Programming Language Semantics
Denotational Semantics

Chapter 5
Based on a lecture by
Martin Abadi
Introduction

• Denotational semantics is supposed to be mathematical:
  – The meaning of an expression is a mathematical object
  – A fair amount of mathematics is involved
• Denotational semantics is compositional
• Denotational semantics is more abstract and canonical than operational semantics
  – No small step vs. big step
• Denotational semantics is also called
  – Fixed point semantics
  – Mathematical semantics
  – Scott-Strachey semantics
Plan

• Definition of the denotational semantics of IMP (first attempt)
• Complete partial orders and related properties
  – Montonicity
  – Continuity
• Definition of denotational semantics of IMP
Denotational semantics

- **A**: \( A\text{exp} \rightarrow (\Sigma \rightarrow N) \)
- **B**: \( B\text{exp} \rightarrow (\Sigma \rightarrow T) \)
- **C**: \( \text{Com} \rightarrow (\Sigma \rightarrow \Sigma) \)
- Defined by structural induction
Denotational semantics of Aexp

• $A$: $A_{exp} \rightarrow (\Sigma \rightarrow \mathbb{N})$
• $A [n] = \{(\sigma, n) \mid \sigma \in \Sigma\}$
• $A [X] = \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\}$
• $A [a_0 + a_1] = \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in A[a_0], (\sigma, n_1) \in A[a_1]\}$
• $A [a_0 - a_1] = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in A[a_0], (\sigma, n_1) \in A[a_1]\}$
• $A [a_0 \times a_1] = \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in A[a_0], (\sigma, n_1) \in A[a_1]\}$

Lemma: $A [a]$ is a function
Denotational semantics of Aexp with λ

• $A : Aexp \rightarrow (\Sigma \rightarrow \mathbb{N})$
• $A \ [n] = \lambda \sigma \in \Sigma. n$
• $A \ [X] = \lambda \sigma \in \Sigma. \sigma(X)$
• $A \ [a_0 + a_1] = \lambda \sigma \in \Sigma. (A \ [a_0] \sigma + A \ [a_1] \sigma)$
• $A \ [a_0 - a_1] = \lambda \sigma \in \Sigma. (A \ [a_0] \sigma - A \ [a_1] \sigma)$
• $A \ [a_0 \times a_1] = \lambda \sigma \in \Sigma. (A \ [a_0] \sigma \times A \ [a_1] \sigma)$
Denotational semantics of Bexp

- **B**: $\text{Bexp} \rightarrow (\Sigma \rightarrow \text{T})$
- $\text{B} [\text{true}] = \{ (\sigma, \text{true}) \mid \sigma \in \Sigma \}$
- $\text{B} [\text{false}] = \{ (\sigma, \text{false}) \mid \sigma \in \Sigma \}$
- $\text{B} [a_0=a_1] = \{ (\sigma, \text{true}) \mid \sigma \in \Sigma \land \text{A}[a_0]\sigma = \text{A}[a_1]\sigma \} \cup \{ (\sigma, \text{false}) \mid \sigma \in \Sigma \land \text{A}[a_0]\sigma \neq \text{A}[a_1]\sigma \}$
- $\text{B} [a_0\leq a_1] = \{ (\sigma, \text{true}) \mid \sigma \in \Sigma \land \text{A}[a_0]\sigma \leq \text{A}[a_1]\sigma \} \cup \{ (\sigma, \text{false}) \mid \sigma \in \Sigma \land \text{A}[a_0]\sigma \neq \text{A}[a_1]\sigma \}$
- $\text{B} [\neg b] = \{ (\sigma, \neg_T t) \mid \sigma \in \Sigma, (\sigma, t) \in \text{B}[b] \}$
- $\text{B} [b_0 \land b_1] = \{ (\sigma, t_0 \land_T t_1) \mid \sigma \in \Sigma, (\sigma, t_0) \in \text{B}[b_0], (\sigma, t_1) \in \text{B}[b_1] \}$
- $\text{B} [b_0 \lor b_1] = \{ (\sigma, t_0 \lor_T t_1) \mid \sigma \in \Sigma, (\sigma, t_0) \in \text{B}[b_0], (\sigma, t_1) \in \text{B}[b_1] \}$

Lemma: $\text{B}[b]$ is a function
Denotational semantics of commands?

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

• So, we may try to define $C \llbracket c \rrbracket$ as a function that maps $\sigma$ to $\sigma'$:
  
  $- C \llbracket . \rrbracket : \text{Com} \rightarrow (\Sigma \rightarrow \Sigma)$
Denotational semantics of commands?

• Problem: running a command might not yield anything if the command does not terminate

• We introduce the special element \( \perp \) to denote a special outcome that stands for non-termination

• For any set \( X \), we write \( X_\perp \) for \( X \cup \{ \perp \} \)

• Convention:
  
  \[ \text{whenever } f \in X \to X_\perp \text{ we extend } f \text{ to } X_\perp \to X_\perp \]
  “strictly” so that \( f(\perp) = \perp \)
Denotational semantics of commands?

• We try:

  – \( C \llbracket \cdot \rrbracket : \text{Com} \rightarrow (\Sigma_{\bot} \rightarrow \Sigma_{\bot}) \)

• \( C \llbracket \text{skip} \rrbracket \sigma = \sigma \)

• \( C \llbracket c_0 ; c_1 \rrbracket \sigma = C \llbracket c_1 \rrbracket (C \llbracket c_0 \rrbracket \sigma) \)

• \( C \llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \rrbracket \sigma = \)
  
  \[ \text{if } B \llbracket b \rrbracket \sigma \text{ then } C \llbracket c_0 \rrbracket \sigma \text{ else } C \llbracket c_1 \rrbracket \sigma \]

• \( C \llbracket \text{while } b \text{ do } c \rrbracket \sigma = ? \)
Examples

- $C \left[ X := 2; \; X := 1 \right] \sigma = \sigma[1/X]

- $C \left[ \text{if true then } X := 2; \; X := 1 \text{ else } \ldots \right] \sigma = \sigma[1/X]

- The semantics does not care about intermediate states

- So far, we did not explicitly need $\bot$
Denotational semantics of commands?

• Abbreviation **$W=C \lbrack \text{while } b \text{ do } C \rbrack$**

• Idea: we rely on the equivalence
  
  while $b$ do $c$ $\sim$ if $b$ then $(c; \text{while } b \text{ do } c)$ else skip

• We may try using unwinding equation
  
  $W(\sigma) = \text{if } B[b] \sigma \text{ then } W(C[c] \sigma) \text{ else } \sigma$

• Unacceptable solution
  
  – Defines $W$ in terms of itself
  – It not evident that a suitable $W$ exists
  – It may not describe $W$ uniquely
    (e.g., for while true do skip)
Introduction to Domain Theory

• We will solve the unwinding equation through a general theory of recursive equations

• Think of programs as processors of streams of bits (streams of 0’s and 1’s, possibly terminated by $)$

What properties can we expect?
Motivation

• Let “isone” be a function that must return “1$” when the input string has at least a 1 and “0$” otherwise

• What should the result of “isone” be on the partial input “00.. 0”?
  – It must be the empty string ε
  – **Monotonicity**: Output is never retracted
    More information about the input is reflected in more information about the output

• How do we express monotonicity precisely?
Montonicity

• Define a partial order
  \( x \sqsubseteq y \)
  – A partial order is reflexive, transitive, and antisymmetric
  – \( y \) is a refinement of \( x \)

• For streams of bits \( x \sqsubseteq y \) when \( x \) is a prefix of \( y \)

• For programs, a typical order is:
  – No output (yet) \( \sqsubseteq \) some output

• Other orders
  – Subsets
Montonicity

• A set equipped with a partial order is a poset
• Definition:
  – D and E are postes
  – A function \( f: D \rightarrow E \) is monotonic if
    \[ \forall x, y \in D: x \leq_D y \implies f(x) \leq_E f(y) \]
  – The semantics of the program ought to be a monotonic function
    • More information about the input leads to more information about the output
Montonicity Example

• Consider our “isone” function with the prefix ordering

• Notation:
  – $0^k$ is the stream with $k$ consecutive 0’s
  – $0^\infty$ is the infinite stream with only 0’s

• Question (revisited): what is $\text{isone}(0^k)$?
  – By definition, $\text{isone}(0^k) = 0$ and $\text{isone}(0^k1) = 1$
  – But $0^k \subseteq 0^k$ and $0^k \not\subseteq 0^k1$
  – “isone” must be monotone, so:
    • $\text{isone}(0^k) \subseteq \text{isone}(0^k) = 0$
    • $\text{isone}(0^k) \subseteq \text{isone}(0^k1) = 1$
  – Therefore, monotonicity requires that $\text{isone}(0^k)$ is a common prefix of 0 and 1, namely $\varepsilon$
The semantics of every program must be monotonic
Motivation

- Are there other constraints on “isone”? 
- Define “isone” to satisfy the equations
  - isone(ε)=ε
  - isone(1s)=1$
  - isone(0s)=isone(s)
  - isone($)=0$
- What about 0$^\infty$?
- **Continuity**: finite output depends only on finite input (no infinite lookahead)
Upper and Lower Bounds

- An upper bound of a set if an element “bigger” than all elements in the set
- The least upper bound is the “smallest” among upper bounds:
  - $x_i \subseteq \cup<x_i>$ for all $i \in \mathbb{N}$
  - $\cup<x_i> \subseteq y$ for all upper bounds $y$ of $<x_i>$ and it is unique if it exists

- Greatests lower bounds are defined similarly
Chains

- A **chain** is a countable increasing sequence
  \[ \langle x_i \rangle = \{ x_i \in X \mid x_0 \sqsubseteq x_1 \sqsubseteq \ldots \} \]
- An upper bound of a set if an element “bigger” than all elements in the set
- The least upper bound is the “smallest” among upper bounds:
  - \( x_i \sqsubseteq \sqcup \langle x_i \rangle \) for all \( i \in \mathbb{N} \)
  - \( \sqcup \langle x_i \rangle \sqsubseteq y \) for all upper bounds \( y \) of \( \langle x_i \rangle \)
    and it is unique if it exists
Complete Partial Orders

- Not every poset has an upper bound
  - with \( \perp \subseteq n \) and \( n \sqsubseteq n \) for all \( n \in \mathbb{N} \)
  - \( \{1, 2\} \) does not have an upper bound
- Sometimes chains have no upper bound
  
  \[
  \begin{array}{c}
  0 \\
  1 \\
  2 \\
  \vdots \\
  0 \\
  \end{array}
  \]
  
  The chain \( 0 \leq 1 \leq 2 \leq \ldots \)
  does not have an upper bound
Complete Partial Orders

• It is convenient to work with posets where every chain (not necessarily every set) has a least upper bound

• A partial order $P$ is complete if every chain in $P$ has a least upper bound also in $P$

• We say that $P$ is a complete partial order (cpo)

• A cpo with a least ("bottom") element $\bot$ is a pointed cpo (pcpo)
Examples of cpo’s

• Any set $P$ with the order $x \sqsubseteq y$ if and only if $x = y$ is a cpo
  It is discrete or flat
• If we add $\bot$ so that $\bot \sqsubseteq x$ for all $x \in P$, we get a flat pointed cpo
• The set $\mathbb{N}$ with $\leq$ is a poset with a bottom, but not a complete one
• The set $\mathbb{N} \cup \{ \infty \}$ with $n \leq \infty$ is a pointed cpo
• The set $\mathbb{N}$ with $\geq$ is a cpo without bottom
• Let $S$ be a set and $\text{P}(S)$ denotes the set of all subsets of $S$ ordered by set inclusion
Constructing cpos

- If D and E are pointed cpos, then so is $D \times E$

$$(x, y) \leq_{D \times E} (x', y') \text{ iff } x \leq_D x' \text{ and } y \leq_E y'$$

$$\perp_{D \times E} = (\perp_D, \perp_E)$$

$$\sqcup (x_i, y_i) = (\sqcup_D x_i, \sqcup_E y_i)$$
Constructing cpos (2)

• If $S$ is a set of $E$ is a pcpos, then so is $S \rightarrow E$

$m \sqsubseteq m'$ iff $\forall s \in S: m(s) \sqsubseteq_E m'(s)$

$\bot_{S \rightarrow E} = \lambda s. \bot_E$

$\sqcup (m, m') = \lambda s. m(s) \sqcup_E m'(s)$
Continuity

• A monotonic function maps a chain of inputs into a chain of outputs:
  \( x_0 \subseteq x_1 \subseteq \ldots \Rightarrow f(x_0) \subseteq f(x_1) \subseteq \ldots \)

• It is always true that:
  \( \bigcup_i \langle f(x_i) \rangle \subseteq f(\bigcup_i \langle x_i \rangle) \)

• But
  \( f(\bigcup_i \langle x_i \rangle) \nsubseteq \bigcup_i \langle f(x_i) \rangle \)
  is not always true
A Discontinuity Example

\[ f(\bigcup_i \langle x_i \rangle) \neq \bigcup_i \langle f(x_i) \rangle \]
Continuity

- Each $f(x_i)$ uses a “finite” view of the input
- $f(\bigcup <x_i> )$ uses an “infinite” view of the input
- A function is **continuous** when $f(\bigcup <x_i> ) = \bigcup_i <f(x_i)>$
- The output generated using an infinite view of the input does not contain more information than all of the outputs based on finite inputs
Examples of Continuous Functions

• For the partial order \((\mathbb{N} \cup \{\infty\}, \leq)\)
  – The identity function is continuous
    \[\text{id}(\sqcap n_i) = \sqcup \text{id}(n_i)\]
  – The constant function “five(n)=5” is continuous
    \[\text{five}(\sqcap n_i) = \sqcup \text{five}(n_i)\]

• For a flat cpo \(A\), any monotonic function \(f: A \rightarrow A\)
  such that \(f\) is strict is continuous

• Chapter 8 of the textbook includes many more
  continuous functions
Fixed Points

• Solve the equation:

\[ W(\sigma) = \begin{cases} 
W(C[c] \sigma) & \text{if } B[b](\sigma) = \text{true} \\
\sigma & \text{if } B[b](\sigma) = \text{false} \\
\bot & \text{if } B[b](\sigma) = \bot 
\end{cases} \]

where \( W: \sum_\bot \rightarrow \sum_\bot \)

\( W = C[\text{while be do do c}] \)

• This equation can be written as \( W = F(W) \)

with:

\[ F(W) = \lambda \sigma. \begin{cases} 
W(C[c] \sigma) & \text{if } B[b](\sigma) = \text{true} \\
\sigma & \text{if } B[b](\sigma) = \text{false} \\
\bot & \text{if } B[b](\sigma) = \bot 
\end{cases} \]
Thus we are looking for a solution for $W = F(W)$
- a fixed point of $F$

Typically there are many fixed points

We may argue that $W$ ought to be continuous
$W \in [\Sigma \rightarrow \Sigma]$ 

Cut the number of solutions

We will see how to find the least fixed point for such an equation provided that $F$ itself is continuous
Fixed Point Theorem

- Define $F^k = \lambda x. F( F(\ldots F( x)\ldots))$ (F composed k times)
- If $D$ is a pointed cpo and $F : D \to D$ is continuous, then
  - for any fixed-point $x$ of $F$ and $k \in \mathbb{N}$
    $F^k (\bot) \sqsubseteq x$
  - The least of all fixed points is
    $\sqcup_k F^k (\bot)$

Proof:

i. By induction on $k$.
   - Base: $F^0 (\bot) = \bot \sqsubseteq x$
   - Induction step: $F^{k+1} (\bot) = F( F^k (\bot)) \sqsubseteq F( x) = x$

ii. It suffices to show that $\sqcup_k F^k (\bot)$ is a fixed-point
   - $F(\sqcup_k F^k (\bot)) = \sqcup_k F^{k+1} (\bot) = \sqcup_k F^k (\bot)$
Fixed-Points (notes)

- If $F$ is continuous on a pointed cpo, we know how to find the least fixed point.
- All other fixed points can be regarded as refinements of the least one.
- They contain more information, they are more precise.
- In general, they are also more arbitrary.
- They also make less sense for our purposes.
Denotational Semantics of IMP

• $\Sigma_\bot$ is a flat pointed cpo
  – A state has more information on non-termination
  – Otherwise, the states must be equal to be comparable (information-wise)

• We want strict functions $\Sigma_\bot \to \Sigma_\bot$ (therefore, continuous functions)

• The partial order on $\Sigma_\bot \to \Sigma_\bot$
  $f \sqsubseteq g$ iff $f(x) = \bot$ or $f(x) = g(x)$ for all $x \in \Sigma_\bot$
  – $g$ terminates with the same state whenever $f$ terminates
  – $g$ might terminate for more inputs
Denotational Semantics of IMP

• Recall that $W$ is a fixed point of $F:[[\Sigma \rightarrow \Sigma] \rightarrow [\Sigma \rightarrow \Sigma]]$

\[
F(w) = \lambda \sigma. \begin{cases} 
  w(C[c](\sigma)) & \text{if } B[b](\sigma) = \text{true} \\
  \sigma & \text{if } B[b](\sigma) = \text{false} \\
  \bot & \text{if } B[b](\sigma) = \bot
\end{cases}
\]

• $F$ is continuous

• Thus, we set

\[C[\text{while } b \text{ do } c] = \bigcup F^k(\bot)\]

• Least fixed point
  – Terminates least often of all fixed points

• Agrees on terminating states with all fixed point
Example(1)

- while true do skip
- $F:[[\Sigma_\bot \rightarrow \Sigma_\bot] \rightarrow [\Sigma_\bot \rightarrow \Sigma_\bot]]$

$\begin{align*}
F &= \lambda w. \lambda \sigma. \begin{cases} 
  w(C[c](\sigma)) & \text{if } B[b](\sigma) = \text{true} \\
  \sigma & \text{if } B[b](\sigma) = \text{false} \\
  \bot & \text{if } B[b](\sigma) = \bot
\end{cases} \\
B[true] &= \lambda \sigma. \text{true} \\
C[\text{skip}] &= \lambda \sigma. \sigma
\end{align*}$

$F = \lambda w. \lambda \sigma. w(\sigma)$
Example(1)

- while true do skip
- \( F: ([\sum \rightarrow \sum] \rightarrow [\sum \rightarrow \sum]) \)

\[
F = \lambda w. \lambda \sigma. w(\sigma)
\]

\( \text{Var} = \{x\} \)

\[
C[\text{while true do skip}] = \bigsqcup F^k(\bot) = \lambda \sigma. \bot
\]
Example(2)

• while false do c
• $F : [([\sum \to \sum] \to [\sum \to \sum])]$

\[
F = \lambda w.\lambda \sigma. \begin{cases} 
  w(C[c](\sigma)) & \text{if } B[b](\sigma) = \text{true} \\
  \sigma & \text{if } B[b](\sigma) = \text{false} \\
  \bot & \text{if } B[b](\sigma) = \bot
\end{cases}
\]

$B[\text{false}] = \lambda \sigma. \text{false}$

$F = \lambda w.\lambda \sigma. \sigma$
Example(2)

- while true do skip
- $F: \left[[\sum_\bot \rightarrow \sum_\bot] \rightarrow [\sum_\bot \rightarrow \sum_\bot]\right]$

\[
F = \lambda w.\lambda \sigma.\sigma
\]

Var = \{x\}

$C[\text{while false do } C] = \bigcup F^k(\bot) = \lambda \sigma.\sigma$
Example(3)

- while \(x \neq 3\) do \(x = x - 1\)
- \(F : [\sum \rightarrow \sum \rightarrow [\sum \rightarrow \sum]]\)

\[
F = \lambda w. \lambda \sigma. \begin{cases} 
  w(C[c](\sigma)) & \text{if } B[b](\sigma) = \text{true} \\
  \sigma & \text{if } B[b](\sigma) = \text{false} \\
  \bot & \text{if } B[b](\sigma) = \bot
\end{cases}
\]

\[
B[x!=3] = \lambda \sigma. \sigma(x) \neq 3
\]

\[
c[x=x-1] = \lambda \sigma. \sigma(\sigma(x)-1/x))
\]

\[
F = \lambda w. \lambda \sigma. \begin{cases} 
  w(\sigma(\sigma(x)-1/x))) & \text{if } \sigma(x) \neq 3 \\
  \sigma & \text{if } \sigma(x) = 3
\end{cases}
\]
Example(3)

- while $x \neq 3$ do $x = x - 1$

\[ F = \lambda w. \lambda \sigma. \begin{cases} B(\sigma(\sigma(x)-1/x)) & \text{if } \sigma(x) \neq 3 \\ \sigma & \text{if } \sigma(x) = 3 \end{cases} \]

<table>
<thead>
<tr>
<th>$F^0(\bot)$</th>
<th>$\lambda \sigma. \bot$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F^1(\bot)$</td>
<td>$\lambda \sigma. \text{if } \sigma(x) = 3 \text{ then } \sigma(3/x) \text{ else } \bot$</td>
</tr>
<tr>
<td>$F^2(\bot)$</td>
<td>$\lambda \sigma. \text{if } 3 \leq \sigma(x) \leq 4 \text{ then } \sigma(3/x) \text{ else } \bot$</td>
</tr>
<tr>
<td>$F^k(\bot)$</td>
<td>$\lambda \sigma. \text{if } 3 \leq \sigma(x) \leq 3+k-1 \text{ then } \sigma(3/x) \text{ else } \bot$</td>
</tr>
<tr>
<td>$\bigcup_k F^k(\bot)$</td>
<td>$\lambda \sigma. \text{if } 3 \leq \sigma(x) \text{ then } \sigma(3/x) \text{ else } \bot$</td>
</tr>
</tbody>
</table>
Example 4 Nested Loops

\[ Z = 0 ; \]

while \( X > 0 \) do (\n  \[ Y = X ; \]
    while \( (Y>0) \) do \n      \[ Z = Z + Y ; \]
      \[ Y = Y - 1; \) \]
    \[ X = X - 1 \]
  )
Equivalence of Semantics

• $\forall \sigma, \sigma' \in \Sigma:$
  $\sigma' = \text{C}[c] \sigma \iff \langle c, \sigma \rangle \rightarrow \sigma' \iff \langle c, \sigma \rangle \Rightarrow^* \sigma'$
Complete Partial Orders

• Let \((D, \sqsubseteq)\) be a partial order
  – \(D\) is a **complete lattice** if every subset has both greatest lower bounds and least upper bounds
Knaster-Tarski Theorem

- Let $f: L \rightarrow L$ be a monotonic function on a complete lattice $L$

- The least fixed point $\text{lfp}(f)$ exists

  $- \text{lfp}(f) = \bigcap\{x \in L: f(x) \sqsubseteq x\}$
Summary

• Denotational definitions are not necessarily better than operational semantics, and they usually require more mathematical work
• The mathematics may be done once and for all
• The mathematics may pay off:
• Some of its techniques are being transferred to operational semantics.
• It is trivial to prove that “If $B[b_1] = B[b_2]$ and $C[c_1] = C[c_2]$ then $C[\text{while } b_1 \text{ do } c_1] = C[\text{while } b_2 \text{ do } c_2]$” (compare with the operational semantics)
Summary

• Denotational semantics provides a way to declare the meaning of programs in an abstract way
  – Can handle side-effects
  – Loops
  – Recursion
  – Gotos
  – Non-determinism
  – But not low level concurrency

• Fixed point theory provides a declarative way to specify computations
  – Many usages