

Denotational Semantics

- Winskel Ch. 5

1

Denotation Semantics

- The most abstract of the approaches covered in this class
 - Quite powerful, kind of heavy on the math
 - We will only consider the basic issues for IMP
- Key idea: semantics as **partial functions**
 - Expression: function mapping states to values
 - Statement: function mapping states to states

2

Notation

- States: Σ - set of states
- Values: \mathbb{N} is the set of integers, \mathbb{T} is the set $\{\text{true}, \text{false}\}$
- Program constructs: sets ArithExp, BoolExp, Stmt
- $\mathcal{E} : \text{ArithExp} \rightarrow (\Sigma \rightarrow \mathbb{N})$
- $\mathcal{B} : \text{BoolExp} \rightarrow (\Sigma \rightarrow \mathbb{T})$
- $\mathcal{C} : \text{Stmt} \rightarrow (\Sigma \rightarrow \Sigma)$

3

Notation

- If c is a stmt, $\mathcal{E}[c]$ is a function $\Sigma \rightarrow \Sigma$ that tells how c changes the state
 - Actually, it should look like this: $\mathcal{C}[c]$
 - c denotes the function $\mathcal{E}[c]$, and the function is a denotation of c
- If ae is an arithmetic expression, $\mathcal{E}[ae]$ is a function $\Sigma \rightarrow \mathbb{N}$ that gives the value of ae for different states
- Similarly for $\mathcal{B}[be]$: it is $\Sigma \rightarrow \mathbb{T}$

4

Denotations for Expressions

$$\mathcal{E}[n](\sigma) = n$$

$$\mathcal{E}[X](\sigma) = \sigma(X)$$

$$\mathcal{E}[ae_1 + ae_2](\sigma) = \mathcal{E}[ae_1](\sigma) + \mathcal{E}[ae_2](\sigma)$$

...

→ syntax → math

$$\mathcal{B}[\text{true}](\sigma) = \text{true} \quad \mathcal{B}[\text{false}](\sigma) = \text{false}$$

$$\mathcal{B}[ae_1 = ae_2](\sigma) = (\mathcal{E}[ae_1](\sigma) = \mathcal{E}[ae_2](\sigma))$$

...

5

Denotations for Statements

$$\mathcal{E}[\text{skip}](\sigma) = \sigma$$

$$\mathcal{E}[X := ae](\sigma) = \sigma[X \leftarrow \mathcal{E}[ae](\sigma)] \quad \text{i.e. } \sigma[\mathcal{E}[ae](\sigma)/X]$$

$$\mathcal{E}[c_1; c_2](\sigma) = \mathcal{E}[c_2](\mathcal{E}[c_1](\sigma))$$

$$\mathcal{E}[\text{if } b \text{ then } c_1 \text{ else } c_2](\sigma) = \begin{cases} \mathcal{E}[c_1](\sigma) & \text{if } \mathcal{B}[b](\sigma) \\ \mathcal{E}[c_2](\sigma) & \text{else} \end{cases}$$

$$\mathcal{E}[\text{while } b \text{ do } c](\sigma) = \begin{cases} \sigma & \text{if not } \mathcal{B}[b](\sigma) \\ \mathcal{E}[\text{while } b \text{ do } c](\mathcal{E}[c](\sigma)) & \text{else} \end{cases}$$

6

So What?

- Axiomatic, operational, denotational: why all these different notations for the same thing?
- Operational: relatively low-level step-by-step operation, tied to the language
- Axiomatic: pre/post conditions in first-order logic, has the feel of a proof
- Denotational: uses familiar and powerful mathematical objects (functions)

7

Compositionality

- One of the advantages of denotational semantics is that it is **compositional**
 - The semantics is defined only in terms of substatements
- Operational semantics is not compositional

$\langle b, \sigma \rangle \rightarrow \text{true} \quad \langle c, \sigma \rangle \rightarrow \sigma' \quad \langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''$
 $\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma''$

8

Language Extension

- Nested blocks (but no procedures)

```
begin int x; x := 0;
  begin int x; x := 1; end;
  write x; end; // should print 0
```

At end, go back to the state before begin? But

```
begin int x; int y; x := 0; y := 0;
  begin int x; x := 1; y := 1; end;
  write x; write y; end;
```

should print 01, not 00

9

Possible Solution: Split the State

- ϵ : (environment) maps each variable to its current location
 - λ : (store) maps each location to a value
- $$\mathcal{E}[\text{skip}](\epsilon, \lambda) = \lambda \quad \mathcal{E}[x := 5](\epsilon, \lambda) = \lambda[\epsilon(x) \leftarrow 5]$$
- $$\mathcal{D}[\text{int } x](\epsilon) = \epsilon[x \leftarrow \text{new_location}(\epsilon)]$$
- $$\mathcal{E}[\text{begin } ds; cs; \text{end}](\epsilon, \lambda) = \mathcal{E}[cs](\mathcal{D}[ds](\epsilon), \lambda)$$
- $$\mathcal{E}[c_1; c_2](\epsilon, \lambda) = \mathcal{E}[c_2](\epsilon, \mathcal{E}[c_1](\epsilon, \lambda))$$
- Since c_1 may be a block, the execution of c_2 uses the "upper-level" environment

10

Semantics of Loops

- Unfortunately, our earlier definition for loops is not quite right
- $$\mathcal{E}[\text{while } b \text{ do } c](\sigma) = \begin{cases} \sigma & \text{if not } \mathcal{B}[b](\sigma) \\ \mathcal{E}[c](\sigma) & \text{else } \mathcal{E}[\text{while } b \text{ do } c](\mathcal{E}[c](\sigma)) \end{cases}$$
- Let w be "while true do skip"
- $$\mathcal{E}[w](\sigma) = \begin{cases} \sigma & \text{if not } \mathcal{B}[w](\sigma) \\ \mathcal{E}[w](\mathcal{E}[\text{skip}](\sigma)) & \text{else } \mathcal{E}[w](\mathcal{E}[\text{skip}](\sigma)) \end{cases}$$
- $\mathcal{E}[w](\sigma) = \mathcal{E}[w](\sigma)$??? any function satisfies this

11

An Even Bigger Problem

- The function for "while" is defined in terms of itself (recursive definition)
- $$f(\sigma) = \{ \sigma \mid \mathcal{B}[b](\sigma) = \text{false} \} \cup \{ f(\mathcal{E}[c](\sigma)) \mid \mathcal{B}[b](\sigma) = \text{true} \}$$
- This is not compositional ...
 - Solution: use standard math machinery

12

Functionals and Fixed Points

- Functional: maps functions to functions

$$\tau[f](\sigma) = \{ \sigma \mid \mathcal{B}[b](\sigma) = \text{false} \} \cup \{ f(\mathcal{E}[c](\sigma)) \mid \mathcal{B}[b](\sigma) = \text{true} \}$$

- $\tau: (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$
- Consider any function f such that $\tau[f]=f$
 - Standard math: f is a **fixed point** of τ
- A fixed point of τ can be thought of as representing the semantics of "while"

13

Partial Orders

- Partial order P : a set on which we have defined a binary relation \leq (also \sqsubseteq):
 - reflexive: $p \leq p$
 - transitive: $p \leq q$ and $q \leq r$ imply $p \leq r$
 - anti-symmetric: $p \leq q$ and $q \leq p$ imply $p = q$
- Uses in programming languages
 - Definition of semantics
 - Definition of program analyses for compilers
 - Intuition: $p \leq q$ means that p "approximates" q - i.e., p represents "less info" than q

14

More Definitions

- Bottom of a partial order: $\perp \leq p$ for all p
 - Intuitively, it means "no information"
- Monotonic $f: P \rightarrow P$
 - if $p \leq q$, then $f(p) \leq f(q)$
- Least fixed point x of $f: P \rightarrow P$
 - $f(x) = x$ - i.e. x is a fixed point of f
 - for any fixed point z of f , $x \leq z$

15

Application to Denotational Semantics

- Define a special "bottom" state \perp
 - helps to represent "undefined" behavior
 - The set of states becomes $\Sigma' = \Sigma \cup \{\perp\}$
- $\mathcal{E}[c](\sigma) = \perp$ means that the semantics of c is not defined for starting state σ
- $\mathcal{E}[c](\perp) = \perp$
- Define a function $\Omega : \Sigma' \rightarrow \Sigma'$ such that $\Omega(\sigma) = \perp$ for all σ
 - Represents completely undefined semantics

16

More on the Bottom State

- $\mathcal{B}[be]$ and $\mathcal{E}[ae]$ are not defined for the bottom state
 - We cannot evaluate an expression if we know nothing about the state
- The bottom state plays two roles
 - "right now I don't know the semantics, so I will approximate the result with bottom"
 - In the fully computed semantics (in the least fixed point), bottom represents non-termination or run-time error

17

More on the Bottom State

- In general, $\mathcal{B}[be]$ and $\mathcal{E}[ae]$ may return the bottom state due to run-time errors
 - Examples: division by zero; overflow
 - In our language IMP, this is not possible: no division; the domain is all integers
- If $\mathcal{B}[be]$ or $\mathcal{E}[ae]$ return bottom, the semantic function for the surrounding statement should also return bottom
 - e.g. since $\mathcal{E}[5/0](\sigma) = \perp$, $\mathcal{E}[x := 5/0](\sigma) = \perp$

18

Partial Orders

- Σ' is a partial order, defined as $\perp \leq \sigma$
 - i.e. "bottom" is an approximation of any state
 - For any $f: \Sigma' \rightarrow \Sigma'$, since $f(\perp) = \perp$, f is monotonic
- Generalize to a partial order for functions $f: \Sigma' \rightarrow \Sigma'$
 - $f \leq g$ if and only if $f(\sigma) \leq g(\sigma)$ for all σ
 - i.e. g "has more information" than f
 - if $f \leq g$, then $f(\sigma) \neq \perp$ means $f(\sigma) = g(\sigma)$
 - Ω is the bottom of this order

19

Properties of Function Chains

- Consider a chain of functions $\Sigma' \rightarrow \Sigma'$
 - $f_1 \leq f_2 \leq f_3 \leq \dots \leq f_n \leq \dots$
- The chain has an upper bound f s.t. $f_i \leq f$
 - Definition of f :
 - $f(\sigma) = \perp$ if $f_i(\sigma) = \perp$ for all i
 - $f(\sigma) = f_i(\sigma)$ if $f_i(\sigma) \neq \perp$ for some i
- This is a **least** upper bound
 - $f \leq g$ for any other upper bound g

20

Functional τ

$\tau : (\Sigma' \rightarrow \Sigma') \rightarrow (\Sigma' \rightarrow \Sigma')$

$\tau[f](\sigma) = \perp$ if $\sigma = \perp$
 σ if $\mathcal{B}[b](\sigma) = \text{false}$
 $f(\mathcal{C}[c](\sigma))$ if $\mathcal{B}[b](\sigma) = \text{true}$

This functional is monotonic
 $\Omega \leq \tau[\Omega]$ by the definition of Ω
 $\tau[\Omega] \leq \tau[\tau[\Omega]]$ by monotonicity
 $\tau^i[\Omega] \leq \tau^{i+1}[\Omega]$ for any i

21

Least Fixed Point for τ

- $\Omega \leq \tau[\Omega] \leq \dots \leq \tau^i[\Omega] \leq \tau^{i+1}[\Omega] \leq \dots$
- Consider the least upper bound of this chain (guaranteed to exist)
- It can be proven that this is the **unique least fixed point** for τ
 - $\tau[f] = f$
 - for any g such that $\tau[g] = g$, we have $f \leq g$

22

Back to While Loops

$\mathcal{E}[\text{while } b \text{ do } c](\sigma) =$
if not $\mathcal{B}[b](\sigma)$ then σ
else $\mathcal{E}[\text{while } b \text{ do } c](\mathcal{E}[c](\sigma))$

- $f = \mathcal{E}[\text{while } b \text{ do } c]$ is a solution of this equation iff f is a fixed point of τ
- The semantics of the while loop is the least fixed point of τ
 - least fixed point = the "most precise" solution: if the loop does not terminate for some initial state, the result is bottom

23

Infinite Loop

- Let w be "while true do skip". $\mathcal{E}[w] = ?$

$\tau[f](\sigma) = \perp$ if $\sigma = \perp$
 $f(\sigma)$ if $\mathcal{B}[\text{true}](\sigma) = \text{true}$

$\tau[f](\sigma) = f(\sigma)$ - that is, $\tau[f] = f$

- **Any** semantic function is a fixed point
- The **least** fixed point is Ω : therefore, $\mathcal{E}[w](\sigma) = \perp$ for all starting states σ
 - Without the "least" part: there may have been some σ for which $\mathcal{E}[w](\sigma) \neq \perp$

24

Another Example

while $x < 10$ do $x := x + 1$

$\tau[f](\sigma) = \perp$ if $\sigma = \perp$
 σ if $\sigma(x) \geq 10$
 $f(\sigma[x \leftarrow \sigma(x)+1])$ if $\sigma(x) < 10$

$\Omega(\sigma) = \perp$

$\tau[\Omega](\sigma) =$ bottom for bottom state, σ if
 $\sigma(x) \geq 10$, bottom if $\sigma(x) < 10$

$\tau[\tau[\Omega]](\sigma) =$ bottom for bottom state, σ if
 $\sigma(x) \geq 10$, $\sigma[x \leftarrow 10]$ if $\sigma(x) = 9$, bottom if $\sigma(x) < 9$

least fixed point = bottom for bottom state, σ if
 $\sigma(x) \geq 10$, $\sigma[x \leftarrow 10]$ if $\sigma(x) < 10$

25

Intuition

- The semantics is obtained as the limit of a series of approximations
 - Each approximation is a function, more precise than the previous one (i.e., with fewer bottom values)
- If f_5 is the fifth approximation, then the value of $f_5(\sigma)$ is
 - the proper final state if the loop terminates in four or fewer iterations
 - bottom, otherwise

26
