Denotational Semantics

- Winskel Ch. 5

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

Notation

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

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

Denotations for Expressions

\[
\mathcal{E}(n)(\sigma) = n \\
\mathcal{E}(X) = \sigma(X) \\
\mathcal{E}(\text{ae}_1 + \text{ae}_2)(\sigma) = \mathcal{E}(\text{ae}_1)(\sigma) + \mathcal{E}(\text{ae}_2)(\sigma) \\
\cdots
\]

\[
\mathcal{B}[\text{true}](\sigma) = \text{true} \\
\mathcal{B}[\text{false}](\sigma) = \text{false} \\
\mathcal{B}(\text{ae}_1 = \text{ae}_2)(\sigma) = (\mathcal{E}(\text{ae}_1)(\sigma) = \mathcal{E}(\text{ae}_2)(\sigma)) \\
\cdots
\]

Denotations for Statements

\[
\mathcal{C}[	ext{skip}](\sigma) = \sigma \\
\mathcal{C}[X := \text{ae}](\sigma) = \sigma[X \leftarrow \mathcal{E}[\text{ae}](\sigma)] \quad \text{i.e. } \sigma[\mathcal{E}[\text{ae}](\sigma)/X] \\
\mathcal{C}[c_1; c_2](\sigma) = \mathcal{C}[c_2](\mathcal{C}[c_1](\sigma)) \\
\mathcal{C}[\text{if } b \text{ then } c_1 \text{ else } c_2](\sigma) = \\
\quad \text{if } \mathcal{B}[b](\sigma) \text{ then } \mathcal{C}[c_1](\sigma) \text{ else } \mathcal{C}[c_2](\sigma) \\
\mathcal{C}[\text{while } b \text{ do } c](\sigma) = \\
\quad \text{if not } \mathcal{B}[b](\sigma) \text{ then } \sigma \\
\quad \text{else } \mathcal{C}[\text{while } b \text{ do } c](\mathcal{C}[c](\sigma))
\]
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)

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

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

Language Extension

- Nested blocks (but no procedures)
  \[
  \begin{aligned}
  &\text{begin } \text{ int } x; \ x := 0; \\
  &\quad \text{begin } \text{ int } x; \ x := 1; \ \text{end}; \\
  &\quad \text{write } x; \ \text{end}; \quad // \text{ should print 0}
  \end{aligned}
  \]
  At end, go back to the state before begin? But
  \[
  \begin{aligned}
  &\text{begin } \text{ int } x; \ \text{int } y; \ x := 0; \ y := 0; \\
  &\quad \text{begin } \text{ int } x; \ x := 1; \ y := 1; \ \text{end}; \\
  &\quad \text{write } x; \ \text{write } y; \ \text{end};
  \end{aligned}
  \]
  should print 01, not 00
Possible Solution: Split the State

- $\varepsilon$: (environment) maps each variable to its current location
- $\lambda$: (store) maps each location to a value

$$C[\text{skip}](\varepsilon, \lambda) = \lambda \quad C[x := 5](\varepsilon, \lambda) = \lambda[\varepsilon(x) \leftarrow 5]$$

$$D[\text{int } x](\varepsilon) = \varepsilon[x \leftarrow \text{new-location}(\varepsilon)]$$

$$C[\text{begin } ds; \text{cs}; \text{end}](\varepsilon, \lambda) = C[\text{cs}](D[\text{ds}](\varepsilon), \lambda)$$

$$C[c_1; c_2](\varepsilon, \lambda) = C[c_2](\varepsilon, C[c_1](\varepsilon, \lambda))$$

Since $c_1$ may be a block, the execution of $c_2$ uses the "upper-level" environment

Semantics of Loops

- Unfortunately, our earlier definition for loops is not quite right

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

- Let $w$ be "while true do skip"

$$C[w](\sigma) =$$
- if not $B[b](\sigma)$ then $\sigma$
- else $C[w](C[\text{skip}](\sigma))$

$$C[w](\sigma) = C[w](\sigma)$$ ??? any function satisfies this

An Even Bigger Problem

- The function for "while" is defined in terms of itself (recursive definition)

$$f(\sigma) = \{ \sigma \mid B[b](\sigma) = \text{false} \} \cup$$
- $\{ f(C[c](\sigma)) \mid B[b](\sigma) = \text{true} \}$

- This is not compositional ...

- Solution: use standard math machinery
Functionals and Fixed Points

- Functional: maps functions to functions
  \[ \tau[f](\sigma) = \{ \sigma \mid B[b](\sigma) = \text{false} \} \cup \{ f(C[c](\sigma)) \mid B[b](\sigma) = \text{true} \} \]
- \( \tau : (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma) \)
- Consider any function \( f \) such that \( \tau[f] = f \)
  - Standard math: \( f \) is a fixed point of \( \tau \)
- A fixed point of \( \tau \) can be thought of as representing the semantics of "while"

Partial Orders

- Partial order \( P \): a set on which we have defined a binary relation \( \leq \) (also \( \subseteq \)):
  - 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 \( q \)

More Definitions

- Bottom of a partial order: \( \bot \leq p \) 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 \( f \), \( x \leq z \)
Application to Denotational Semantics

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

More on the Bottom State

- \( B[be] \) and \( E[ae] \) 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

- In general, \( B[be] \) and \( E[ae] \) 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 \( B[be] \) or \( E[ae] \) return bottom, the semantic function for the surrounding statement should also return bottom
  - e.g. since \( E[5/0](\sigma) = \perp \), \( C[x := 5/0](\sigma) = \perp \)
Partial Orders

- \( \Sigma' \) is a partial order, defined as \( \bot \leq \sigma \)
- i.e. "bottom" is an approximation of any state
- For any \( f : \Sigma' \rightarrow \Sigma' \), since \( f(\bot) = \bot \), \( f \) 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 \( \sigma \)
- i.e. \( g \) "has more information" than \( f \)
- if \( f \leq g \), then \( f(\sigma) \neq \bot \) means \( f(\sigma) = g(\sigma) \)
- \( \Omega \) is the bottom of this order

Properties of Function Chains

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

Functional \( \tau \)

\[
\tau : (\Sigma' \rightarrow \Sigma') \rightarrow (\Sigma' \rightarrow \Sigma')
\]
\[
\tau[f](\sigma) = \begin{cases} 
\bot & \text{if } \sigma = \bot \\
\sigma & \text{if } B[b](\sigma) = \text{false} \\
f(C[c](\sigma)) & \text{if } B[b](\sigma) = \text{true}
\end{cases}
\]

This functional is monotonic
\( \Omega \leq \tau[\Omega] \) by the definition of \( \Omega \)
\( \tau[\Omega] \leq \tau[\tau[\Omega]] \) by monotonicity
\( \tau[i][\Omega] \leq \tau[i+1][\Omega] \) for any \( i \)
Least Fixed Point for $\tau$

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

Back to While Loops

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

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

Infinite Loop

- Let $w$ be "while true do skip". $\epsilon[w] =$
  - $\tau[f](\sigma) =$
    - $\bot$ if $\sigma = \bot$
    - $f(\sigma)$ if $B[true](\sigma) = true$
  - $\tau[f](\sigma) = f(\sigma)$ - that is, $\tau[f] = f$
- Any semantic function is a fixed point
  - The least fixed point is $\Omega$: therefore, $\epsilon[w](\sigma) = \bot$ for all starting states $\sigma$
  - Without the "least" part: there may have been some $\sigma$ for which $\epsilon[w](\sigma) \neq \bot$
Another Example

while x < 10 do x := x + 1

\[ f(\sigma) = \begin{cases} \bot & \text{if } \sigma = \bot \\ \sigma & \text{if } \sigma(x) \geq 10 \\ f(\sigma[x \leftarrow \sigma(x)+1]) & \text{if } \sigma(x) < 10 \end{cases} \]

\[ \Omega(\sigma) = \bot \]

\[ \{\Omega\}(\sigma) = \text{bottom for bottom state, } \sigma \text{ if } \sigma(x) \geq 10, \text{bottom if } \sigma(x) < 10 \]

\[ \{\{\Omega\}\}(\sigma) = \text{bottom for bottom state, } \sigma \text{ if } \sigma(x) \geq 10, [x \leftarrow 10] \text{ if } \sigma(x) = 9, \text{bottom if } \sigma(x) < 9 \]

**least fixed point** = bottom for bottom state, \( \sigma \) if \( \sigma(x) \geq 10 \), \([x \leftarrow 10]\) if \( \sigma(x) = 9 \), bottom if \( \sigma(x) < 9 \)

---

**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