Introduction to Denotational Semantics

Class Likes/Dislikes Survey
- “would change [the bijection question] to be one that still tested students’ recollection of set theory but that didn’t take as much time”
- “I liked the bijection proof in the homework. I thought it ended up being pretty neat.”
- “my guess is the student would benefit more from a rephrasing or alternate explanation”
- “I don’t need to hear the things explained in another way”

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

You’re On Jeopardy!
Alex Trebek: “The answer is this property of denotational semantics...”
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 the homework assignment of your choice if you can find a post-1995 paper in a first- or second-tier PL conference that uses denotational semantics.

Foreshadowing

- **Denotational semantics** assigns meanings to programs
- The meaning will be a mathematical object
  - A number $a \in \mathbb{Z}$
  - A boolean $b \in \{\text{true, false}\}$
  - 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 (= syntax-directed)

New Notation

- ‘Cause, why not?
- /FLF092h A[·] /FLF08Bh = “means” or “denotes”
- Example:
  - /FLF092h foo /FLF08Bh = “denotation of foo”
  - /FLF092h 3 < 5 /FLF08Bh = true
  - /FLF092h 3 + 5 /FLF08Bh = 8
- Sometimes we write A[·] for arith, B[·] for boolean, C[·] 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
  $$ B[\cdot] : \text{Bexp} \rightarrow (\Sigma \rightarrow \{\text{true, false}\}) $$
- 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
  $$ A[\cdot] : \text{Aexp} \rightarrow (\Sigma \rightarrow \mathbb{Z}) $$
- $A[n] \sigma = \text{the integer denoted by literal n}$
- $A[x] \sigma = \sigma(x)$
- $A[e_1 + e_2] \sigma = A[e_1] \sigma + A[e_2] \sigma$
- $A[e_1 - e_2] \sigma = A[e_1] \sigma - A[e_2] \sigma$
- $A[e_1 * e_2] \sigma = A[e_1] \sigma * A[e_2] \sigma$
- This is a **total function** (= defined for all expressions)
Denotational Semantics of Boolean Expressions

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

\[
\begin{align*}
B[true]_\sigma &= true \\
B[false]_\sigma &= false \\
B[b_1 \land b_2]_\sigma &= B[b_1]_\sigma \land B[b_2]_\sigma \\
B[e_1 = e_2]_\sigma &= \text{if } A[e_1]_\sigma = A[e_2]_\sigma \text{ then true else false}
\end{align*}
\]

Denotational Semantics for Commands

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

So, we try to define $C[c] : \text{Comm} \rightarrow (\Sigma \rightarrow \Sigma)$

\[
\begin{align*}
C[skip]_\sigma &= \sigma \\
C[x := e]_\sigma &= \sigma[x := A[e]_\sigma] \\
C[c_1; c_2]_\sigma &= C[c_2] (C[c_1]_\sigma) \\
C[if \ b \ then \ c_1 \ else \ c_2]_\sigma &= \text{if } B[b]_\sigma \text{ then } C[c_1]_\sigma \text{ else } C[c_2]_\sigma \\
C[while \ b \ do \ c]_\sigma &= ?
\end{align*}
\]

Examples

- $C[x:=2; x:=1]_\sigma = \sigma[x := 1]$
- $C[if \ true \ then \ x:=2; x:=1 \ else \ldots]_\sigma = \sigma[x := 1]$

The semantics does not care about intermediate states

- 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 last time)  
  \hspace{1cm} while $b$ do $c$ \iff $b$ then $c$; while $b$ do $c$ else skip  
- Try  
  $$W(\sigma) = \begin{cases}  
  \sigma & \text{if } B[b]\sigma \text{ then } W(C[c] \sigma) \text{ else } \sigma 
  \end{cases}$$  
- 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 composable

Denotational Game Plan

- Since WHILE is recursive  
  \hspace{1cm} always have something like: $W(\sigma) = F(W(\sigma))$  
- Admits many possible values for $W(\sigma)$  
- We will order them  
  \hspace{1cm} With respect to non-termination  
- And then find the least fixed point  
- LFP $W(\sigma) = F(W(\sigma))$ \iff meaning of “while”

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 $W$  
  \hspace{1cm} depends only on $C[c]$ and $B[b]$  
- Try the examples again:  
  \hspace{1cm} For $C[\text{while true do skip}]$, $W(\sigma) = \perp$  
  \hspace{1cm} For $C[\text{while } x \neq 0 \text{ do } x := x - 2]$, $W(\sigma) = \perp$  
- $W(\sigma) = \begin{cases}  
  \sigma[x := 0] & \text{if } \sigma(x) = 2k \land \sigma(x) \geq 0 
  \perp & \text{otherwise} 
  \end{cases}$

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 $\sigma'$)  
  $$W'(\sigma) = \begin{cases}  
  \sigma' & \text{if } B[b]\sigma \iff 2k \land \sigma(c) \geq 0 
  \perp & \text{otherwise} 
  \end{cases}$$

WHILE Semantics

- Define $W_k : \Sigma \rightarrow \Sigma_\perp$ (for $k \in \mathbb{N}$) such that  
  $$W_k(\sigma) = \begin{cases}  
  \sigma' & \text{if } "\text{while } b \text{ do } c" \text{ in state } \sigma \text{ terminates in fewer than } k \text{ iterations in state } \sigma' 
  \perp & \text{otherwise} 
  \end{cases}$$  
- We can define the $W_k$ functions as follows:  
  $$W_0(\sigma) = \perp$$  
  $$W_k(\sigma) = \begin{cases}  
  W_{k-1}(C[c] \sigma) & \text{if } B[b]\sigma \text{ for } k \geq 1 
  \perp & \text{otherwise} 
  \end{cases}$$

More on WHILE

- The solution is not quite satisfactory because  
  \hspace{1cm} It has an operational flavor  
  \hspace{1cm} 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 $\sigma$ 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 $C \vdash c : \mathbb{Z} \to \mathbb{Z}$

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:
    $W(x) = \begin{cases} 
    W(x) & \text{if } x \geq 0 \\
    \text{if } x \text{ even then } n \text{ else } m 
    \end{cases}$
  - Solutions (for all values $n, m \in \mathbb{Z}_\perp$):
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ and } x \text{ even then } 0 \text{ else } \perp 
    \end{cases}$
  - Desired solution:
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ and } x \text{ even then } 0 \text{ else } \perp 
    \end{cases}$

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:
  $S_f = \{ (x, y) \mid x \in \mathbb{Z}, f(x) = y = \perp \}$
  - A list of the "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 $C[\text{while } x \neq 0 \text{ do } x := x - 2]$
  - Unwinding equation:
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ then } W(x - 2) \text{ else } x 
    \end{cases}$
  - Not all solutions are comparable:
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ then } x \text{ even then } 0 \text{ else } 1 \text{ else } 2 
    \end{cases}$
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ then } x \text{ even then } 0 \text{ else } \perp \text{ else } 3 
    \end{cases}$
    $W(x) = \begin{cases} 
    \text{if } x \geq 0 \text{ then } x \text{ even then } 0 \text{ else } \perp \text{ else } \perp 
    \end{cases}$
    (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
  \[ \text{while } b \text{ do } c = \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ else skip} \]
- We define a context \( C \) (command with a hole)
  \[ C = \text{if } b \text{ then } c; \text{ \bullet } \text{ else skip} \]
- while \( b \) do \( c = [\text{while } b \text{ do } c] \)
  - The grammar for \( C \) does not contain "while \( b \) do \( c \)"
- We can find such a (recursive) context for any looping construct
  - Consider: \( \text{fact } n = \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fact } (n - 1) \)
  - \( C = \lambda x. \text{if } n = 0 \text{ then } 1 \text{ else } n \times \text{fact } (n - 1) \)
  - \( \text{fact } C \) [fact]

Fixed-Point Equations

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

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!

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 \( k \in \mathbb{N} \), \( P(\lambda x.\bot) \sqsubseteq G \)
  - The least of all fixed points is \( \sqsubseteq \lambda x. P(\lambda x.\bot) \)
- Proof (not detailed in the lecture):
  1. By mathematical induction on \( k \).
     - Base: \( P(\lambda x.\bot) = \lambda x.\bot \sqsubseteq G \)
     - Inductive: \( P^{n+1}(\lambda x.\bot) = \text{if } (P^n(\lambda x.\bot)) \sqsubseteq F(G) = G \)
  2. Suffices to show that \( \sqsubseteq \lambda x. P(\lambda x.\bot) \) is a fixed-point
     - \( F(\sqsubseteq \lambda x. P(\lambda x.\bot)) = \sqsubseteq \lambda x. P(\lambda x.\bot) \)

WHILE Semantics

- We can use the fixed-point theorem to write the denotational semantics of while:
  \[ [\text{while } b \text{ do } c] = \sqsubseteq \lambda x. P(\lambda x.\bot) \]
  where \( F x = \text{if } b \text{ then } f (\{c\} x) \text{ else } x \)
- Example: [while true do skip] = \( \lambda x.\bot \)
- Example: [while \( x \neq 0 \text{ then } x := x - 1 \) = \( F (\lambda x.\bot) x = \text{if } x = 0 \text{ then } x \neq 0 \text{ else } \bot \)
  - \( F^1 (\lambda x.\bot) x = \text{if } x = 0 \text{ then } x \neq 0 \text{ else } \bot \)
  - \( F^2 (\lambda x.\bot) x = \text{if } 1 \geq x > 0 \text{ then } 0 \text{ else } \bot \)
  - \( \text{LFP}_F = \text{if } x \geq 0 \text{ then } 0 \text{ else } \bot \)
- 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 $\left[ H \right] (0) \neq \bot$
- 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?**

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 Today
- 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., $W_0(\sigma)$ or LFP)
- Read Winskel Chapter 6
- Read Hoare article
- Read Floyd article