Brutus Is An Honorable Man

- HW2 will not be due today.
- Homework X+1 will never be due until after I have returned Homework X to you.
- Normally this is never an issue, but I was sick yesterday and was hosting a party so I didn’t get it done.

Introduction to Denotational Semantics

Class Likes/Dislikes Survey

- + humor/style = 5
- + readings = 2
- - 5pm class = 2
- - hand-waving proofs
- - proving for the sake of proving
- - not do reading ⇒ no penalty

Dueling Semantics

- Operational semantics is
  - simple
  - of many flavors (natural, small-step, more or less abstract)
  - not compositional
  - commonly used in the real (modern research) world

- Denotational semantics is
  - mathematical (the meaning of a syntactic expression is a mathematical object)
  - compositional

- Denotational semantics is also called: fixed-point semantics, mathematical semantics, Scott-Strachey semantics

Typical Student Reaction To Denotation Semantics

Denotational Semantics Learning Goals

- DS is compositional (!)
- When should I use DS?
- In DS, meaning is a “math object”
- DS uses ⊥ (“bottom”) to mean non-termination
- DS uses fixed points and domains to handle while
  - This is the tricky bit
DS In The Real World

- ADA was formally specified with it
- Handy when you want to study non-trivial models of computation
  - e.g., “actor event diagram scenarios”, process calculi
- Nice when you want to compare a program in Language 1 to a program in Language 2

Deno-Challenge

- You may skip homework assignment 3 or 4 if you can find a post-1999 paper in a first- or second-tier PL conference that uses denotational semantics and you write me a two paragraph summary of that paper.

Foreshadowing

- Denotational semantics assigns meanings to programs
- The meaning will be a mathematical object
  - A number \( a \in \mathbb{Z} \)
  - A boolean \( b \in \{\text{true}, \text{false}\} \)
  - A function \( c : \Sigma \rightarrow (\Sigma \cup \{\text{non-terminating}\}) \)
- The meaning will be determined compositionally
  - Denotation of a command is based on the denotations of its immediate sub-commands (= more than merely syntax-directed)

New Notation

- ‘Cause, why not?
  - \([\square]\) = “means” or “denotes”
- Example:
  - \([\text{foo}]\) = “denotation of foo”
  - \([3 < 5]\) = true
  - \([3 + 5]\) = 8
- Sometimes we write \(A[\cdot]\) for arith, \(B[\cdot]\) for boolean, \(C[\cdot]\) for command

Rough Idea of Denotational Semantics

- The meaning of an arithmetic expression \(e\) in state \(\sigma\) is a number \(n\)
- So, we try to define \(A[e]\) as a function that maps the current state to an integer:
  - \(A[\cdot] : \text{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z})\)
- The meaning of boolean expressions is defined in a similar way
  - \(B[\cdot] : \text{Bexp} \rightarrow (\Sigma \rightarrow \{\text{true}, \text{false}\})\)
- 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
  \[ A : \text{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z}) \]
  \[ A[n] \sigma = \text{the integer denoted by literal } n \]
  \[ A[x] \sigma = \sigma(x) \]
  \[ A[e_1 + e_2] \sigma = A[e_1] \sigma + A[e_2] \sigma \]
  \[ A[e_1 * e_2] \sigma = A[e_1] \sigma \cdot A[e_2] \sigma \]

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

Denotational Semantics of Boolean Expressions

- We inductively define a function
  \[ B : \text{Bexp} \rightarrow (\Sigma \rightarrow \{\text{true}, \text{false}\}) \]
  \[ B[\text{true}] \sigma = \text{true} \]
  \[ B[\text{false}] \sigma = \text{false} \]
  \[ B[b_1 \land b_2] \sigma = B[b_1] \sigma \land B[b_2] \sigma \]
  \[ B[e_1 = e_2] \sigma = \text{if } A[e_1] \sigma = A[e_2] \sigma \text{ then true else false} \]

Denotational Semantics of 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 \( \sigma' \)
  \[ C[] : \text{Comm} \rightarrow (\Sigma \rightarrow \Sigma) \]

- Will this work? Bueller?

\( \bot = \text{Non-Termination} \)

- We introduce the special element \( \bot \) to denote a special resulting state that stands for non-termination
- For any set \( X \), we write \( X_\bot \) to denote \( X \cup \{\bot\} \)

Convention:
whenever \( f : X \rightarrow X_\bot \) we extend \( f \) to \( X_\bot \)
so that \( f(\bot) = \bot \)
- This is called strictness

Denotational Semantics of Commands

- We try:
  \[ C[] : \text{Comm} \rightarrow (\Sigma \rightarrow \Sigma) \]
  \[ C[\text{skip}] \sigma = \sigma \]
  \[ C[x := e] \sigma = \sigma[x := A[e] \sigma] \]
  \[ C[c_1; c_2] \sigma = C[c_2] (C[c_1] \sigma) \]
  \[ C[\text{if } b \text{ then } c_1 \text{ else } c_2] \sigma = \]
  \[ \text{if } B[b] \sigma \text{ then } C[c_1] \sigma \text{ else } C[c_2] \sigma \]
  \[ C[\text{while } b \text{ do } c] \sigma = ? \]
Examples

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

Denotational Semantics of WHILE

- Notation: \( W = C[\text{while } b \text{ do } c] \)
- Idea: rely on the equivalence (from notes last time) \( \text{while } b \text{ do } c = \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip} \)
- Try \( W(\sigma) = \text{if } B[b] \sigma \text{ then } W(C[c])[\sigma] \text{ else } \sigma \)
- This is called the unwinding equation
- It is not a good denotation of \( W \) because:
  - It defines \( W \) in terms of itself
  - It is not evident that such a \( W \) exists
  - It does not describe \( W \) uniquely
  - It is not compositional

More on WHILE

- The unwinding equation does not specify \( W \) uniquely
- Take \( C[\text{while true do skip}] \)
- The unwinding equation reduces to \( W(\sigma) = W(\sigma) \), which is satisfied by every function!
- Take \( C[\text{while } x \neq 0 \text{ do } x := x - 2] \)
- The following solution satisfies equation (for any \( \sigma' \))
  \[
  W(\sigma) = \{ \begin{align*}
  \sigma' & \text{ if } \sigma(x) = 2k \land \sigma(x) \geq 0 \\
  \bot & \text{ otherwise}
  \end{align*}
  \]

Denotational Game Plan

- Since WHILE is recursive
  - always have something like: \( W(\sigma) = F(W(\sigma)) \)
- Admits many possible values for \( W(\sigma) \)
- We will order them
  - With respect to non-termination = "least"
  - And then find the least fixed point
- LFP \( W(\sigma) = F(W(\sigma)) \) == meaning of "while"

WHILE \( k \)-steps Semantics

- Define \( W_k : \Sigma \rightarrow \Sigma_\bot \) (for \( k \in \mathbb{N} \)) such that
  - \( W_k(\sigma) = \{ \sigma' \}
    \begin{cases}
      \bot & \text{if } \exists k. W_k(\sigma) = \sigma' \\
      \uparrow & \text{if } \sigma(x) = 2k \land \sigma(x) \geq 0 \\
      \sigma' & \text{if } \sigma(x) = 2k \land \sigma(x) \geq 0 \\
      \bot & \text{otherwise}
    \end{cases}
  \)
- We can define the \( W_k \) functions as follows:
  - \( W_0(\sigma) = \bot \)
  - \( W_k(\sigma) = \{ W_{k-1}(C[c])[\sigma] \}
    \begin{cases}
      \bot & \text{if } B[b][\sigma] \text{ for } k \geq 1 \\
      \sigma & \text{otherwise}
    \end{cases}
\)

WHILE Semantics

- How do we get \( W \) from \( W_k \)?
  - \( W(\sigma) = \{ \sigma' \}
    \begin{cases}
      \bot & \text{if } \exists k. W_k(\sigma) = \sigma' \\
      \uparrow & \text{otherwise}
    \end{cases}
  \)
- This is a valid compositional definition of \( W \)
  - Depends only on \( C[c] \) and \( B[b] \)
- Try the examples again:
  - For \( C[\text{while true do skip}] \)
    - \( W_k(\sigma) = \bot \) for all \( k \), thus \( W(\sigma) = \bot \)
  - For \( C[\text{while } x \neq 0 \text{ do } x := x - 2] \)
    - \( W(\sigma) = \{ \sigma[x := 0] \}
      \begin{cases}
        \bot & \text{if } \sigma(x) = 2n \land \sigma(x) \geq 0 \\
        \sigma & \text{otherwise}
      \end{cases}
\)

More on WHILE

• The solution is not quite satisfactory because
  - It has an operational flavor (= "run the loop")
  - It does not generalize easily to more complicated semantics (e.g., higher-order functions)
• However, precisely due to the operational flavor this solution is easy to prove sound w.r.t. operational semantics

That Wasn’t Good Enough!?  

Simple Domain Theory

• Consider programs in an eager, deterministic language with one variable called "x"
  - All these restrictions are just to simplify the examples
• A state σ is just the value of x
  - Thus we can use $\mathbb{Z}$ instead of $\Sigma$
• The semantics of a command give the value of final x as a function of input x
  \[ C(\sigma) : \mathbb{Z} \rightarrow \mathbb{Z}_\perp \]

Examples - Revisited

• Take $C[\text{while true do skip}]$
  - Unwinding equation reduces to $W(x) = W(x)$
  - Any function satisfies the unwinding equation
  - Desired solution is $W(x) = \perp$
• Take $C[\text{while } x \neq 0 \text{ do } x := x^2]$
  - Unwinding equation: $W(x) = \text{if } x \neq 0 \text{ then } W(x^2) \text{ else } x$
  - Solutions (for all values $n, m \in \mathbb{Z}_\perp$):
    - $W(x) = \text{if } x \geq 0 \text{ then if } x \text{ even then } 0 \text{ else } n$
    - Solutions (for all values $n, m \in \mathbb{Z}_\perp$):
      - Desired solution: $W(x) = \text{if } x \geq 0 \land x \text{ even then } 0 \text{ else } \perp$

An Ordering of Solutions

• The desired solution is the one in which all the arbitrariness is replaced with non-termination
  - The arbitrary values in a solution are not uniquely determined by the semantics of the code
• We introduce an ordering of semantic functions
• Let $f, g \in \mathbb{Z} \rightarrow \mathbb{Z}_\perp$
• Define $f \sqsubseteq g$ as
  - $\forall x \in \mathbb{Z}, f(x) = \perp$ or $f(x) = g(x)$
  - A "smaller" function terminates at most as often, and when it terminates it produces the same result

Alternative Views of Function Ordering

• A semantic function $f \in \mathbb{Z} \rightarrow \mathbb{Z}_\perp$ can be written as $S_f \subseteq \mathbb{Z} \times \mathbb{Z}$ as follows:
  - set of "terminating" values for the function
• If $f \sqsubseteq g$ then
  - $S_f \subseteq S_g$ (and vice-versa)
  - We say that $g$ refines $f$
  - We say that $f$ approximates $g$
  - We say that $g$ provides more information than $f$
The "Best" Solution

- Consider again \( C \) [while \( x \neq 0 \) do \( x := x - 2 \)]
  - Unwinding equation:
    \[
    W(x) = \begin{cases} 
      0 & \text{if } x \neq 0 \text{ then } W(x - 2) \text{ else } x \\
      \bot & \text{if } x \neq 0 \text{ then } W(x - 2) \text{ else } \bot 
    \end{cases}
    \]
- Not all solutions are comparable:
  - The grammar for \( C \) does not contain "while \( b \) do \( c \)"
  - We can find such a (recursive) context for any looping construct

Fixed-Point Equations

- The meaning of a context is a semantic functional
  \[
  F : (\mathbb{Z} \to \mathbb{Z}) \to (\mathbb{Z} \to \mathbb{Z}) \text{ such that}
  \]
  \[
  F(C[w]) = F[w]
  \]
- For "while": \( C = \begin{cases} 
  c & \text{if } b \text{ then } c; \text{ else } \text{skip} \\
  \bot & \text{else } \bot
  \end{cases} \)
  - \( F \) depends only on \( [c] \) and \( [b] \)
- We can rewrite the unwinding equation for while
  - \( W(x) = \begin{cases} 
    0 & \text{if } [b] \text{ then } W([c]) \text{ else } x \\
    \bot & \text{else } \bot
  \end{cases} \)
  - or, \( W = F \wedge \) (by function equality)
  - or, \( W \) is a least fixed point

Can We Solve This?

- Good news: the functions \( F \) that correspond to contexts in our language have least fixed points!
- The only way \( F \) diverges is by invoking it
- If any such invocation diverges, then \( F \) diverges!
- It turns out: \( F \) is monotonic, continuous
  - Not shown here!

New Notation: \( \lambda \)

- \( \lambda x. e \)
  - an anonymous function with body \( e \) taking argument \( x \)
- Example: \( \text{double}(x) = x+x \)
  \[
  \text{double} = \lambda x. x+x
  \]
- Example: \( \text{allFalse}(x) = \text{false} \)
  \[
  \text{allFalse} = \lambda x. \text{false}
  \]
- Example: \( \text{multiply}(x,y) = x\cdot y \)
  \[
  \text{multiply} = \lambda x. \lambda y. x\cdot y
  \]
The Fixed-Point Theorem

- If $F$ is a semantic functional corresponding to a context in our language
  - $F$ is monotonic and continuous (we assert)
  - For any fixed-point $G$ of $F$ and $k \in \mathbb{N}$
    \[ F^k(\lambda x. \bot) \sqsubseteq G \]
  - The least of all fixed points is $\bigcup_k F^k(\lambda x. \bot)$

Proof (not detailed in the lecture):
1. By mathematical induction on $k$.
   - Base: $F^0(\lambda x. \bot) = \lambda x. \bot \sqsubseteq G$
   - Inductive: $F^{k+1}(\lambda x. \bot) = F(F^k(\lambda x. \bot)) \sqsubseteq F(G) = G$
2. Suffices to show that $\bigcup_k F^k(\lambda x. \bot)$ is a fixed-point
   \[ F(\bigcup_k F^k(\lambda x. \bot)) = \bigcup_k F^{k+1}(\lambda x. \bot) = \bigcup_k F^k(\lambda x. \bot) \]

WHILE Semantics

- We can use the fixed-point theorem to write the denotational semantics of while:
  \[ \text{while } b \text{ do } c \]
  \[ = \bigcup_k F^k(\lambda x. \bot) \]
  where $F^0 = \lambda x. \bot$
- Example: $\text{while } \text{true } \text{do } \text{skip} = \lambda x. \bot$
- Example: $\text{while } x \neq 0 \text{ then } x := x - 1$

Discussion

- We can write the denotational semantics but we cannot always compute it.
  - Otherwise, we could decide the halting problem
  - $H$ is halting for input $0$ iff $H[0] \neq \bot$
- We have derived this for programs with one variable
  - Generalize to multiple variables, even to variables ranging over richer data types, even higher-order functions: domain theory

Recall: Learning Goals

- DS is compositional
- When should I use DS?
- In DS, meaning is a “math object”
- DS uses $\bot$ (“bottom”) to mean non-termination
- DS uses fixed points and domains to handle while
  - This is the tricky bit

Can You Remember?

You just survived the hardest lecture in 615.
It’s all downhill from here.

Homework

- Homework 2 Due FRIDAY
- Homework 3 Out Today
  - Not as long as it looks - separated out every exercise sub-part for clarity.
  - Your denotational answers must be compositional (e.g., $W_n(m)$ or LFP)
- Read Winskel Chapter 6
- Read Hoare article
- Read Floyd article