Mathematically describe a program's behavior

- rigorous/precise
- abstract/symbolic representation
- amenable to proof methods

Types of semantics
- Operational semantics - describe execution of a program
- Denotational semantics - model as another type of math object

Types of operational semantics
- Big step: \( e \downarrow v \) "\( e \) evaluates to \( v \)" (one big step)
- Small step: \( e \Rightarrow e' \Rightarrow e'' \Rightarrow ... \Rightarrow e \) show all steps

To assert/prove things about behavior of programs:
1. Make assertions about objects (syntax, typing, execution, etc.)
2. Connect these assertions
3. Build up into larger narratives

Judgment - type of thing we might want to assert
- \( n \) is odd \( \Rightarrow e \Rightarrow e' \) \( \Rightarrow v \Rightarrow e \Rightarrow e' \) \( e \) has type \( \mathbb{Z} \)

We assert/define judgments w/inference rules

\[
\frac{\text{premise}, ... \text{premise}}{\text{conclusion}}
\]

Axiom - rule w/no premises
Ex. even/odd

- O is even: 0 is even
- EVEN: n is odd
- LRCO: n+1 is even
- LRC: n is even
- n+1 is odd

EVL
- x is even
- x+y is even

EVR
- y is even
- x+y is even

OCPODD
- x is odd
- y is odd
- x*y is odd

Proof trees ("derivations")

End w/axioms

Each rule should be a rule from our system

Building a proof tree:

Can use forward chaining - start w/axioms - what can we derive?
May not be able to reach goal, may take a long time

Backward chaining - start w/conclusion

0 is even
1 is odd
2 is even
3 is odd
6 is even
7 is odd

IMP - A simple imperative language

Arithmetic exprs: 
E ::= n \times l E + E
^ Any int (0, 42,...) Variables

Boolean exprs: 
B ::= true | false | E = E

Statements: 
S ::= skip \leftarrow n => r ("pass" in Python)

| x := E |
| S |
| if B then S else S |
| while B do S |
Evaluation: \( \langle E, \sigma \rangle \downarrow v \) under environment \( \sigma \), \( E \) evals to \( v \).

\[ \langle B, \sigma \rangle \downarrow b \]

\[ \langle S, \sigma \rangle \uparrow \sigma' \]

\[ \langle \tau, \sigma \rangle \downarrow \tau \]

\[ \frac{n \in \mathbb{Z}}{\text{Var}} \quad \frac{\sigma(x) = n}{\langle x, \sigma \rangle \uparrow \sigma} \]

\[ \frac{e_1 \neq e_2 \Rightarrow \text{true \ or \ false}}{\langle e, \sigma \rangle \downarrow \langle e_1 \neq e_2 \rangle \sigma \sigma'} \]

\[ \frac{\text{true \ or \ false}}{\langle b, \sigma \rangle \downarrow b} \]

\[ \langle \text{skip}, \sigma \rangle \uparrow \sigma \]

\[ \langle x := e, \sigma \rangle \sigma' \]

\[ \langle \text{if } B \text{ then } S_1 \text{ else } S_2, \sigma \rangle \uparrow \sigma' \]

\[ \langle \text{while } B \text{ do } S, \sigma \rangle \downarrow \sigma' \]

\[ \text{Prove: } \langle (B+a) + (4+b), e \rangle \rightarrow 5, b \rightarrow 6; \sigma \uparrow 18 \]

\[ \langle 3+a, \sigma \rangle \downarrow \mathbb{V} \]

\[ \langle 4+b, \sigma \rangle \downarrow 10 \]

\[ \langle (3+a) + (4+b), \sigma \rangle \downarrow 15 \]
Prove: \( \text{While B do S} \) is equivalent to \( \text{if B then (S; while B do S) else skip} \)

WTS: \( \langle W, o \rangle \triangleright \langle W, o \rangle' \) \( \iff \) \( \text{if B then (S; W) else skip, o} \triangleright \langle W, o \rangle' \)


\[ \Rightarrow \text{Assume } \langle W, o \rangle \triangleright \langle W, o \rangle' \text{ so } \]

\[ \frac{\text{Skip}}{\langle B, o \rangle \triangleright \langle \text{skip, o} \rangle \triangleright \langle W, o \rangle' \triangleright \langle W, o \rangle'} \]

\[ \frac{\text{If } B \text{ then } \langle S; W \rangle \text{ else skip, o} \triangleright \langle W, o \rangle'}{\langle B, o \rangle \triangleright \langle \text{true} \rangle \triangleright \langle S; W, o \rangle' \triangleright \langle W, o \rangle'} \]

\[ \frac{\text{Assume } \langle \text{if B then (S; W) else skip, o} \rangle \triangleright \langle W, o \rangle'}{\langle B, o \rangle \triangleright \langle \text{true} \rangle \triangleright \langle S; W, o \rangle' \triangleright \langle W, o \rangle'} \]