
Denotational Semantics

Lecture 7

CS 565

2/5/08

So far: Operational Semantics

- Small step
 - Good for implementing interpreter
- Big step / natural
 - Fewer rules
- Contextual
 - Isolating order and meaning

Denotational Semantics

- Brief detour from operational semantics
- Investigate other well-known semantics
 - First stop is denotational semantics
 - A.k.a. mathematical or Scott-Strachey sem.
 - Denotation refers to “meaning” and denotational semantics is the study of a program’s meaning in terms of its structure as mathematical objects
 - Domains (or sets) the primary vehicle

Functions as Sets

- Think of programs as functions input \rightarrow output
 - A function can be viewed as a set
 - Its input/output pairings can be collected into a set
 - The set is called the function's graph
 - $\text{graph}(f) = \{ (x, f(x)) \mid x \in \text{Dom}(f) \}$
 - We understand things like function application and composition in terms of graphs:
 - $f(a) = b$ iff $(a, b) \in \text{graph}(f)$
 - $\text{graph}(g \circ f) = \{ (x, z) \mid x \in \text{Dom}(f), \exists y \in \text{Dom}(g) \text{ s.t.}, (x, y) \in \text{graph}(f) \text{ and } (y, z) \in \text{graph}(g) \}$

Semantic Domains

- For now think of a semantic domain as a set
 - Product domains: $A \times B$
 - Sum domain: $A + B$
 - Function domain: $A \rightarrow B$
 - Lifted domains: A_{\perp} where $A_{\perp} = A \cup \{\perp\}$
 - read \perp as “bottom” denoting non-termination or “no value”, c.f commands
- Accompanying a domain are set of operations that map elements in one domain to elements in another.
- A domain along with its operations is called a semantic algebra.

Example

- Natural numbers:
 - Domain $\text{Nat} = \mathbb{N}$
 - Operations:
 - zero
 - one
 - ...
 - plus: $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
 - ...

Denotational Definitions

- Consists of three parts:
 - an abstract syntax tree
 - semantic algebras
 - a valuation function that maps syntactic phrases in the abstract syntax to elements in the domains specified by the semantic algebra

Denotational Semantics for IMP

- Semantic (Valuation) Functions:
 - $\mathcal{A}: AExp \rightarrow (\Sigma \rightarrow \mathbb{N})$
 - $\mathcal{B}: Bexp \rightarrow (\Sigma \rightarrow \mathbb{T})$
 - $\mathcal{C}: Com \rightarrow (\Sigma \rightarrow \Sigma)$
- $\sigma \in \Sigma = Var \rightarrow Value$
- Functions are described through graphs
 - $f(a)=b$ iff $(a,b) \in g$

Denotational Definitions for AExp

- $\mathcal{A}[[n]]\sigma = n$
 - $\mathcal{A}[[n]] = \{ (\sigma, n) \mid \sigma \in \Sigma \}$
- $\mathcal{A}[[x]]\sigma = \sigma(x)$
 - $\mathcal{A}[[x]] = \{ (\sigma, \sigma(x)) \mid \sigma \in \Sigma \}$
- $\mathcal{A}[[e_1 + e_2]]\sigma = \mathcal{A}[[e_1]]\sigma + \mathcal{A}[[e_2]]\sigma$
 - $\mathcal{A}[[e_1 + e_2]] = \{ (\sigma, n_1 + n_2) \mid (\sigma_1, n_1) \in \mathcal{A}[[e_1]] \text{ \& } (\sigma_2, n_2) \in \mathcal{A}[[e_2]] \}$

Denotational Definitions for BExp

- $\mathcal{B}[\text{true}]\sigma = \text{true}$
 - $\mathcal{B}[\text{true}] = \{ (\sigma, \text{true}) \mid \sigma \in \Sigma \}$

- $\mathcal{B}[\text{false}]\sigma = \text{false}$
 - $\mathcal{B}[\text{false}] = \{ (\sigma, \text{false}) \mid \sigma \in \Sigma \}$

□ $\mathcal{B}[\mathbf{b}_1 \wedge \mathbf{b}_2]\sigma = \mathcal{B}[\mathbf{b}_1]\sigma \wedge_{\mathcal{T}} \mathcal{B}[\mathbf{b}_2]\sigma$

■ $\mathcal{B}[\mathbf{b}_1 \wedge \mathbf{b}_2]=$

$$\{ (\sigma, \mathbf{t}_1 \wedge_{\mathcal{T}} \mathbf{t}_2) \mid \sigma \in \Sigma \ \& \ (\sigma, \mathbf{t}_1) \in \mathcal{B}[\mathbf{b}_1] \ \& \ (\sigma, \mathbf{t}_2) \in \mathcal{B}[\mathbf{b}_2] \}$$

□ $\mathcal{B}[\mathbf{e}_1 = \mathbf{e}_2]\sigma = \mathcal{A}[\mathbf{e}_1]\sigma = \mathcal{A}[\mathbf{e}_2] \sigma$

■ $\mathcal{B}[\mathbf{e}_1 = \mathbf{e}_2]=$

$$\{ (\sigma, \text{true}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[\mathbf{e}_1]\sigma = \mathcal{A}[\mathbf{e}_2]\sigma \} \cup$$

$$\{ (\sigma, \text{false}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[\mathbf{e}_1]\sigma \neq \mathcal{A}[\mathbf{e}_2]\sigma \}$$

Denotational Definitions for CExp

- $\mathcal{C}[\text{skip}]\sigma = \sigma$
 - $\mathcal{C}[\text{skip}] = \{ (\sigma, \sigma) \mid \sigma \in \Sigma \}$
- $\mathcal{C}[x := e]\sigma = \sigma[x \mapsto A[e]\sigma]$
 - $\mathcal{C}[x := e] = \{ (\sigma, \sigma[x \mapsto n]) \mid \sigma \in \Sigma \ \& \ n = A[e]\sigma \}$
- $\mathcal{C}[c_1; c_2]\sigma = \mathcal{C}[c_2]\mathcal{C}[c_1]\sigma$
 - $\mathcal{C}[c_1; c_2] = \{ (\sigma, \sigma'') \mid (\sigma, \sigma') \in \mathcal{C}[c_1] \ \& \ (\sigma', \sigma'') \in \mathcal{C}[c_2] \}$
 - $= \mathcal{C}[c_2] \circ \mathcal{C}[c_1]$

□ $\mathcal{C}[\text{if } b \text{ then } c_1 \text{ else } c_2]\sigma =$

$\mathcal{B}[b]\sigma = \text{true} \Rightarrow \mathcal{C}[c_1]\sigma ;$

$\mathcal{B}[b]\sigma = \text{false} \Rightarrow \mathcal{C}[c_2]\sigma$

■ $\mathcal{C}[\text{if } b \text{ then } c_1 \text{ else } c_2] = \{ (\sigma, \sigma') \mid \mathcal{B}[b]\sigma = \text{true} \ \& \ (\sigma, \sigma') \in \mathcal{C}[c_1] \} \cup \{ (\sigma, \sigma') \mid \mathcal{B}[b]\sigma = \text{false} \ \& \ \sigma' \in \mathcal{C}[c_2] \}$

While Loops

- First attempt:

- $\mathcal{C}[\text{while } b \text{ do } c]\sigma =$

- $\mathcal{B}[b]\sigma = \text{true} \Rightarrow \mathcal{C}[\text{while } b \text{ do } c] \mathcal{C}[c] \sigma;$

- $\mathcal{B}[b]\sigma = \text{false} \Rightarrow \sigma$

- Not sound: the meaning of function is defined in terms of itself

- Need a way of defining meaning of a recursive definition without recursion

Recursive Definition

- $\llbracket \text{while } B \text{ do } C \rrbracket =$
 $f \llbracket B \rrbracket, \llbracket C \rrbracket (\llbracket \text{while } B \text{ do } C \rrbracket)$
 - $w = f(w)$

- Is there a w that solves the equation?
 - Intuition: use successive approximations
 - Step from recursive to iterative

Example

- $\llbracket \text{while } X > 0 \text{ do } (Y := X * Y; X := X - 1) \rrbracket$
 - Defines f, b, c
- State = $Z \times Z$ pairs of integers
- $D = \text{State} \rightarrow \text{State}$
- We seek a minimal solution for $w = f(w)$
with $f: D \rightarrow D$ defined by
 - $f(w)(x, y) = (x, y)$ if $x \leq 0$
 $w(x-1, x*y)$ if $x > 0$

Successive Approximations

- Partial order \sqsubseteq on \mathcal{D} (State \rightarrow State)
 - $w \sqsubseteq w'$ iff (State is (x, y))
 - w is defined at (x, y) then so is w' and moreover $w(x, y) = w'(x, y)$
 - $\text{graph}(w) \subseteq \text{graph}(w')$
- Least element $\perp \in \mathcal{D}$ w.r.t. \sqsubseteq
 - totally undefined partial function ($\perp \sqsubseteq w \forall w \in \mathcal{D}$)

-
- Apply f successively, build up a sequence of partial functions w_0, w_1, w_2, \dots

- $w_0 = \perp, w_{n+1} = f(w_n)$

- $w_1(x,y) = f(\perp)(x,y) = (x,y)$ if $x \leq 0$
undefined if $x \geq 1$

- $w_2(x,y) = f(w_1)(x,y) = (x,y)$ if $x \leq 0$
 $(0,y)$ if $x = 1$
undefined if $x \geq 2$

-
- | | | |
|---|----------------------------------|---------------|
| □ | $w_3(x,y) = f(w_2)(x,y) = (x,y)$ | if $x \leq 0$ |
| | $(0,y)$ | if $x = 1$ |
| | $(0,2*y)$ | if $x = 2$ |
| | undefined | if $x \geq 3$ |

 - | | | |
|---|----------------------------------|---------------|
| □ | $w_4(x,y) = f(w_3)(x,y) = (x,y)$ | if $x \leq 0$ |
| | $(0,y)$ | if $x = 1$ |
| | $(0,2*y)$ | if $x = 2$ |
| | $(0,6*y)$ | if $x = 3$ |
| | undefined | if $x \geq 4$ |

 - | | | |
|---|------------------------------------|----------------|
| □ | General: $w_n(x,y) = (x,y)$ | if $x \leq 0$ |
| | $(0,(!x)*y)$ | if $0 < x < n$ |
| | undefined | if $x \geq n$ |

-
- Union of all w_n is w_∞ s.t.

$$w_\infty(x,y) = (x,y) \quad \text{if } x \leq 0$$

$$(0, (!x)^*y) \quad \text{if } x > 0$$

- w_∞ is a fixed point of f

$$f(w_\infty)(x,y) = (x,y) \quad \text{if } x \leq 0$$

$$w_\infty(x-1, x^*y) \quad \text{if } x > 0 \text{ (def. of } w_\infty)$$

$$= (x,y) \quad \text{if } x \leq 0$$

$$(0, 1^*y) \quad \text{if } x = 1$$

$$(0, !(x-1)^*x^*y) \quad \text{if } x > 1$$

$$= w_\infty$$

Least Fixpoints

- In fact, w is the least fixed point of f
 - For all $w \in D$, $w = f(w) \Rightarrow w_\infty \sqsubseteq w$
- Can we generalize this?
 - Is there always a fixed point for a recursive definition?
 - Can we determine a least one?
 - How can we determine it?