Big-Step Semantics
(introduction to Semantics)
"Semantics" = meaning/significance (ancient Greek)

mathematical of programs
why mathematical?

- rigor / precision
- brevity
- amenable to proof methods (e.g., proof by induction)
Approaches to modeling semantics

- **Operational semantics** — describe the execution of a program and its effects directly, as on an abstract machine.

- **Denotational semantics** — model language constructs as "denotations", i.e., values in a mathematical domain, and reason about those.

- **Axiomatic semantics** — describe a program in terms of logical predicates that are satisfied before and after its execution.
Type of Operational Semantics

- **Big-step** - evaluate an entire expression in one big step
  
  \( e \Downarrow v \): expression \( e \) evaluates to value \( v \)

- **Small-step** - evaluation is modeled as a series of small steps
  
  \( e \rightarrow e' \rightarrow e'' \rightarrow e''' \rightarrow \ldots \rightarrow v \)

  \( e \rightarrow* v \): \( e \) evaluate to \( v \) after a series of steps
How can we assert/prove that evaluations and computations proceed a certain way?

RULES!

i.e., specifications for how different assertions are related to each other

—would like to be able to use these rules to formally reason about programs (e.g. inductive proofs)
Inference Rules

form of a rule: \[ \text{[NAME]} \over \text{premise}_1, \ldots, \text{premise}_k \rightarrow \text{conclusion} \]

where each premise \( \vdash \) the conclusion is an assertion about a syntactic object

aka "judgement"

e.g., assertions: \( n \) is odd
\( e \rightarrow e' \)
\( e \| e \)
\( \vdash e : t \leftarrow \text{type of } e \)
Inference Rules

form of a rule: \[
[\text{NAME}] \quad \text{premise}_1, \ldots, \text{premise}_k \quad \text{[side conditions]} \quad \text{conclusion}
\]

meaning of a rule: if all premises (and side condition(s), if present) hold, then the conclusion holds

A rule with no premises is an axiom,
otherwise it is a proper rule
e.g., inference rules for even/odd integers over multiplication

\[ \text{MOD} 0 \quad x \text{ mod } 2 = 0 \]
\[ x \text{ is even} \]

\[ \text{MOD} 1 \quad x \text{ is odd} \]
\[ x \text{ mod } 2 = 1 \]

\[ \text{EVEN} \times \text{EVEN} \]
\[ x \text{ is even} \quad y \text{ is even} \]
\[ x \times y \text{ is even} \]

\[ \text{ODD} \times \text{EVEN} \]
\[ x \text{ is odd} \quad y \text{ is even} \]
\[ x \times y \text{ is even} \]

\[ \text{EVEN} \times \text{ODD} \]
\[ x \text{ is even} \quad y \text{ is odd} \]
\[ x \times y \text{ is even} \]

\[ \text{ODD} \times \text{ODD} \]
\[ x \text{ is odd} \quad y \text{ is odd} \]
\[ x \times y \text{ is odd} \]
Proof Trees

- We can derive a proof tree for a given assertion, where:
  - the root of the proof tree (its final conclusion) is the assertion we wish to prove
  - each subtree is a proof tree of the conclusion's premises

E.g.,

```

      leaves = axioms
    /     \
   P_1   P_3
   /     \        /   / \
  C_1   P_2  P_4  P_5
          /     \   / \
         C_3   C_4
          /     \   / \
         C    "root"
```
Derivation Strategies

- forward chaining / bottom-up construction
  - start w/ axioms and work towards conclusion
  - end goal may not be reachable — when to stop?

- backward chaining / top-down construction
  - start w/ conclusion and work "backwards" towards axioms
  - goal is known — can we derive it?
e.g., prove "21347 × 12345 is odd"

21347 mod 2 = 1

12345 mod 2 = 1

21347 is odd

12345 is odd

21347 × 12345 is odd
Big-step Semantics

- A big-step semantic rule makes an assertion about what a syntactic object would evaluate to on an abstract machine.

- State of the abstract machine (aka "configuration") includes:
  - The syntactic object to evaluate (e.g., expressions, statements, etc.)
  - The environment (e.g., variable → value mappings)
The Evaluation Relation: “\(\downarrow\)"

\(\downarrow\) is a relation over:
- syntactic objects
- environments
- possible results

i.e., \(\downarrow \in S \times \Sigma \times R\)

we write (in "mixfix" form): \(\langle s, o \rangle \downarrow r\)
"Simple Imperative Programming Language" (IMP)

```
arithmetic exps:   E ::= Integer | Var | E + E | E * E

boolean exps:     B ::= true | false | E < E | E > E | E = E

statements:       S ::= skip | Var := E | S_1 ; S_2 | if B then S_1 else S_2 | while B do S
```

var mappings constitute environment
Evaluation relations:

**arith exps**: \( \langle E, \sigma \rangle \downarrow _e v \)

**boolean exps**: \( \langle B, \sigma \rangle \downarrow _b b \)

**statements**: \( \langle S, \sigma \rangle \downarrow _s \sigma' \)

*we will overload \( \downarrow \) when it is clear by context which relation to use.*
**Arithmetic Expression Rules**

**LITERAL**
\[
\langle i, \sigma \rangle \downarrow i \quad \text{if } i \in \mathbb{Z}
\]

**VARIABLE**
\[
\sigma(x) = v \\
\langle x, \sigma \rangle \downarrow v
\]

x is a var, and \( x := v \in \sigma \)

**ARITH**
\[
\langle e_1, \sigma \rangle \downarrow v_1, \langle e_2, \sigma \rangle \downarrow v_2 \quad \Rightarrow \quad \langle e_1 \oplus e_2, \sigma \rangle \downarrow v_1 \oplus v_2
\]
Boolean Expression Rules

LITERAL \( \downarrow b \)

\( \text{if } b \in \{ \text{true, false} \} \)

REL

\( \downarrow_{e_1} v_1 \quad \downarrow_{e_2} v_2 \)

\( \downarrow_b v_1 \lor v_2 \)
Skip Rule

\[ \langle \text{skip}, \sigma \rangle \downarrow \sigma \]
Assignment Rule

\[ \text{ASSIGN} \quad \langle e, \sigma \rangle \Downarrow e \Downarrow \nu \]

\[ \langle x := e, \sigma \rangle \Downarrow \sigma (x := \nu) \]

Create new/update var mapping in env.
Sequencing $(S_1; S_2)$ Rule

\[
\text{SEQ} \quad \frac{\langle s_1, \sigma \rangle \downarrow \sigma' \quad \langle s_2, \sigma' \rangle \downarrow \sigma''}{\langle s_1; s_2, \sigma \rangle \downarrow \sigma''}
\]
if-statement Rules

**IF-T**
\[
\frac{\langle b, 0 \rangle \Downarrow b \text{ true} \quad \langle s_1, 0 \rangle \Downarrow 0'}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2, 0 \rangle \Downarrow 0'}
\]

**IF-F**
\[
\frac{\langle b, 0 \rangle \Downarrow b \text{ false} \quad \langle s_2, 0 \rangle \Downarrow 0'}{\langle \text{if } b \text{ then } s_1 \text{ else } s_2, 0 \rangle \Downarrow 0'}
\]
**while-statement Rules**

\[
\begin{align*}
\text{WHILE-F} & \quad \frac{\langle b, \sigma \rangle \downarrow \text{false}}{\langle \text{while } b \text{ do } S, \sigma \rangle \downarrow \sigma}
\end{align*}
\]

\[
\begin{align*}
\text{WHILE-T} & \quad \frac{\langle b, \sigma \rangle \downarrow \text{true} \quad \langle S, \sigma \rangle \downarrow \sigma' \quad \langle \text{while } b \text{ do } S, \sigma' \rangle \downarrow \sigma''}{\langle \text{while } b \text{ do } S, \sigma \rangle \downarrow \sigma''}
\end{align*}
\]
- prove that \( 3a + 4b = 39 \) in \( \sigma = \{ a := 5, b := 6 \} \)

- prove that \( \text{while } b \text{ do } s \) is equivalent to \( \text{if } b \text{ then } (s ; \text{while } b \text{ do } s) \text{ else skip} \)
prove that \( 3 \times a + 4 \times b = 39 \) in \( \sigma = \{ a := 5, b := 6 \} \)

\[
\begin{align*}
\text{LITERAL} & \quad \text{VAR} & \quad \text{LITERAL} & \quad \text{VAR} \\
\langle 3, 0 \rangle & \downarrow 3 & \langle a, 0 \rangle & \downarrow 5 \\
\langle 4, 0 \rangle & \downarrow 4 & \langle b, 0 \rangle & \downarrow 6 \\
\big< 3 \times a, 0 \big> & \downarrow 15 & \big< 4 \times b, 0 \big> & \downarrow 24 \\
\text{ARITH} & \quad \big< 3 \times a + 4 \times b, 0 \big> & \downarrow 39
\end{align*}
\]
prove that \( \text{while } b \text{ do } S \) is equivalent to \( \text{if } b \text{ then } (S \text{ ; while } b \text{ do } S) \text{ else skip} \)

- two statements \( S_1 \) and \( S_2 \) are equivalent \( (S_1 \sim S_2) \) if, for any two environments \( \sigma, \sigma' \),

\[
\langle S_1, \sigma \rangle \downarrow \sigma' \iff \langle S_2, \sigma \rangle \downarrow \sigma'
\]

bicategorical \( / \text{ iff} \)

- let \( W \) be "while \( b \) do \( S \)"; want to show:

\[
\langle W, \sigma \rangle \downarrow \sigma' \iff \langle \text{if } b \text{ then } (S \text{ ; } W) \text{ else skip}, \sigma \rangle \downarrow \sigma'
\]
- prove that \( \text{while } b \text{ do } S \) is equivalent to \( \text{if } b \text{ then } (S; \text{while } b \text{ do } S) \text{ else } \text{skip} \)

- consider \( \text{WHILE-F} \) and \( \text{WHILE-T} \) rules for \( W \)

\[
\text{WHILE-F} \quad \langle b, \sigma \rangle \Downarrow \text{false} \\
\quad \langle W, \sigma \rangle \Downarrow \sigma
\]

\[
\text{IF-F} \quad \langle b, \sigma \rangle \Downarrow \text{false} \\
\quad \langle \text{if } b \text{ then } (S; W) \text{ else } \text{skip}, \sigma \rangle \Downarrow \sigma
\]
- prove that \( \text{while } b \text{ do } s \) is equivalent to \( \text{if } b \text{ then } (s; \text{while } b \text{ do } s) \text{ else skip} \)

- consider \( \text{WHILE-F} \) and \( \text{WHILE-T} \) rules for \( W \)

\[
\begin{align*}
\text{WHILE-T} & \quad <b, \sigma> \Downarrow \text{true} \\
& \quad <s, \sigma> \Downarrow \sigma' \\
& \quad <w, \sigma'> \Downarrow \sigma''
\end{align*}
\]
- prove that \( \textbf{while } b \text{ do } S \) is equivalent to \( \text{if } b \text{ then } (S; \textbf{while } b \text{ do } S) \text{ else skip } \)

- consider \textbf{WHILE-F} and \textbf{WHILE-T} rules for \( W \)

\[
\begin{align*}
\text{IF-T} & \quad \frac{\langle b, \delta \rangle \downarrow \text{true}}{
\langle \text{if } b \text{ then } (S; W) \text{ else skip } \rangle \downarrow \delta'}
\end{align*}
\]

\[
\begin{align*}
\text{SEQ} & \quad \frac{\langle S, \delta \rangle \downarrow \delta'' \quad \langle W, \delta'' \rangle \downarrow \delta'}{
\langle S; W, \delta \rangle \downarrow \delta'}
\end{align*}
\]
Back to Interpreters

\[ \downarrow \equiv \text{eval} \]

\[ \sigma \equiv \text{env} \]

i.e., Big-step semantics also gives us a rigorous, concise way to specify the behavior of an interpreter/language!

e.g., dozens of pages of formal semantics vs. hundreds of English for specification of Java lang.