Announcements

- Homework 3 due tonight

Feedback

- Homeworks challenging++
- Readings even harder++
- Wish there were more resources
- Pace

Questions?

Summary: Operational Semantics

- Precise specification of dynamic semantics
  - order of evaluation (or that it doesn't matter)
  - error conditions (sometimes implicitly, by rule applicability: "no applicable rule" = "get stuck")
- Simple and abstract (vs. implementations)
  - no low-level details such as stack and memory management, data layout, etc.
- Often not compositional (see while)
- Basis for many proofs about a language
  - Especially when combined with type systems!
- Basis for much reasoning about programs
- Point of reference for other semantics

On to Denotational Semantics
Commentary: Truth vs. Provability

Given a proof system, e.g., \(<e, \sigma \downarrow n>\) is **provably** if there exists a well-formed derivation with \(<e, \sigma \downarrow n>\) as its conclusion.
- “well-formed” = “every step in the derivation is a valid instance of one of the rules of inference for this system.”
- We would **like** truth and provability to be closely related.

Truth?

- We will not formally define “truth” yet.
- Instead we appeal to your intuition.
  - \(<2+2, \sigma \downarrow 4>\) — should be true
  - \(<2+2, \sigma \downarrow 5>\) — should be false
- Discussion Question: How might we define truth?

Discussion: Defining truth

Completeness

- A proof system (like our operational semantics) is **complete** if every true judgment is provable.
- If we replaced the subtract rule with:
  \[
  \begin{array}{c}
  \langle e_1, \sigma \downarrow n \rangle \\
  \langle e_2, \sigma \downarrow 0 \rangle \\
  \langle e_1 - e_2, \sigma \downarrow n \rangle
  \end{array}
  \]
- Our op. sem. would be **incomplete**:
  \(<-4-2, \sigma \downarrow 2>\) — true but not provable.

Consistency or Soundness

- A proof system is **consistent** (or **sound**) if every provable judgment is true.
- If we replaced the subtract rule with:
  \[
  \begin{array}{c}
  \langle e_1, \sigma \downarrow n_1 \rangle \\
  \langle e_2, \sigma \downarrow n_2 \rangle \\
  \langle e_1 - e_2, \sigma \downarrow n_1 + 3 \rangle
  \end{array}
  \]
- Our op. sem. would be **inconsistent** (or **unsound**):
  \(<-6-1, \sigma \downarrow 9>\) — false but provable.
Desired Traits

- Typically an operational semantics is always **complete** (unless you forget a rule)
- If you are not careful, however, your system may be **unsound**
- Usually that is **very bad**
  - A paper with an unsound type system is usually rejected
  - Papers often prove (sketch) that a system is sound
- In this class your work should be **complete and consistent** (e.g., on homework problems)

One-Slide Summary: Take-Away

- **Denotational semantics** is a formal way of assigning meanings to programs
  - mathematical objects
- Denotational semantics is **compositional**
  - meaning of (while \( b \) do \( c \)) depends on \( b \) and \( c \)
- Denotational semantics uses **fixed points** and **domains**

Dueling Semantics: Why Bother?

- Operational semantics is
  - simple, commonly used in (modern) research
  - not compositional
  \(<b, \sigma> \cup \text{true} \rightarrow <c, \sigma> \cup \sigma' \quad \text{while } b \text{ do } c, \sigma \cup \sigma'\)
  \(<\text{while } b \text{ do } c, \sigma > \cup \sigma'\)
- **Denotational semantics** is
  - purely mathematical (the meaning of an expression is a mathematical object)
  - compositional

Typical Student Reaction to Denotational Semantics

Recall truth vs. provability discussion

Denotational Semantics Learning Goals

When to use denotational semantics?

- base line of truth
- complex languages

Uses **fixed points** and **domains**

Remember SLAM and BLAST?
Denotational semantics
assigns meanings to programs

- The meaning will be a mathematical object
  - A number \( \in \mathbb{Z} \)
  - A boolean \( \in \{ \text{true}, \text{false} \} \)
  - A state transformer \( : \Sigma \to (\Sigma \cup \{ \bot \}) \)

New Notation

\( [ ] \) = “means” or “denotes”

- Examples:
  \( [\text{foo}] \) = “denotation of \text{foo}”
  \( [3 < 5] \) = true
  \( [3 + 5] \) = 8

- Sometimes we write
  \( A[\cdot] \) for \( A\text{exp} \), \( B[\cdot] \) for \( B\text{exp} \), \( C[\cdot] \) for \( C\text{om} \)

Rough Idea of Denotational Semantics

- The meaning of an arithmetic expression \( e \) in state \( \sigma \) is a number \( n \)
- Define \( A[e] \) as a function that maps the current state to an integer:
  \( A[\cdot] : \text{Aexp} \to (\Sigma \to \mathbb{Z}) \)
- The meaning of boolean expressions is defined in a similar way
  \( B[\cdot] : \text{Bexp} \to (\Sigma \to \{ \text{true}, \text{false} \}) \)
- All of these denotational function are total
  - Defined for all syntactic elements
  - For other languages it might be convenient to define the semantics only for well-typed elements

Denotational Semantics of Arithmetic Expressions

- We inductively define a function
  \( A[\cdot] : \text{Aexp} \to (\Sigma \to \mathbb{Z}) \)

- \( A[n] \sigma = \text{the integer denoted by literal } n \)
- \( A[x] \sigma = \sigma(x) \)
- \( A[e_1 + e_2] \sigma = A[e_1] \sigma + A[e_2] \sigma \)
- \( A[e_1 - e_2] \sigma = A[e_1] \sigma - A[e_2] \sigma \)
- \( A[e_1 \ast e_2] \sigma = A[e_1] \sigma \ast A[e_2] \sigma \)

- This is a total function (= defined for all expressions)

Denotational Semantics of Boolean Expressions

- We inductively define a function
  \( B[\cdot] : \text{Bexp} \to (\Sigma \to \{ \text{true}, \text{false} \}) \)

- \( B[\text{true}] \sigma = \text{true} \)
- \( B[\text{false}] \sigma = \text{false} \)
- \( B[\text{b}_1 \land \text{b}_2] \sigma = B[\text{b}_1] \sigma \land B[\text{b}_2] \sigma \)
- \( B[\text{e}_1 = \text{e}_2] \sigma = \text{if } A[\text{e}_1] \sigma = A[\text{e}_2] \sigma \text{ then true else false} \)
Seems Easy So Far

Denotational Semantics for Commands

Running a command $c$ starting from a state $\sigma$ yields another state $\sigma'$.

So, we try to define $C[c]$ as a function that maps $\sigma$ to $\sigma'$:

$$C[] : \text{Com} \rightarrow (\Sigma \rightarrow \Sigma)$$

Will this work?

Denotational Semantics of Commands

We try:

- $C[] : \text{Com} \rightarrow (\Sigma \rightarrow \Sigma)$
- $C[\text{skip}] : \sigma \mapsto \sigma$
- $C[x := e] : \sigma \mapsto \sigma[x := A[e]]$
- $C[c_1 ; c_2] : \sigma \mapsto C[c_2] (C[c_1] \sigma)$
- $C[\text{if } b \text{ then } c_1 \text{ else } c_2] : \sigma \mapsto$
  - if $B[b] \sigma$ then $C[c_1] \sigma$ else $C[c_2] \sigma$

Examples

- $C[x := 2; \ x := 1] \sigma = \sigma[x := 1]$
- $C[\text{if } \text{true then } x := 2; x := 1 \text{ else } ...] \sigma = \sigma[x := 1]$
- The semantics does not care about intermediate states (cf. "big-step")
- We haven't used $\bot$ yet
Denotational Semantics of Commands

\[ C[\text{while } b \text{ do } c] \sigma = ? \]

- Can’t use the same tricks as in operational semantics (directly)