Formal Semantics

Instructor: Raju Halder

Why formal semantics

- To understand how programs behave
- To build a mathematical model useful for program analysis and verification

Kinds of semantics

- Operational Semantics
  - specify how commands and expressions execute on an abstract machine.
- Denotational Semantics
  - defining the meaning of programming languages by mathematical concepts.
- Axiomatic Semantics
  - giving the meaning of a programming construct by axioms or proof rules in a program logic.

A Simple imperative Language - IMP

Syntactic Elements:

- numbers $\mathbf{N}$, consisting of all integer numbers, ranged over by metavariables n, m
- truth values $\mathbf{T}$=$\{\mathbf{true}, \mathbf{false}\}$,
- locations $\mathbf{Loc}$, ranged over by X, Y
- arithmetic expressions $\mathbf{Aexp}$, ranged over by a
- boolean expressions $\mathbf{Bexp}$, ranged over by b
- commands $\mathbf{Com}$, ranged over by c

Abstract Syntax using Backus-Naur Forms (BNF)

- For Aexp: $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
- For Bexp: $b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
- For Com:

  $c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$

States

- Set of states is defied by the following function:

$$\sigma : \mathrm{Loc} \to \mathrm{N}.$$

Structural Operational Semantics for arithmetic expressions

Evaluation of numbers $\langle n, \sigma \rangle \to n$

Evaluation of locations $\langle X, \sigma \rangle \to \sigma(X)$

Evaluation of sums

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the sum of } n_0 \text{ and } n_1}{\langle a_0 + a_1, \sigma \rangle \to n}$$

Evaluation of subtractions

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the result of subtracting } n_1 \text{ from } n_0}{\langle a_0 - a_1, \sigma \rangle \to n}$$

Evaluation of products

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the product of } n_0 \text{ and } n_1}{\langle a_0 \times a_1, \sigma \rangle \to n}$$

Derivation Tree

$$\cfrac{\cfrac{\cfrac{}{\langle Init, \sigma_0 \rangle \to 0} \qquad \cfrac{}{\langle 5, \sigma_0 \rangle \to 5}}{\langle (Init + 5), \sigma_0 \rangle \to 5} \qquad \cfrac{\cfrac{}{\langle 7, \sigma_0 \rangle \to 7} \qquad \cfrac{}{\langle 9, \sigma_0 \rangle \to 9}}{\langle 7 + 9, \sigma_0 \rangle \to 16}}{\langle (Init + 5) + (7 + 9), \sigma_0 \rangle \to 21}$$

Equivalence of arithmetic expressions

- Two arithmetic expressions are semantically equivalent if they evaluate to the same value in all states

$$a_0 \sim a_1 \quad \text{iff} \quad \forall \sigma \in \Sigma\, \forall n \in \mathrm{N}.\ \langle a_0, \sigma \rangle \rightarrow n \Leftrightarrow \langle a_1, \sigma \rangle \rightarrow n$$

- “X+4*Y” and “Y*4+X” syntactically not equivalent, but semantically they are equivalent.

Operational Semantics of Boolean expressions

$$\langle \mathbf{true}, \sigma \rangle \to \mathbf{true} \qquad \langle \mathbf{false}, \sigma \rangle \to \mathbf{false}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to n}{\langle a_0 = a_1, \sigma \rangle \to \mathbf{true}} \qquad \frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad n \not\equiv m}{\langle a_0 = a_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad \text{if } n \text{ is less than or equal to } m}{\langle a_0 \le a_1, \sigma \rangle \to \mathbf{true}}$$

$$\frac{\langle a_0, \sigma \rangle \to n \qquad \langle a_1, \sigma \rangle \to m \qquad \text{if } n \text{ is not less than or equal to } m}{\langle a_0 \le a_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{true}}{\langle \neg b, \sigma \rangle \to \mathbf{false}} \qquad \frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \neg b, \sigma \rangle \to \mathbf{true}}$$

$$\frac{\langle b_0, \sigma \rangle \to t_0 \qquad \langle b_1, \sigma \rangle \to t_1 \qquad \text{if } t \text{ is } \mathbf{true} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{true}}{\langle b_0 \wedge b_1, \sigma \rangle \to t}$$

$$\frac{\langle b_0, \sigma \rangle \to t_0 \qquad \langle b_1, \sigma \rangle \to t_1 \qquad \text{if } t \text{ is } \mathbf{false} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{false}}{\langle b_0 \vee b_1, \sigma \rangle \to t}$$

Operational semantics of commands

$$\sigma[m/X](Y) = \begin{cases} m & \text{if } Y = X \\ \sigma(Y) & \text{if } Y \neq X \end{cases}$$

Atomic commands

$$\langle \mathbf{skip}, \sigma \rangle \to \sigma \qquad \frac{\langle a, \sigma \rangle \to m}{\langle X := a, \sigma \rangle \to \sigma[m/X]}$$

Sequencing

$$\frac{\langle c_0, \sigma \rangle \to \sigma'' \qquad \langle c_1, \sigma'' \rangle \to \sigma'}{\langle c_0; c_1, \sigma \rangle \to \sigma'}$$

Conditionals

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \qquad \langle c_0, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'} \qquad \frac{\langle b, \sigma \rangle \to \mathbf{false} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'}$$

While-loops

$$\frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \qquad \langle c, \sigma \rangle \to \sigma'' \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \to \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma'}$$

Equivalence of commands

$$c_0 \sim c_1 \text{ iff } \forall \sigma, \sigma' \in \Sigma.\ \langle c_0, \sigma \rangle \to \sigma' \Leftrightarrow \langle c_1, \sigma \rangle \to \sigma'$$

Small steps operational semantics

$$\frac{\langle a_0, \sigma \rangle \rightarrow_1 \langle a_0', \sigma \rangle}{\langle a_0 + a_1, \sigma \rangle \rightarrow_1 \langle a_0' + a_1, \sigma \rangle}$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow_1 \langle a_1', \sigma \rangle}{\langle n + a_1, \sigma \rangle \rightarrow_1 \langle n + a_1', \sigma \rangle}$$

$$\langle n + m, \sigma \rangle \rightarrow_1 \langle p, \sigma \rangle \qquad p \text{ is the sume of } n \text{ and } m$$

$$\langle X := 5; Y := 1, \sigma \rangle \rightarrow_1 \langle Y := 1, \sigma[5/X] \rangle \rightarrow_1 \sigma[5/X][1/Y]$$

Denotational Semantics

- Operational semantics is too concrete, build out of syntax.
- Difficult to compare two programs written in different programming languages.
- Represented by partial functions on states

Denotation Semantics of Arithmetic expressions

$$\mathcal{A} : \mathbf{Aexp} \to (\Sigma \to \mathbf{N})$$

$$\begin{aligned}
\mathcal{A}[\![n]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![X]\!] &= \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![a_0 + a_1]\!] &= \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 - a_1]\!] &= \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 \times a_1]\!] &= \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}
\end{aligned}$$

Denotation Semantics of boolean expressions

$$\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathbf{T})$$

$$\begin{aligned}
\mathcal{B}[\![\mathrm{true}]\!] &= \{(\sigma, \mathrm{true}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![\mathrm{false}]\!] &= \{(\sigma, \mathrm{false}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![a_0 = a_1]\!] &= \{(\sigma, \mathrm{true}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[\![a_0]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
&\quad \{(\sigma, \mathrm{false}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[\![a_0]\!]\sigma \neq \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
\mathcal{B}[\![\neg b]\!] &= \{(\sigma, \neg_T t) \mid \sigma \in \Sigma \ \& \ (\sigma, t) \in \mathcal{B}[\![b]\!]\} \\
\mathcal{B}[\![b_0 \wedge b_1]\!] &= \{(\sigma, t_0 \wedge_T t_1) \mid \sigma \in \Sigma \ \& \ (\sigma, t_0) \in \mathcal{B}[\![b_0]\!] \ \& \ (\sigma, t_1) \in \mathcal{B}[\![b_1]\!]\} \\
&\ldots
\end{aligned}$$

Denotation Semantics of commands

$$\mathcal{C} : \mathbf{Aexp} \to (\Sigma \to \Sigma)$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!] &= \{(\sigma,\sigma) \mid \sigma \in \Sigma\} \\
\mathcal{C}[\![X := a]\!] &= \{(\sigma,\sigma[n/X]) \mid \sigma \in \Sigma \ \ \& \ \ n = \mathcal{A}[\![a]\!]\sigma\} \\
\mathcal{C}[\![c_0; c_1]\!] &= \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!] \\
\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!] &= \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_0]\!]\} \cup \\
&\quad\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_1]\!]\} \\
\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] &= fix(\Gamma)
\end{aligned}$$

where

$$\begin{aligned}
\Gamma(\varphi) = \ &\{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \varphi \circ \mathcal{C}[\![c]\!]\} \cup \\
&\{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\}
\end{aligned}$$

Axiomatic Semantics

- Is my program correct?
  - Does my program satisfy its specification?

- Original purpose: formal program verification

- A formal proof system for properties of the program based on formal logic (predicate calculus)

- Known as Hoare or Floyd-Hoare rules.

Example specifications

- This program terminates.
- All array accesses are within array bounds, no null dereferences, and no unexpected exceptions
- The method returns a sorted array
- The variables x and y are always identical whenever z is 0

Example

- A program that computes the sum of the first hundred numbers:

$S := 0;$
$N := 1;$
while $\neg(N = 101)$ **do**
    $S := S + N;$
    $N := N + 1;$

- Adding assertions at each point......

$$
\begin{array}{l}
S := 0; \\
\{S = 0\} \\
N := 1; \\
\{N = 1\} \\
\textbf{while } \neg(N = 101) \textbf{ do} \\
\qquad\qquad \{1 \leq N < 101 \wedge S = \sum_{1 \leq m < N} m\} \\
\qquad\qquad S := S + N; \\
\qquad\qquad \{1 \leq N < 101 \wedge S = \sum_{1 \leq m \leq N} m\} \\
\qquad\qquad N := N + 1; \\
\{N = 101 \wedge S = \sum_{1 \leq m \leq 100} m\}
\end{array}
$$

Partial Correctness

- Any terminating execution of “c” from a state satisfying “A” ends up in a state satisfying “B”.

$$\{A\}c\{B\}$$

- Example: {y<= x} z := x; z := z + 1 {y < z}
- Known as Hoare Assertions or Hoare Triples.
- Does not say anything about non-terminating execution. Example, $\{\text{true}\}\text{while true do skip}\{\text{false}\}$

Total Correctness

- For all states σ which satisfy “A”, the execution of “c” from σ must terminate in a state σ' that satisfies “B”.

$$[A]c[B]$$

The Assertion Language

- Aexpv:
  - extending Aexp with integer variables
  - *i* ranges over integer variables, *n* ranges over numbers, *X* ranges over locations

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

- Assn:

$$A ::= \mathrm{true} \mid \mathrm{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \forall i.A \mid \exists i.A$$

Free Integer Variables

- Using structural induction:

$$FV(n) = FV(X) = \emptyset$$

$$FV(i) = \{i\}$$

$$FV(a_0 + a_1) = FV(a_0 - a_1) = FV(a_0 \times a_1) = FV(a_0) \cup FV(a_1)$$

$$FV(\mathbf{true}) = FV(\mathbf{false}) = \emptyset$$

$$FV(a_0 = a_1) = FV(a_0 \leq a_1) = FV(a_0) \cup FV(a_1)$$

$$FV(A_0 \wedge A_1) = FV(A_0 \vee A_1) = FV(A_0 \Rightarrow A_1) = FV(A_0) \cup FV(A_1)$$

$$FV(\neg A) = FV(A)$$

$$FV(\forall i.A) = FV(\exists i.A) = FV(A) \backslash \{i\}$$

Substitution

$$n[a/i] \equiv n \qquad X[a/i] \equiv X$$

$$j[a/i] \equiv j \qquad i[a/i] \equiv a$$

$$(a_0 + a_1)[a/i] \equiv (a_0[a/i] + a_1[a/i])$$

$$\ldots$$

$$\mathbf{true}[a/i] \equiv \mathbf{true} \qquad \mathbf{false}[a/i] \equiv \mathbf{false}$$

$$(a_0 = a_1)[a/i] \equiv (a_0[a/i] = a_1[a/i])$$

$$(A_0 \wedge A_1)[a/i] \equiv (A_0[a/i] \wedge A_1[a/i])$$

$$(\neg A)[a/i] \equiv \neg(A[a/i])$$

$$(\forall j.A)[a/i] \equiv \forall j.(A[a/i]) \qquad (\forall i.A)[a/i] \equiv \forall i.A$$

$$(\exists j.A)[a/i] \equiv \exists j.(A[a/i]) \qquad (\exists i.A)[a/i] \equiv \exists i.A$$

The meaning of Aexpv

An interpretation is a function that assigns an integer to each integer variable $I : \mathrm{Intvar} \to \mathrm{N}$

The value of an expression $a \in Aexprv$ in an interpretation *I* and state σ is written $\mathcal{A}v[\![a]\!]I\sigma$.

$$\mathcal{A}v[\![n]\!]I\sigma = n$$
$$\mathcal{A}v[\![X]\!]I\sigma = \sigma(X)$$
$$\mathcal{A}v[\![i]\!]I\sigma = I(i)$$
$$\mathcal{A}v[\![a_0 + a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma + \mathcal{A}v[\![a_1]\!]I\sigma$$
$$\mathcal{A}v[\![a_0 - a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma - \mathcal{A}v[\![a_1]\!]I\sigma$$
$$\mathcal{A}v[\![a_0 \times a_1]\!]I\sigma = \mathcal{A}v[\![a_0]\!]I\sigma \times \mathcal{A}v[\![a_1]\!]I\sigma$$

The meaning of Assn

- For an assertion $A \in$ Assn, $\sigma \models^I A$ means σ satisfies A in interpretation I.
- $I[n/i](j) = n$ if $j \equiv i$, and $I(j)$ otherwise.

$$
\begin{aligned}
&\sigma \models^I \mathbf{true} \\
&\sigma \models^I (a_0 = a_1) \text{ if } \mathcal{A}v[\![a_0]\!]I\sigma = \mathcal{A}v[\![a_1]\!]I\sigma \\
&\sigma \models^I A \wedge B \text{ if } \sigma \models^I A \text{ and } \sigma \models^I B \\
&\sigma \models^I A \Rightarrow B \text{ if } \sigma \not\models^I A \text{ or } \sigma \models^I B \\
&\sigma \models^I \forall i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for all } n \in \mathbf{N} \\
&\sigma \models^I \exists i.A \text{ if } \sigma \models^{I[n/i]} A \text{ for some } n \in \mathbf{N} \\
&\perp \models^I A \\
&\ldots
\end{aligned}
$$

Proof rules for partial correctness

- Proof rules are also called Hoare rules and proof system is called Hoare Logic

$$\{A\}\ \mathbf{skip}\ \{A\}$$

$$\{B[a/X]\}\ X := a\ \{B\}$$

$$\frac{\{A\}c_0\{C\} \quad \{C\}c_1\{B\}}{\{A\}\ c_0; c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}c_0\{B\} \quad \{A \wedge \neg b\}c_1\{B\}}{\{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\ \{B\}}$$

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{A \wedge \neg b\}}$$

$$\frac{\models (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{B\}}$$

Example

Let $w \equiv (\textbf{while } X > 0 \textbf{ do } Y := X \times Y; X := X - 1)$, and show

$$\{X = n \;\&\; n \geq 0 \;\&\; Y = 1\} w \{Y = n!\}$$

Take $I \equiv (Y \times X! = n! \;\&\; X \geq 0)$, then

$$\{I \wedge X > 0\} Y := X \times Y; X := X - 1 \{I\}$$

and so $\{I\} w \{I \wedge X \not> 0\}$.

Note $X = n \;\&\; n \geq 0 \;\&\; Y = 1 \Rightarrow I$ and $I \wedge X \not> 0 \Rightarrow Y = n!$

- **Soundness:** if {P} S {Q} can be proven, then it is certain that executing S from a store satisfying P will only terminate in stores satisfying Q
- **Completeness:** the converse of soundness

- Axiomatic semantics has many applications, such as:
  - Program verifiers
  - Symbolic execution tools for bug hunting
  - Software validation tools
  - Malware detection
  - Automatic test generation