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

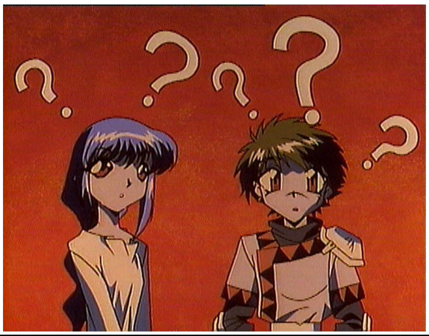
Class Likes/Dislikes Survey

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

Dueling Semantics

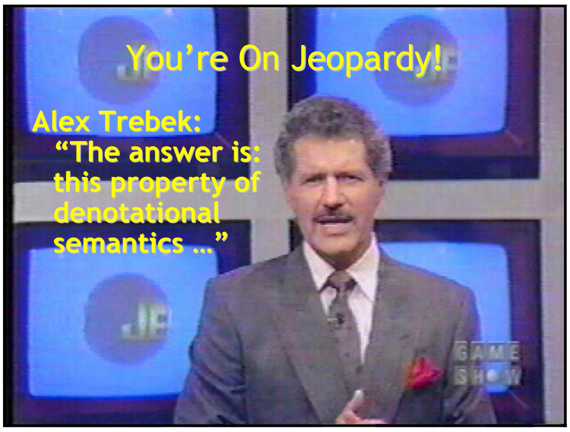
- Operational semantics is
  - simple
  - of many flavors (natural, small-step, more or less abstract)
  - not compositional
  - commonly used in the real (modern research) world
- Denotational semantics is
  - mathematical (the meaning of a syntactic expression is a mathematical object)
  - compositional
- 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 $\perp$ ("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 $c : \Sigma \to (\Sigma \cup \{\text{non-terminating}\})$
- 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?
  - $[\![\ ]\!]$ = "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 σ is a number n
- So, we try to define $A[\![e]\!]$ as a function that maps the current state to an integer:

$$A[\![\cdot]\!] : \text{Aexp} \to (\Sigma \to \mathbb{Z})$$

- 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

$$A[\![\cdot]\!] : \text{Aexp} \to (\Sigma \to \mathbb{Z})$$

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

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

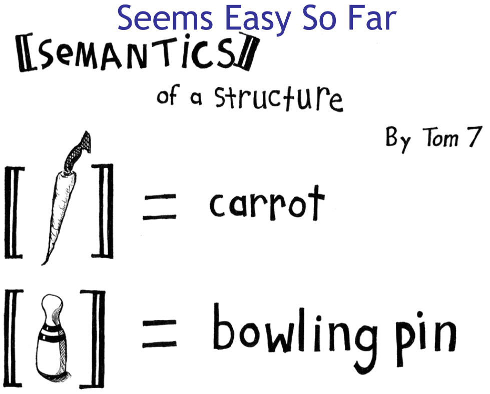
Denotational Semantics of Boolean Expressions

- We inductively define a function

$$B[\![\cdot]\!] : \text{Bexp} \to (\Sigma \to \{\textbf{true}, \textbf{false}\})$$

$B[\![\text{true}]\!]\sigma = \textbf{true}$
$B[\![\text{false}]\!]\sigma = \textbf{false}$
$B[\![b_1 \wedge b_2]\!]\sigma = B[\![b_1]\!]\ \sigma \wedge B[\![b_2]\!]\ \sigma$
$B[\![e_1 = e_2]\!]\sigma =$ if $A[\![e_1]\!]\ \sigma = A[\![e_2]\!]\ \sigma$ then **true** else **false**

Seems Easy So Far



Denotational Semantics for Commands

- Running a command c starting from a state σ yields another state σ'
- So, we try to define $C[\![c]\!]$ as a function that maps σ to σ'

$$C[\![\cdot]\!] : \text{Comm} \to (\Sigma \to \Sigma)$$

- Will this work? Bueller?

$\perp$ = Non-Termination

- We introduce the special element $\perp$ to denote a special resulting state that stands for non-termination
- For any set X, we write

$X_\perp$ to denote $X \cup \{\perp\}$

Convention:

whenever $f \in X \to X_\perp$ we extend f to $X_\perp \to X_\perp$ so that $f(\perp) = \perp$
- This is called strictness

Denotational Semantics of Commands

- We try:

$$C[\![\cdot]\!] : \text{Comm} \to (\Sigma \to \Sigma_\perp)$$

$C[\![\text{skip}]\!]\ \sigma = \sigma$
$C[\![x := e]\!]\ \sigma = \sigma[x := A[\![e]\!]\ \sigma]$
$C[\![c_1; c_2]\!]\ \sigma = C[\![c_2]\!]\ (C[\![c_1]\!]\ \sigma)$
$C[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!]\ \sigma =$
if $B[\![b]\!]\sigma$ then $C[\![c_1]\!]\sigma$ else $C[\![c_2]\!]\sigma$
$C[\![\text{while } b \text{ do } c]\!]\ \sigma = ?$

Examples

- C⟦x:=2; x:=1⟧ σ =
  σ[x := 1]
- C⟦if true then x:=2; x:=1 else …⟧ σ =
  σ[x := 1]
- The semantics does not care about intermediate states (cf. "big-step")
- We haven't used ⊥ yet

Denotational Semantics of WHILE

- Notation: W = C⟦while b do c⟧
- Idea: rely on the equivalence (from notes last time)
  while b do c ≈ if b then c; while b do c else skip
- Try
  W(σ) = if B⟦b⟧σ then W(C⟦c⟧σ) else σ
- 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⟦while true do skip⟧
- The unwinding equation reduces to W(σ) = W(σ), which is satisfied by every function!
- Take C⟦while x ≠ 0 do x := x – 2⟧
- The following solution satisfies equation (for any σ')

$$W(\sigma) = \begin{cases} \sigma[x := 0] & \text{if } \sigma(x) = 2k \wedge \sigma(x) \geq 0 \\ \sigma' & \text{otherwise} \end{cases}$$

Denotational Game Plan

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

WHILE k-steps Semantics

- Define W_k: $\Sigma \to \Sigma_\perp$ (for $k \in \mathbb{N}$) such that

$$W_k(\sigma) = \begin{cases} \sigma' & \text{if "while b do c" in state } \sigma \text{ terminates in fewer than } k \text{ iterations in state } \sigma' \\ \perp & \text{otherwise} \end{cases}$$

- We can define the W_k functions as follows:

$$W_0(\sigma) = \perp$$

$$W_k(\sigma) = \begin{cases} W_{k-1}(C[\![c]\!]\sigma) & \text{if } B[\![b]\!]\sigma \text{ for } k \geq 1 \\ \sigma & \text{otherwise} \end{cases}$$

WHILE Semantics

- How do we get W from W_k?

$$W(\sigma) = \begin{cases} \sigma' & \text{if } \exists k. W_k(\sigma) = \sigma' \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

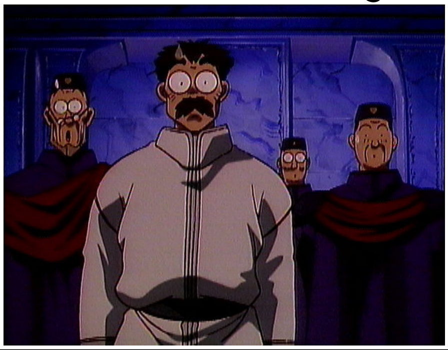
- This is a valid compositional definition of W
  - Depends only on C⟦c⟧ and B⟦b⟧
- Try the examples again:
  - For C⟦while true do skip⟧
    $W_k(\sigma) = \perp$ for all k, thus $W(\sigma) = \perp$
  - For C⟦while x ≠ 0 do x := x – 2⟧

$$W(\sigma) = \begin{cases} \sigma[x:=0] & \text{if } \sigma(x) = 2n \wedge \sigma(x) \geq 0 \\ \perp & \text{otherwise} \end{cases}$$

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 Σ
- The semantics of a command give the value of final x as a function of input x

$$C[\![c]\!] : \mathbb{Z} \to \mathbb{Z}_\perp$$

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, $m \in \mathbb{Z}_\perp$):
    $W(x) = \text{if } x \geq 0 \text{ then}$
    $\quad\quad \text{if } x \text{ even then } 0 \text{ else } n$
    $\quad \text{else } m$
  - Desired solution: $W(x) = \text{if } x \geq 0 \wedge x \text{ even then } 0 \text{ else } \perp$

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 $f, g \in \mathbb{Z} \to \mathbb{Z}_\perp$
- Define $f \sqsubseteq g$ as

  $\forall x \in \mathbb{Z}.\ f(x) = \perp \text{ or } f(x) = g(x)$
  - 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 $f \in \mathbb{Z} \to \mathbb{Z}_\perp$ can be written as $S_f \subseteq \mathbb{Z} \times \mathbb{Z}$ as follows:

$$S_f = \{ (x, y) \mid x \in \mathbb{Z},\ f(x) = y \neq \perp \}$$

  - 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 ≠ 0 do x := x - 2⟧
  - Unwinding equation:
    W(x) = if x ≠ 0 then W(x - 2) else x
- Not all solutions are comparable:
  W(x) = if x ≥ 0 then if x even then 0 else 1 else 2
  W(x) = if x ≥ 0 then if x even then 0 else ⊥ else 3
  W(x) = if x ≥ 0 then if x even then 0 else ⊥ else ⊥
  (last one is least and best)
- Is there always a least solution?
- How do we find it?
- *If only we had a general framework* for answering these questions …

Fixed-Point Equations

- Consider the general unwinding equation for while
  while b do c ≡ if b then c; while b do c else skip
- We define a context C (command with a hole)
  C = if b then c; • else skip
  while b do c ≡ C[while b do c]
  - The grammar for C does not contain "while b do c"
- We can find such a (recursive) context for

Fixed-Point Equations

- The meaning of a context is a semantic functional $F : (\mathbb{Z} \to \mathbb{Z}_\perp) \to (\mathbb{Z} \to \mathbb{Z}_\perp)$ such that
  F ⟦C[w]⟧ = F ⟦w⟧
- For "while": C = if b then c; • else skip
  F w x = if ⟦b⟧ x then w (⟦c⟧ x) else x
  - F depends only on ⟦c⟧ and ⟦b⟧
- We can rewrite the unwinding equation for while
  - W(x) = if ⟦b⟧ x then W(⟦c⟧ x) else x
  - or, W x = F W x for all x,
  - or, W = F W (by function equality)

Fixed-Point Equations

- The meaning of "while" is a solution for W = F W
- Such a W is called a fixed point of F
- We want the least fixed point
  - We need a general way to find least fixed points
- Whether such a least fixed point exists depends on the properties of function F
  - Counterexample: F w x = if w x = ⊥ then 0 else ⊥
  - Assume W is a fixed point
  - F W x = W x = if W x = ⊥ then 0 else ⊥
  - Pick an x, then if W x = ⊥ then W x = 0 else W x = ⊥
  - Contradiction. This F has no 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 w x uses w is by invoking it
- If any such invocation diverges, then F w x diverges!
- It turns out: F is monotonic, continuous
  - Not shown here!

New Notation: λ

- **λx. e**
  - an anonymous function with body e taking argument x
- Example: double(x) = x+x
  double = λx. x+x
- Example: allFalse(x) = false
  allFalse = λx. false
- Example: multiply(x,y) = x*y
  multiply = λx. λy. x*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

    $\sqcup_k F^k(\lambda x.\bot)$
- 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 $\sqcup_k F^k(\lambda x.\bot)$ is a fixed-point

     $F(\sqcup_k F^k(\lambda x.\bot)) = \sqcup_k F^{k+1}(\lambda x.\bot) = \sqcup_k F^k(\lambda x.\bot)$

WHILE Semantics

- We can use the fixed-point theorem to write the denotational semantics of while:

  $[\![\text{while b do c}]\!] = \sqcup_k F^k(\lambda x.\bot)$

  where $F\ f\ x = \text{if } [\![b]\!]\ x \text{ then } f\ ([\![c]\!]\ x) \text{ else } x$
- Example: $[\![\text{while true do skip}]\!] = \lambda x.\bot$
- Example: $[\![\text{while } x \neq 0 \text{ then } x := x - 1]\!]$
  - $F\ (\lambda x.\bot)\ x = \text{if } x = 0 \text{ then } x \text{ else } \bot$
  - $F^2\ (\lambda x.\bot)\ x = \text{if } x = 0 \text{ then } x \text{ else if } x - 1 = 0 \text{ then } x - 1 \text{ else } \bot$

    $= \text{if } 1 \geq x \geq 0 \text{ then } 0 \text{ else } \bot$
  - $F^3\ (\lambda x.\bot)\ x = \text{if } 2 \geq x \geq 0 \text{ then } 0 \text{ else } \bot$
  - $LFP_F = \text{if } x \geq 0 \text{ then } 0 \text{ else } \bot$
- Not easy to find the closed form for general LFPs!

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

Can You Remember?

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

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

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., $W_k(\sigma)$ or LFP)
- Read Winskel Chapter 6
- Read Hoare article
- Read Floyd article