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:** Aexp $\rightarrow$ $(\Sigma\rightarrow N)$
- **B**: Bexp $\rightarrow(\Sigma\rightarrow T)$
- **C:** Com $\rightarrow(\Sigma\rightharpoonup\Sigma)$
- Defined by structural induction

Denotational semantics of Aexp

- **A:** Aexp $\to$ $(\Sigma \to N)$
- **A** $[\![n]\!] = \{(\sigma, n) \mid \sigma \in \Sigma\}$
- **A** $[\![X]\!] = \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\}$
- **A** $[\![a_0+a_1]\!] = \{(\sigma, n_0+n_1) \mid (\sigma, n_0) \in \mathbf{A}[\![a_0]\!], (\sigma, n_1) \in \mathbf{A}[\![a_1]\!]\}$
- **A** $[\![a_0-a_1]\!] = \{(\sigma, n_0-n_1) \mid (\sigma, n_0) \in \mathbf{A}[\![a_0]\!], (\sigma, n_1) \in \mathbf{A}[\![a_1]\!]\}$
- **A** $[\![a_0 \times a_1]\!] = \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \mathbf{A}[\![a_0]\!], (\sigma, n_1) \in \mathbf{A}[\![a_1]\!]\}$

Lemma: A $[\![a]\!]$ is a function

Denotational semantics of Aexp with λ

- $\mathbf{A}$**:** $\mathrm{Aexp} \rightarrow (\Sigma \rightarrow N)$
- $\mathbf{A}\,[\![n]\!] = \lambda\sigma\in\Sigma.n$
- $\mathbf{A}\,[\![X]\!] = \lambda\sigma\in\Sigma.\sigma(X)$
- $\mathbf{A}\,[\![a_0+a_1]\!] = \lambda\sigma\in\Sigma.(\mathbf{A}\,[\![a_0]\!]\sigma+\mathbf{A}[\![a_1]\!]\sigma)$
- $\mathbf{A}\,[\![a_0\text{-}a_1]\!] = \lambda\sigma\in\Sigma.(\mathbf{A}\,[\![a_0]\!]\sigma\text{-}\mathbf{A}[\![a_1]\!]\sigma)$
- $\mathbf{A}\,[\![a_0\times a_1]\!] = \lambda\sigma\in\Sigma.(\mathbf{A}\,[\![a_0]\!]\sigma \times \mathbf{A}[\![a_1]\!]\sigma)$

Denotational semantics of Bexp

- $\mathbf{B}$**:** $\text{Bexp} \to (\Sigma \to \mathrm{T})$
- $\mathbf{B}\,[\![\text{true}]\!] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma\}$
- $\mathbf{B}\,[\![\text{false}]\!] = \{(\sigma, \text{false}) \mid \sigma \in \Sigma\}$
- $\mathbf{B}\,[\![a_0 = a_1]\!] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma \;\&\; \mathbf{A}[\![a_0]\!]\sigma = \mathbf{A}[\![a_1]\!]\,\sigma\} \cup$
  $\{(\sigma, \text{false}) \mid \sigma \in \Sigma \;\&\; \mathbf{A}[\![a_0]\!]\sigma \neq \mathbf{A}[\![a_1]\!]\,\sigma\}$
- $\mathbf{B}\,[\![a_0 \leq a_1]\!] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma \;\&\; \mathbf{A}[\![a_0]\!]\sigma \leq \mathbf{A}[\![a_1]\!]\,\sigma\} \cup$
  $\{(\sigma, \text{false}) \mid \sigma \in \Sigma \;\&\; \mathbf{A}[\![a_0]\!]\sigma \not\leq \mathbf{A}[\![a_1]\!]\,\sigma\}$
- $\mathbf{B}\,[\![\neg b]\!] = \{(\sigma, \neg_{\mathrm{T}}\, t) \mid \sigma \in \Sigma, (\sigma, t) \in \mathbf{B}[\![b]\!]\}$
- $\mathbf{B}\,[\![b_0 \wedge b_1]\!] = \{(\sigma, t_0 \wedge_{\mathrm{T}} t_1) \mid \sigma \in \Sigma, (\sigma, t_0) \in \mathbf{B}[\![b_0]\!], (\sigma, t_1) \in \mathbf{B}[\![b_1]\!]\}$
- $\mathbf{B}\,[\![b_0 \vee b_1]\!] = \{(\sigma, t_0 \vee_{\mathrm{T}} t_1) \mid \sigma \in \Sigma, (\sigma, t_0) \in \mathbf{B}[\![b_0]\!], (\sigma, t_1) \in \mathbf{B}[\![b_1]\!]\}$

Lemma: $\mathrm{B}[\![b]\!]$ is a function

Denotational semantics of commands?

- Running a command c starting from a state σ yields another state σ'
- So, we may try to define **C** $[\![c]\!]$ as a function that maps σ to σ':
  - $\mathbf{C}\,[\![.]\!] : \mathrm{Com} \rightarrow (\Sigma \rightarrow \Sigma)$

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:
  - whenever $f \in X \to X_\perp$ we extend f to $X_\perp \to X_\perp$ "strictly" so that $f(\perp) = \perp$

Denotational semantics of commands?

- We try:
  - $C [\![.]\!] : Com \rightarrow (\Sigma_\perp \rightarrow \Sigma_\perp)$
- $C [\![skip]\!]\sigma = \sigma$
- $C [\![c_0 ; c_1]\!]\sigma = C [\![c_1]\!] (C [\![c_0]\!]\sigma)$
- $C [\![\text{if } b \text{ then } c0 \text{ else } c1]\!] \sigma =$
  $\text{if } B[\![b]\!] \sigma \text{ then } C [\![c_0]\!] \sigma \text{ else } C [\![c_1]\!] \sigma$
- $C [\![\text{while } b \text{ do } c]\!] \sigma = ?$

Examples

- $C[\![X:= 2;\ X:=1]\!]\sigma = \sigma[1/X]$
- $C[\![\text{if true then } X:=2;\ X:=1 \text{ else } \ldots]\!]\sigma = \sigma[1/X]$
- The semantics does not care about intermediate states
- So far, we did not explicitly need $\perp$

Denotational semantics of commands?

- Abbreviation $W = C[\![\text{while } b \text{ do } C]\!]$
- Idea: we rely on the equivalence

  while b do c ~ if b then (c; while b 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 $\forall x, y \in D: x \sqsubseteq_D y \Rightarrow f(x) \sqsubseteq_E f(y)$
  - 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^∞ is the infinite stream with only 0's
- Question (revisited): what is isone(0^k)?
  - By definition, isone(0^k\$) = 0\$ and isone(0^k1\$) = 1\$
  - But $0^k \sqsubseteq 0^k\$$ and $0^k \sqsubseteq 0^k 1\$$
  - "isone" must be monotone, so:
    - isone(0^k) $\sqsubseteq$ isone(0^k\$) = 0\$
    - isone(0^k) $\sqsubseteq$ isone(0^k1\$) = 1\$
  - Therefore, monotonicity requires that isone(0^k) is a common prefix of 0\$ and 1\$, namely ε

Thesis

- 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^∞?
- Continuity: finite output depends only on finite input (no infinite lookahead)

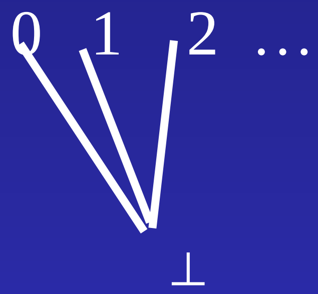
Upper and Lower Bounds

- 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:
  - $x_i \sqsubseteq \sqcup\langle x_i \rangle$ for all $i \in N$
  - $\sqcup\langle x_i \rangle \sqsubseteq y$ for all upper bounds y of $\langle x_i \rangle$
    and it is unique if it exists
- 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 $i \in N$
  - $\sqcup \langle x_i \rangle \sqsubseteq y$ 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 $\perp \sqsubseteq n$ and $n \sqsubseteq n$ for all $n \in N$
  - $\{1, 2\}$ does not have an upper bound
- Sometimes chains have no upper bound

⋮

2

1

0

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 ⊥ 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 $\perp$ so that $\perp \sqsubseteq$ x for all x $\in$ P, we get a flat pointed cpo
- The set N with $\leq$ is a poset with a bottom, but not a complete one
- The set N $\cup$ { ∞ } with n $\leq \infty$ is a pointed cpo
- The set N with$\geq$ is a cpo without bottom
- Let S be a set and 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
  $D \times E$
  $(x, y) \sqsubseteq_{D\times E} (x', y')$ iff $x \sqsubseteq_D x'$ and $y \sqsubseteq_E y'$
  $\perp_{D\times E} = (\perp_D, \perp_E)$
  $\sqcup (x_i, y_i) = (\sqcup_D x_i, \sqcup_E y_i)$

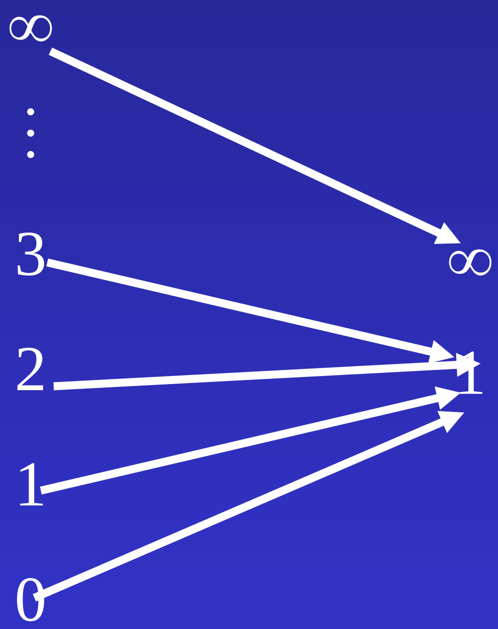
Constructing cpos (2)

- If S is a set of E is a pcpos, then so is
  $S \to E$
  $m \sqsubseteq m'$ iff $\forall s \in S$: $m(s) \sqsubseteq_E m'(s)$
  $\perp_{S \to E} = \lambda s.\ \perp_E$
  $\sqcup (m , m') = \lambda s.m(s) \sqcup_E m'(s)$

Continuity

- A monotonic function maps a chain of inputs into a chain of outputs:
  $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \Rightarrow f(x_0) \sqsubseteq f(x_1) \sqsubseteq \ldots$
- It is always true that:
  $\sqcup_i <f(x_i)> \sqsubseteq f(\sqcup_i <x_i>)$
- But
  $f(\sqcup_i <x_i>) \sqsubseteq \sqcup_i <f(x_i)>$
  is not always true

A Discontinuity Example



$f(\sqcup_i \langle x_i \rangle) \neq \sqcup i \langle f(x_i) \rangle$

Continuity

- Each $f(x_i)$ uses a “finite” view of the input
- $f(\sqcup\langle x_i\rangle)$ uses an “infinite” view of the input
- A function is continuous when
  $f(\sqcup\langle xi\rangle) = \sqcup_i \langle f(x_i)\rangle$
- 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 (N $\cup\{\infty\}, \leq$)
  - The identity function is continuous
    $id(\sqcup n_i) = \sqcup id(n_i)$
  - The constant function “five(n)=5” is continuous
    $five(\sqcup n_i) = \sqcup five(n_i)$
- For a flat cpo A, any monotonic function
  $f: A_\perp \rightarrow A_\perp$
  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 $W:\Sigma_\bot \rightarrow \Sigma_\bot$

  $W= C[\![\text{while be do c}]\!]$

- This equation can be written as $W = F(W)$ with:

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

Fixed Point (cont)

- 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 $W \in [\Sigma_\perp \to \Sigma_\perp]$
- 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 $k \in N$
    $F^k(\perp) \sqsubseteq x$
  - The least of all fixed points is
    $\sqcup_k F^k(\perp)$
- Proof:

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

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

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

- $\Sigma_\perp$ 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 $\Sigma_\perp \rightarrow \Sigma_\perp$ (therefore, continuous functions)
- The partial order on $\Sigma_\perp \rightarrow \Sigma_\perp$
  $f \sqsubseteq g$ iff $f(x) = \perp$ or $f(x) = g(x)$ for all $x \in \Sigma_\perp$
  - 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
  $F:[[\Sigma_\perp \to \Sigma_\perp] \to [\Sigma_\perp \to \Sigma_\perp]]$

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

- F is continuous
- Thus, we set

$$C[\![\text{while b do c}]\!] = \sqcup F^k(\perp)$$

- Least fixed point
  - Terminates least often of all fixed points
- Agrees on terminating states with all fixed point

Example(1)

- while true do skip
- $F:[[\Sigma_\perp \to \Sigma_\perp] \to [\Sigma_\perp \to \Sigma_\perp]]$

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

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

$C[\![\text{skip}]\!]=\lambda\sigma.\sigma$

$$F = \lambda\, w.\lambda\sigma.w(\sigma)$$

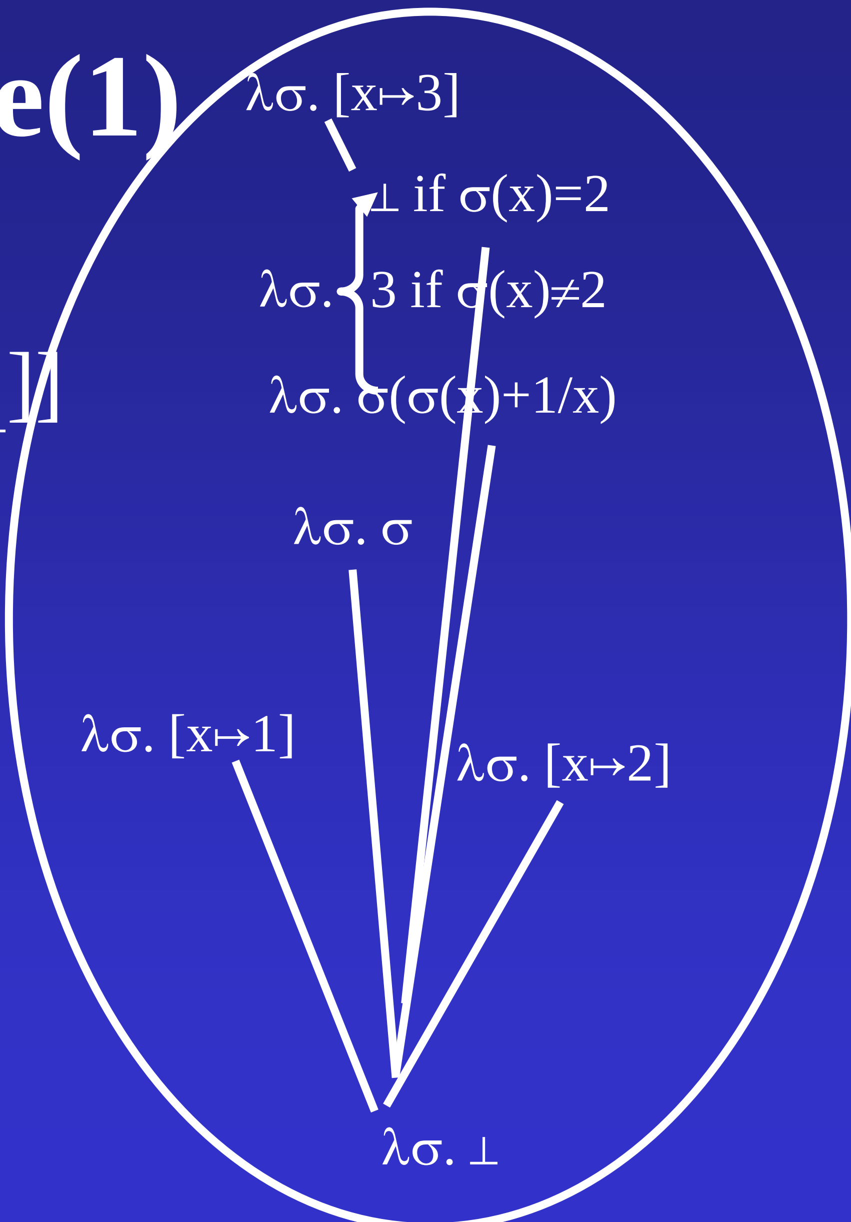
Example(1)

- while true do skip
- $F: [[\Sigma_\perp \to \Sigma_\perp] \to [\Sigma_\perp \to \Sigma_\perp]]$

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

$$Var = \{x\}$$

$$C[\![\text{while true do skip}]\!] = \sqcup F^k(\perp) = \lambda\sigma.\perp$$



Example(2)

- while false do c
- $F:[[\Sigma_\perp \to \Sigma_\perp] \to [\Sigma_\perp \to \Sigma_\perp]]$

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

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

$$F = \lambda w.\lambda\sigma.\sigma$$

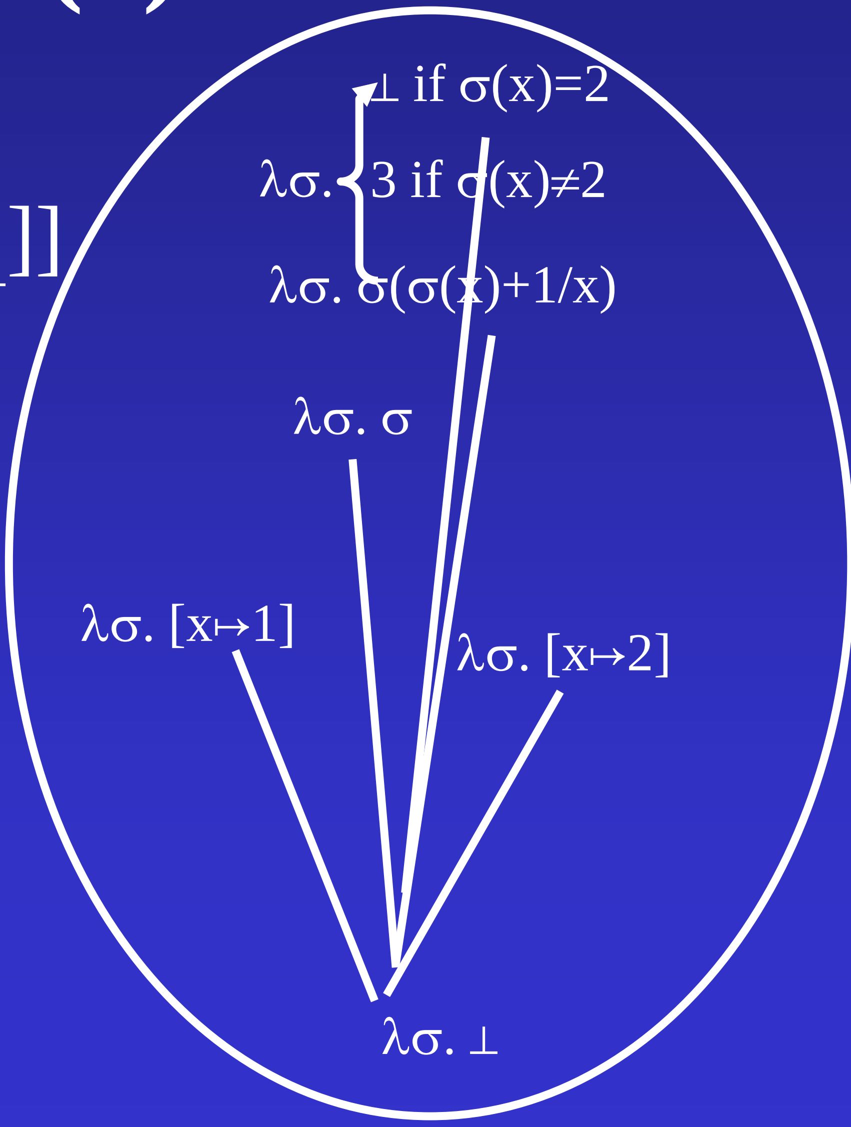
Example(2)

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

$F = \lambda w.\lambda\sigma.\sigma$

Var ={x}

$C[\![\text{while false do } C]\!] = \sqcup F^k(\perp) = \lambda\sigma.\sigma$



Example(3)

- while x≠3 do x = x -1
- F:[[$\sum_\perp \to \sum_\perp$]$\to$[$\sum_\perp \to \sum_\perp$]]

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

$B[\![x!=3]\!]=\lambda\sigma.\sigma(x)\neq 3$ $\quad$ $c[\![x=x-1]\!]=\lambda\sigma.\ \sigma(\sigma(x)-1/x))$

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

Example(3)

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

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

$F^0(\perp)$	$\lambda\sigma.\ \perp$
$F^1(\perp)$	$\lambda\sigma.$ if $\sigma(x)=3$ then $\sigma(3/x)$ else $\perp$
$F^2(\perp)$	$\lambda\sigma.$ if $3 \leq \sigma(x) \leq 4$ then $\sigma(3/x)$ else $\perp$
$F^k(\perp)$	$\lambda\sigma.$ if $3 \leq \sigma(x) \leq 3+k-1$ then $\sigma(3/x)$ else $\perp$
$\sqcup_k F^k(\perp)$	$\lambda\sigma.$ if $3 \leq \sigma(x)$ then $\sigma(3/x)$ else $\perp$

Example 4 Nested Loops

```
Z = 0 ;
while X > 0 do (
    Y = X;
     while (Y>0) do
            Z = Z + Y ;
            Y =  Y- 1; )
   X = X – 1
   )
```

Equivalence of Semantics

- $\forall \sigma, \sigma' \in \Sigma$:
  $\sigma' = C[\![c]\!]\sigma \Leftrightarrow \langle c, \sigma\rangle \rightarrow \sigma' \Leftrightarrow \langle c, \sigma\rangle \Rightarrow^* \sigma'$

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 lfp(f) exists
  - lfp(f) = $\sqcap\{x \in L: f(x) \sqsubseteq x\}$

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