Program Semantics

17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

Why Semantics?

• Semantics describe formally what a program means
  • Typically, how the program executes
• Framework for analysis
  • Precise definitions
  • Proofs of correctness
• Semantics in practice
  • Difficult to define for full languages
    • But see Standard ML!
  • Very useful for thinking about how analysis applies to the “core” of a language
    • Extension to full language is assumed to be easy
      • Sometimes true, sometimes not!
Forms of Program Semantics

- **Big-Step Reduction Semantics**
  - Shows result of program
  - Depends on environment $E : \text{Var} \rightarrow \text{Value}$
  - Forms: $E \vdash a \downarrow n \quad E \vdash S \downarrow E'$
    - In environment $E$, expression $a$ reduces to number $n$
    - In environment $E$, statement $S$ executes to a new environment $E'$
  - The primary semantics used in this course

- **Small-Step Reduction Semantics**
  - Shows step-by-step execution of program
  - Form: $(E, e) \rightarrow (E', e')$
    - In environment $E$, expression $e$ steps to expression $e'$ and produces a new environment $E'$

- **Denotational Semantics**
  - Form: $\llbracket P \rrbracket = O$
    - The meaning of program $P$ is mathematical object $O$

The WHILE Language

- A simple procedural language with:
  - assignment
  - statement sequencing
  - conditionals
  - while loops

- Used in early papers (e.g. Hoare 69) as a “sandbox” for thinking about program semantics

- We will use it to illustrate several different kinds of analysis
### WHILE Syntax

- **Categories of syntax**
  - $S \in \text{Stmt}$ statements
  - $a \in \text{AExp}$ arithmetic expressions
  - $x,y \in \text{Var}$ variables
  - $n \in \text{Num}$ number literals
  - $P \in \text{BExp}$ boolean expressions

- **Syntax**
  - $S ::= x := a \mid \text{skip} \mid S_1 ; S_2$
    - if $P$ then $S_1$ else $S_2$
    - while $P$ do $S$
  - $a ::= x \mid n \mid a_1 \ op_a \ a_2$
  - $\text{op}_a ::= + \mid - \mid * \mid / \ ...$
  - $P ::= \text{true} \mid \text{false} \mid \neg P \mid P_1 \ op_b \ P_2 \mid a_1 \ op_r \ a_2$
  - $\text{op}_b ::= \text{and} \mid \text{or} \ ...$
  - $\text{op}_r ::= \prec \mid \preceq \mid \succ \mid \succeq \ ...$

---

### Example WHILE Program

```plaintext
y := x;
z := 1;
while y>1 do
  z := z * y;
y := y - 1
```

Computes the factorial function, with the input in $x$ and the output in $z$
Big-Step Semantics

- We use big-step to show expression evaluation
- Inference rule format
  - Premises above the line \( \text{premise}_1 \) \( \text{premise}_2 \)
  - Conclusion below the line \( \text{conclusion} \)
  - Read, “If premises, then conclusion”
- Example: operators
  - If expression \( a \) evaluates to a number \( n \)
  - And expression \( a' \) evaluates to a number \( n' \)
  - Then the expression \( a + a' \) evaluates to \( n + n' \)
  - In the rule, we distinguish the textual \( + \) operator from the mathematical \( + \) operator (for which we use boldface font)

\[
E \vdash a \downarrow n \quad E \vdash a' \downarrow n' \\
\hline
E \vdash a + a' \downarrow n + n'
\]

While Expression Big-Step Semantics

- Values evaluate to themselves
  - \( n, \text{true}, \text{false} \)
  - \( E \{ x \} = n \)
  - \( E \vdash x \downarrow n \)

- Variables \( x \) evaluate to the value in the environment
  - \( E \{ x \} \)
  - \( E \vdash a \downarrow n \quad E \vdash a' \downarrow n' \\
\hline
E \vdash \text{op} a' \downarrow n \text{ op } n'
\]

- Operators evaluate according to mathematical operators
  - \(+, -, *, /, \text{not}, \text{and}, \text{or}, <, \leq, \ldots\)
  - \text{math}: indicates mathematical operators
  - \text{boldface}: indicates mathematical values
  - \text{italics}: indicates program text

\[
E \vdash \text{true} \downarrow \text{true} \\
E \vdash \text{false} \downarrow \text{false} \\
E \vdash P \downarrow b \\
E \vdash !P \downarrow !b \\
E \vdash P \downarrow b \quad E \vdash P' \downarrow b' \\
\hline
E \vdash P \text{ op } P' \downarrow b \text{ op } b'
\]
Applying Semantic Rules

- A tree of inference rules forms a derivation
  - Rules at top are axioms; they have no premises

- Example:
  - $E = \{x \rightarrow 3, y \rightarrow 5\}$
  - $P = x + 3 > y$

\[
\begin{aligned}
E\{x\} &= 3 \\
E \vdash x \downarrow 3 \\
E \vdash 3 \downarrow 3 \\
E \vdash y \downarrow 5 \\
E \vdash x + 3 \downarrow 6 \\
E \vdash x + 3 > y \downarrow \text{true}
\end{aligned}
\]

Practice: Applying Semantic Rules

- Example:
  - $E = \{x \rightarrow 3, y \rightarrow 5\}$
  - $P = x > y \text{ and true}$

\[
\begin{aligned}
E \vdash n \downarrow n \\
E\{x\} &= n \\
E \vdash x \downarrow n \\
E \vdash a \downarrow n \\
E \vdash a' \downarrow n' \\
E \vdash \text{op} \ a' \downarrow n \text{ op } n' \\
E \vdash \text{true} \downarrow \text{true} \\
E \vdash \text{false} \downarrow \text{false} \\
E \vdash P \downarrow b \\
E \vdash !P \downarrow !b \\
E \vdash P \downarrow b \\
E \vdash P' \downarrow b' \\
E \vdash P \text{ op } P' \downarrow b \text{ op } b'
\end{aligned}
\]
**WHILE Statement Semantics**

- In a statement semantics, we are not looking for a resulting value, but for updates to variables in the environment.

- Example: assignment
  - If the right-hand side evaluates to a value \( n \)
  - Then the assignment generates a new environment where \( x \) maps to \( n \)

\[
E \vdash x := a \Downarrow E \{ x \mapsto n \}
\]

**WHILE Statement Semantics**

- sequences execute statements in order
- `skip` does not affect the environment
- `if` executes either first or second statement, depending on \( P \)
- `while` executes the body followed by the loop if \( P \) is `true`

\[
\begin{align*}
E \vdash a \Downarrow n & \quad \text{if } P \text{ then } S_1 \Downarrow E' \quad E' \vdash S_2 \Downarrow E'' \quad \text{if } P \text{ then } S_1; S_2 \Downarrow E'' \\
E \vdash \text{skip} \Downarrow E & \\
E \vdash P \Downarrow \text{true} & \quad E \vdash S_1 \Downarrow E' \quad E' \vdash S_2 \Downarrow E' \\
E \vdash P \Downarrow \text{false} & \quad E \vdash S_2 \Downarrow E' \quad E' \vdash S_1 \text{clsc} S_2 \Downarrow E' \\
E \vdash P \Downarrow \text{true} & \quad E \vdash S; \text{while } P \text{ do } S \Downarrow E' \\
& \quad E' \vdash \text{while } P \text{ do } S \Downarrow E' \\
E \vdash P \Downarrow \text{false} & \quad E' \vdash \text{while } P \text{ do } S \Downarrow E'
\end{align*}
\]
### WHILE Execution Example

1. \( \{ \} \vdash 5 \Downarrow 5 \) 
   by rule `eval-num`
2. \( \{ \} \vdash x := 5 \Downarrow \{ x \rightarrow 5 \} \) 
   by rule `ex-assign` on (1)
3. \( \{ x \rightarrow 5 \} \vdash x \Downarrow 5 \) 
   by rule `eval-var`
4. \( \{ x \rightarrow 5 \} \vdash 3 \Downarrow 3 \) 
   by rule `eval-num`
5. \( \{ x \rightarrow 5 \} \vdash x > 3 \Downarrow \text{true} \) 
   by rule `eval-rop` on (3),(4)
6. \( \{ x \rightarrow 5 \} \vdash 1 \Downarrow 1 \) 
   by rule `eval-num`
7. \( \{ x \rightarrow 5 \} \vdash y := 1 \Downarrow \{ x \rightarrow 5, x \rightarrow 1 \} \) 
   by rule `ex-assign` on (6)
8. \( \{ x \rightarrow 5 \} \vdash \text{if } x > 3 \text{ then } y := 1 \text{ else } y := 5 \Downarrow \{ x \rightarrow 5, x \rightarrow 1 \} \) 
   by rule `ex-iftrue` on (5),(7)
9. \( \{ } \vdash x := 5; \text{ if } x > 3 \text{ then } y := 1 \text{ else } y := 5 \Downarrow \{ x \rightarrow 5, x \rightarrow 1 \} \) 
   by rule `ex-seq` on (2),(8)