Operational Semantics
Outline

What is semantics?

Operational Semantics
What is the meaning of a program?

Recall: aspects of a language

- **syntax**: the structure of its programs
- **semantics**: the meaning of its programs
### How to define the meaning of a program?

**Formal specifications**
- **denotational semantics**: relates terms directly to values
- **operational semantics**: describes how to evaluate a term
- **axiomatic semantics**: describes the effects of evaluating a term
- ...

**Informal/non-specifications**
- **reference implementation**: execute/compile program in some implementation
- **community/designer intuition**: how people “think” a program should behave
Advantages of a formal semantics

A formal semantics …

- is simpler than an implementation, more precise than intuition
  - can answer: is this implementation correct?

- supports the definition of analyses and transformations
  - prove properties about the language
  - prove properties about programs written in the language

- promotes better language design
  - better understand impact of design decisions
  - apply semantic insights to improve the language design (e.g. compositionality)
Outline

What is semantics?

Operational Semantics
What is operational semantics?

Defines the meaning of a program by describing **how it is evaluated**

**General strategy**

1. **identify machine state**: the state of evaluation
   - sometimes just the term being evaluated

2. define the **machine transitions**: relates old states to new states
   - typically using *inference rules*

3. define semantics in terms of machine transitions (this part is trivial)
Two styles of operational semantics

**Structural operational semantics** (a.k.a. small-step semantics)
- define transition relation ($\rightarrow$) representing one step of evaluation
- semantics is the reflexive, transitive closure of this relation ($\rightarrow^*$)

**Natural semantics** (a.k.a. big-step semantics)
- define transition relation (⇓) representing evaluation to a final state
- semantics is this relation directly

Argument for structural operational semantics:
- reason about intermediate steps
- reason about incomplete derivations
- systematic type soundness proof
- a bit more complicated
Structural operational semantics example

Define one-step evaluation relation

Step 1. identify machine state: \( Exp \)
Step 2. define transition relation: 
\[ e \mapsto e' \subseteq Exp \times Exp \]

Definition: \( e \mapsto e' \subseteq Exp \times Exp \)

\[
\begin{align*}
\text{not} \; \text{true} & \mapsto \text{false} & \text{not} \; \text{false} & \mapsto \text{true} \\
\text{if} \; \text{true} \; e_2 \; e_3 & \mapsto e_2 & \text{if} \; \text{false} \; e_2 \; e_3 & \mapsto e_3 \\
\end{align*}
\]

\[
\begin{align*}
\text{Not} & \quad e \mapsto e' \\
\text{If} & \quad \text{if} \; e \; e_2 \; e_3 \mapsto \text{if} \; e' \; e_2 \; e_3 \\
\end{align*}
\]

\} \quad \text{reduction rules} \quad \text{how to evaluate}
\} \quad \text{congruence rules} \quad \text{where to evaluate}
Defining the one-step transition

Terminology:
- **reduction rule**: replaces an expression by a “simpler” expression
- **redex** (reducible expression): an expression that matches a reduction rule
- **congruence rule**: describes where to find the next redex
- **value**: a final state, has no more redexes (e.g. `true` or `false`)

Observations:
- No rules for values – nothing left to do!
- Congruence rules define the **order of evaluation**
- The **meaning** of a term is the **sequence of steps** that reduce it to a final state
Completion of the semantics

**Semantics**: the reflexive, transitive closure of the one-step transition judgment

**Step 3.** Define the judgment \((\rightarrow^*)\) as follows

- just replace *state* by your machine state
- this last step is the same for any structural operational semantics!

**Definition**: \(s \rightarrow^* s' \subseteq \text{state} \times \text{state}\)

- **Reflex**: \(s \rightarrow^* s\)
- **Trans**: \(s \rightarrow s' \quad s' \rightarrow^* s'' \quad \Rightarrow \quad s \rightarrow^* s''\)
Full definition of the Boolean language

\[ e \in \text{Exp} \quad ::= \quad \text{true} \quad | \quad \text{false} \quad | \quad \text{not} \ e \quad | \quad \text{if} \ e \ e \ e \]

**Definition:** \( e \mapsto e' \subseteq \text{Exp} \times \text{Exp} \)

- \( \text{not} \ \text{true} \mapsto \text{false} \)
- \( \text{not} \ \text{false} \mapsto \text{true} \)
- \( \text{if} \ \text{true} \ e_2 \ e_3 \mapsto e_2 \)
- \( \text{if} \ \text{false} \ e_2 \ e_3 \mapsto e_3 \)

\[
\begin{align*}
\text{Not:} & \quad \frac{e \mapsto e'}{\text{not} \ e \mapsto \text{not} \ e'} \\
\text{If:} & \quad \frac{e \mapsto e'}{\text{if} \ e \ e_2 \ e_3 \mapsto \text{if} \ e' \ e_2 \ e_3}
\end{align*}
\]

**Definition:** \( e \mapsto^* e' \subseteq \text{Exp} \times \text{Exp} \)

- \( \text{Reflexive:} \quad \frac{}{e \mapsto^* e} \)
- \( \text{Transitive:} \quad \frac{e \mapsto e' \quad e' \mapsto^* e''}{e \mapsto^* e''} \)
Reduction sequences

Reduction sequence
Shows the sequence of states after each application of a reduction rule
- congruence rules indicate where to find next redex (underline)
- reduction rules indicate how to reduce it

Example reduction sequence

if (not true) (not false) (if true (not true) false) 
\[\mapsto\] if false (not false) (if true (not true) false) 
\[\mapsto\] if true (not true) false 
\[\mapsto\] not true 
\[\mapsto\] false