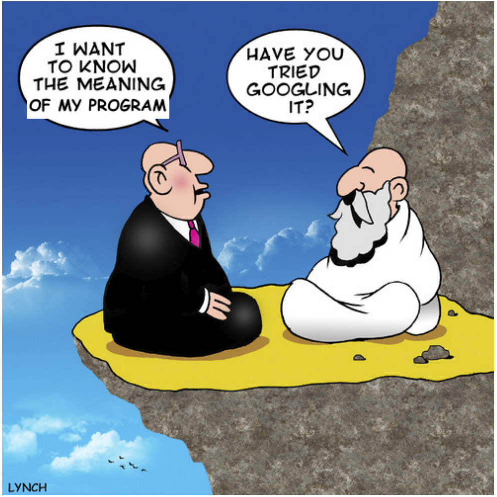
Operational Semantics

Outline

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?

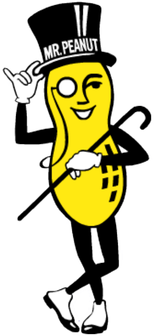
Formal specifications

- **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

Natural semantics (a.k.a. big-step semantics)

- define transition relation ($\Downarrow$) representing evaluation to a **final state**
- semantics is this relation directly

Structural operational semantics (a.k.a. small-step semantics)

- define transition relation ($\mapsto$) representing **one step** of evaluation
- semantics is the **reflexive, transitive closure** of this relation ($\mapsto^*$)

Argument for structural operational semantics:

+ reason about intermediate steps
+ reason about incomplete derivations
+ systematic type soundness proof
– a bit more complicated

Natural semantics example

$$
\begin{array}{rcl}
e \in \mathit{Exp} & ::= & \texttt{true} \\
& | & \texttt{false} \\
& | & \texttt{not}\ e \\
& | & \texttt{if}\ e\ e\ e
\end{array}
$$

Define one-step evaluation relation

Step 1. identify final state: Bool

Step 2. define evaluation relation:
$e \Downarrow b \ \subseteq\ \mathit{Exp} \times \mathit{Bool}$

Definition: $e \Downarrow b \ \subseteq\ \mathit{Exp} \times \mathit{Bool}$

$$\texttt{true} \Downarrow \texttt{true} \qquad \texttt{false} \Downarrow \texttt{false} \qquad \text{Not-T}\ \frac{e \Downarrow \texttt{true}}{\texttt{not}\ e \Downarrow \texttt{false}} \qquad \text{Not-F}\ \frac{e \Downarrow \texttt{false}}{\texttt{not}\ e \Downarrow \texttt{true}}$$

$$\text{If-T}\ \frac{e_1 \Downarrow \texttt{true} \qquad e_2 \Downarrow b}{\texttt{if}\ e_1\ e_2\ e_3 \Downarrow b} \qquad \text{If-F}\ \frac{e_1 \Downarrow \texttt{false} \qquad e_3 \Downarrow b}{\texttt{if}\ e_1\ e_2\ e_3 \Downarrow b}$$

Structural operational semantics example

$$\begin{array}{rcl} e \in \mathit{Exp} & ::= & \texttt{true} \\ & | & \texttt{false} \\ & | & \texttt{not}\ e \\ & | & \texttt{if}\ e\ e\ e \end{array}$$

Define one-step evaluation relation

Step 1. identify machine state: Exp

Step 2. define transition relation:
$e \mapsto e' \subseteq \mathit{Exp} \times \mathit{Exp}$

Definition: $e \mapsto e' \subseteq \mathit{Exp} \times \mathit{Exp}$

$$\texttt{not}\ \texttt{true} \mapsto \texttt{false} \qquad \texttt{not}\ \texttt{false} \mapsto \texttt{true}$$

$$\texttt{if}\ \texttt{true}\ e_2\ e_3 \mapsto e_2 \qquad \texttt{if}\ \texttt{false}\ e_2\ e_3 \mapsto e_3$$

reduction rules
how to evaluate

$$\text{Not}\ \frac{e \mapsto e'}{\texttt{not}\ e \mapsto \texttt{not}\ e'} \qquad \text{If}\ \frac{e \mapsto e'}{\texttt{if}\ e\ e_2\ e_3 \mapsto \texttt{if}\ e'\ e_2\ e_3}$$

congruence rules
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 ($\mapsto^*$) as follows

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

Definition: $s \mapsto^* s' \subseteq state \times state$

$$\text{Refl}\ \frac{}{s \mapsto^* s} \qquad\qquad \text{Trans}\ \frac{s \mapsto s' \qquad s' \mapsto^* s''}{s \mapsto^* s''}$$

Full definition of the Boolean language

$$e \in Exp \quad ::= \quad \mathtt{true} \mid \mathtt{false} \mid \mathtt{not}\ e \mid \mathtt{if}\ e\ e\ e$$

Definition: $e \mapsto e' \subseteq Exp \times Exp$

$$\mathtt{not}\ \mathtt{true} \mapsto \mathtt{false} \qquad \mathtt{not}\ \mathtt{false} \mapsto \mathtt{true}$$

$$\mathtt{if}\ \mathtt{true}\ e_2\ e_3 \mapsto e_2 \qquad \mathtt{if}\ \mathtt{false}\ e_2\ e_3 \mapsto e_3$$

$$\text{Not}\ \frac{e \mapsto e'}{\mathtt{not}\ e \mapsto \mathtt{not}\ e'} \qquad \text{If}\ \frac{e \mapsto e'}{\mathtt{if}\ e\ e_2\ e_3 \mapsto \mathtt{if}\ e'\ e_2\ e_3}$$

Definition: $e \mapsto^* e' \subseteq Exp \times Exp$

$$\text{Refl}\ \frac{}{e \mapsto^* e} \qquad \text{Trans}\ \frac{e \mapsto e' \qquad e' \mapsto^* e''}{e \mapsto^* e''}$$

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

$$
\begin{array}{rl}
 & \texttt{if } \underline{\texttt{(not true)}} \texttt{ (not false) (if true (not true) false)} \\
\mapsto & \underline{\texttt{if false (not false) (if true (not true) false)}} \\
\mapsto & \underline{\texttt{if true (not true) false}} \\
\mapsto & \underline{\texttt{not true}} \\
\mapsto & \texttt{false}
\end{array}
$$