s''\rangle \Rightarrow^{k_2} s'$, *where* $k = k_1 + k_2$.

Proof) Exercise

Proofs

Lemma (1)

For every statement S of **While** *and states s and s',*

$$\langle S, s\rangle \to s' \implies \langle S, s\rangle \Rightarrow^* s'.$$

Proof) By induction on the derivation of $\langle S, s\rangle \to s'$.

- B-ASSN: Assume that $\langle x := a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!](s)]$. From S-ASSN, we get

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

- B-SKIP: Similar.
- B-SEQ: Assume that $\langle S_1; S_2, s\rangle \to s'$ because $\langle S_1, s\rangle \to s''$ and $\langle S_2, s''\rangle \to s'$. By induction hypotheses (IHs), we have

$$\langle S_1, s\rangle \Rightarrow^* s'' \text{ and } \langle S_2, s''\rangle \Rightarrow^* s'.$$

  We derive the required as follows:

$$\begin{array}{lll} \langle S_1; S_2, s\rangle & \Rightarrow^* \langle S_2, s''\rangle & \cdots \text{Lemma (3) and I.H.1} \\ & \Rightarrow^* s' & \cdots \text{I.H.2} \end{array}$$

- B-IfT: Assume that $\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \to s'$ because $\mathcal{B}[\![b]\!](s) = \mathit{true}$ and $\langle S_1, s\rangle \to s'$. We derive the required as follows:

$$\begin{array}{lll} \langle \texttt{if}\ b\ S_1\ S_2, s\rangle & \Rightarrow \langle S_1, s\rangle & \cdots\ \mathcal{B}[\![b]\!](s) = \mathit{true} \\ & \Rightarrow^*\ s' & \cdots\ \text{I.H.} \end{array}$$

- B-IfF: Similar.
- B-WhileT: Assume that $\langle \texttt{while}\ b\ S, s\rangle \to s'$ because $\mathcal{B}[\![b]\!](s) = \mathit{true}$, $\langle S, s\rangle \to s''$, and $\langle \texttt{while}\ b\ S, s''\rangle \to s'$. We derive the required as follows:

$$\begin{array}{lll} \langle \texttt{while}\ b\ S, s\rangle & \Rightarrow \langle \texttt{if}\ b\ (S;\ \texttt{while}\ b\ S)\ \texttt{skip}, s\rangle & \cdots\ \text{S-While} \\ & \Rightarrow \langle S; \texttt{while}\ b\ S, s\rangle & \cdots\ \text{S-IfT} \\ & \Rightarrow^*\ \langle \texttt{while}\ b\ S, s''\rangle & \cdots\ \text{Lemma (3) and I.H.1} \\ & \Rightarrow^*\ s' & \cdots\ \text{I.H.2} \end{array}$$

- B-WhileF: Assume that $\langle \texttt{while}\ b\ S, s\rangle \to s$ because $\mathcal{B}[\![b]\!](s) = \mathit{false}$.

$$\begin{array}{lll} \langle \texttt{while}\ b\ S, s\rangle & \Rightarrow \langle \texttt{if}\ b\ (S;\ \texttt{while}\ b\ S)\ \texttt{skip}, s\rangle & \cdots\ \text{S-While} \\ & \Rightarrow \langle \texttt{skip}, s\rangle & \cdots\ \text{S-IfF} \\ & \Rightarrow s & \cdots\ \text{S-Skip} \end{array}$$

□

Lemma (2)

For every statement S of **While**, *states s and s' and natural number k,*

$$\langle S, s\rangle \Rightarrow^k s' \implies \langle S, s\rangle \to s'.$$

Proof) By induction on the length of the derivation sequence $\langle S, s\rangle \Rightarrow^k s'$ (i.e., induction on k). Base case ($k = 0$): the result holds vacuously since $\langle S, s\rangle \Rightarrow^0 s'$ cannot hold and the implication is true. Inductive case: we assume that the lemma holds for all $k \leq k_0$ for some k_0 and then prove that it holds for $k_0 + 1$. We proceed by cases on how the first step of $\langle S, s\rangle \Rightarrow^{k_0+1} s'$ is obtained.

- S-Assn, S-Skip: Straightforward (and $k_0 = 0$).
- S-Seq1, S-Seq2: Assume that $\langle S_1; S_2, s\rangle \Rightarrow^{k_0+1} s'$. By Lemma (4), there exists a state s'' and natural numbers k_1 and k_2 such that

  $$\langle S_1, s\rangle \Rightarrow^{k_1} s'' \text{ and } \langle S_2, s''\rangle \Rightarrow^{k_2} s'$$

  where $k_1 + k_2 = k_0 + 1$. The induction hypothesis can be applied to each of these derivation sequences because $k_1 \leq k_0$ and $k_2 \leq k_0$. Thus, we get

  $$\langle S_1, s\rangle \to s'' \text{ and } \langle S_2, s''\rangle \to s'.$$

  Using B-Seq, we get the required $\langle S_1; S_2, s\rangle \to s'$.

- S-IfT: We assume $\mathcal{B}[\![b]\!](s) = true$ and the following derivation of length $k_0 + 1$:

$$\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \Rightarrow \langle S_1, s\rangle \Rightarrow^{k_0} s'.$$

By induction hypothesis, we have $\langle S_1, s\rangle \rightarrow s'$. Using B-IfT, we derive the required

$$\langle \texttt{if}\ b\ S_1\ S_2, s\rangle \rightarrow s'$$

- S-IfF: Similar.
- S-While: By assumption, we have

$$\langle \texttt{while}\ b\ S, s\rangle \Rightarrow \langle \texttt{if}\ \ b\ \ (S;\ \texttt{while}\ \ b\ \ S)\ \ \texttt{skip}, s\rangle \Rightarrow^{k_0} s'$$

By induction hypothesis, we have

$$\langle \texttt{if}\ \ b\ \ (S;\ \texttt{while}\ \ b\ \ S)\ \ \texttt{skip}, s\rangle \rightarrow s'$$

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

$$\langle \texttt{while}\ b\ S, s\rangle \rightarrow s'$$

□

Summary

We have defined the operational semantics of **While**.

- *Big-step operational semantics* describes how the overall results of executions are obtained.

$$\boldsymbol{\mathcal{S}_b}[\![\ S\ ]\!] \quad : \quad \mathbf{State} \hookrightarrow \mathbf{State}$$

- *Small-step operational semantics* describes how the individual steps of the computations take place.

$$\boldsymbol{\mathcal{S}_s}[\![\ S\ ]\!] \quad : \quad \mathbf{State} \hookrightarrow \mathbf{State}$$

- The big-step and small-step semantics are equivalent.