

Introduction to Denotational Semantics

Lecture 4-6
CS263

CS 263

1

Plan

- Define the denotational semantics of IMP
 - First attempt
- Domains and their properties
 - Monotonicity
 - Continuity
- Finish the denotational semantics of IMP

CS 263

3

Denotational Semantics of Arithmetic Expressions

- We define inductively a function
 $A[\cdot] : Aexp \rightarrow (\Sigma \rightarrow \mathbb{Z})$

$A[n] \sigma =$ the integer denoted by literal n

$A[x] \sigma = \sigma(x)$

$A[e_1 + e_2] \sigma = A[e_1] \sigma + A[e_2] \sigma$

$A[e_1 - e_2] \sigma = A[e_1] \sigma - A[e_2] \sigma$

$A[e_1 * e_2] \sigma = A[e_1] \sigma * A[e_2] \sigma$

- This is a total function (defined for all expressions)

CS 263

5

Review

- The operational semantics is
 - simple
 - of many flavors (natural, small-step, more or less abstract)
 - not compositional
- Denotational semantics is
 - mathematical (the meaning of a syntactic expression is a mathematical object)
 - compositional
- Denotational semantics is also called: fixed-point semantics, mathematical semantics, Scott-Strachey semantics

CS 263

2

Rough Idea of Denotational Semantics

- The meaning of an arithmetic expression e in state σ is a number n
- So, we try to define $A[e]$ as a function that maps the current state to an integer:
 $A[\cdot] : Aexp \rightarrow (\Sigma \rightarrow \mathbb{Z})$
- The meaning of boolean expressions is defined in a similar way
 $B[\cdot] : Bexp \rightarrow (\Sigma \rightarrow \{\text{true}, \text{false}\})$
- All of these denotational functions are total
 - Defined for all syntactic elements
 - For other languages it might be convenient to define the semantics only for well-typed elements

CS 263

4

Denotational Semantics of Boolean Expressions

- We define inductively a function
 $B[\cdot] : Bexp \rightarrow (\Sigma \rightarrow \{\text{true}, \text{false}\})$

$B[\text{true}] \sigma = \text{true}$

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

$B[b_1 \wedge b_2] \sigma = B[b_1] \sigma \wedge B[b_2] \sigma$

$B[e_1 = e_2] \sigma = \text{if } A[e_1] \sigma = A[e_2] \sigma \text{ then true else false}$

CS 263

6

Denotational Semantics for Commands

- Running a command c starting from a state σ yields another state σ'
- So, we try to define $C[[c]]$ as a function that maps σ to σ'

$$C[[\cdot]] : \text{Comm} \rightarrow (\Sigma \rightarrow \Sigma)$$

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

CS 263

7

Denotational Semantics of Commands

- We introduce the special element \perp to denote a special resulting state that stands for non-termination
- For any set X , we write X_\perp to denote $X \cup \{\perp\}$

Convention:

whenever $f \in X \rightarrow X_\perp$ we extend f to $X_\perp \rightarrow X_\perp$ so that $f(\perp) = \perp$

- This is called strictness

CS 263

8

Denotational Semantics of Commands

- We try:

$$C[[\cdot]] : \text{Comm} \rightarrow (\Sigma \rightarrow \Sigma_\perp)$$

$C[[\text{skip}]] \sigma = \sigma$

$C[[x := e]] \sigma = \sigma[x := A[[e]] \sigma]$

$C[[c_1; c_2]] \sigma = C[[c_2]] (C[[c_1]] \sigma)$

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

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

CS 263

9

Examples

• $C[[x := 2; x := 1]] \sigma = \sigma[x := 1]$

• $C[[\text{if true then } x := 2; x := 1 \text{ else } \dots]] \sigma = \sigma[x := 1]$

- The semantics does not care of the intermediate states

- We didn't need \perp yet

CS 263

10

Denotational Semantics of WHILE

- Notation: $W = C[[\text{while } b \text{ do } c]]$
- Idea: rely on the equivalence
 $\text{while } b \text{ do } c = \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ else skip}$

- Try

$$W(\sigma) = \text{if } B[[b]]\sigma \text{ then } W(C[[c]]\sigma) \text{ else } \sigma$$

- This is the unwinding equation
- But it is not an acceptable definition of W because:
 - It defines W in terms of itself
 - It is not evident that such a W exists
 - It does not describe W uniquely
 - It is not compositional

CS 263

11

More on WHILE

- The unwinding equation does not specify W uniquely
- Take $C[[\text{while true do skip}]]$
- The unwinding equation reduces to $W(\sigma) = W(\sigma)$, which is satisfied by every function!

- Take $C[[\text{while } x \neq 0 \text{ do } x := x - 2]]$

- The following solution satisfies equation (for any σ')

$$W(\sigma) = \begin{cases} \sigma[x := 0] & \text{if } \sigma(x) = 2k \wedge \sigma(x) \geq 0 \\ \sigma' & \text{otherwise} \end{cases}$$

CS 263

12

Denotational Semantics of WHILE

- Define $W_k: \Sigma \rightarrow \Sigma_{\perp}$ (for $k \in \mathbb{N}$) such that

$$W_k(\sigma) = \begin{cases} \sigma & \text{if "while b do c" in state } \sigma \text{ terminates in} \\ & \text{fewer than } k \text{ iterations in state } \sigma \\ \perp & \text{otherwise} \end{cases}$$

- We can define the W_k functions as follows:

$$W_0(\sigma) = \perp$$

$$W_k(\sigma) = \begin{cases} W_{k-1}(C[c]\sigma) & \text{if } B[b]\sigma \\ \sigma & \text{otherwise} \end{cases} \quad \text{for } k \geq 1$$

CS 263

13

Denotational Semantics of WHILE

- How do we get W from W_k ?

$$W(\sigma) = \begin{cases} \sigma & \text{if } \exists k. W_k(\sigma) = \sigma \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

- This is a valid compositional definition of W
 - Depends only on $C[c]$ and $B[b]$

- Try the examples again:

- For $C[\text{while true do skip}]$

$$W_k(\sigma) = \perp \quad \text{for all } k, \text{ thus } W(\sigma) = \perp$$

- For $C[\text{while } x \neq 0 \text{ do } x := x - 2]$

$$W(\sigma) = \begin{cases} \sigma[x := 0] & \text{if } \sigma(x) = 2k \wedge \sigma(x) \geq 0 \\ \perp & \text{otherwise} \end{cases}$$

CS 263

14

More on WHILE

- The solution is not quite satisfactory because
 - It has an operational flavor
 - It does not generalize easily to more complicated semantics (e.g., higher-order functions)
- However, precisely due to the operational flavor this solution is easy to prove sound w.r.t operational semantics

CS 263

15

Introduction to Domain Theory

CS 263

16

A Simplified Setup

- Consider programs in an eager, deterministic language with one variable called "x"
 - All these restrictions are just to simplify the examples
- A state σ is just the value of x
 - Thus we can use \mathbb{Z} instead of Σ
- The semantics of a command gives the value of final x as a function of input x

$$C[c] : \mathbb{Z} \rightarrow \mathbb{Z}_{\perp}$$

CS 263

17

Examples. Revisited

- Take $C[\text{while true do skip}]$
 - Unwinding equation reduces to $W(x) = W(x)$
 - Any function satisfies the unwinding equation
 - Desired solution is $W(x) = \perp$
- Take $C[\text{while } x \neq 0 \text{ do } x := x - 2]$
 - Unwinding equation:

$$W(x) = \text{if } x \neq 0 \text{ then } W(x - 2) \text{ else } x$$
 - Solutions (for all values $n, m \in \mathbb{Z}_{\perp}$):

$$W(x) = \text{if } x \geq 0 \text{ then} \\ \quad \text{if } x \text{ even then } 0 \text{ else } n \\ \quad \text{else } m$$
 - Desired solution: $W(x) = \text{if } x \geq 0 \wedge x \text{ even then } 0 \text{ else } \perp$

CS 263

18

An Ordering of Solutions

- The desired solution is the one in which all the arbitrariness is replaced with non-termination
 - The arbitrary values in a solution are not uniquely determined by the semantics of the code
- We introduce an ordering of semantic functions
- Let $f, g \in \mathbb{Z} \rightarrow \mathbb{Z}_\perp$
- Define $f \sqsubseteq g$ as
 - $\forall x \in \mathbb{Z}. f(x) = \perp$ or $f(x) = g(x)$
 - A "larger" function is obtained by replacing some \perp in the "smaller" function with actual values

CS 263

19

Alternative Views of Function Ordering

- A semantic function $f \in \mathbb{Z} \rightarrow \mathbb{Z}_\perp$ can be written as $S_f \subseteq \mathbb{Z} \times \mathbb{Z}$ as follows:
 - $S_f = \{ (x, y) \mid x \in \mathbb{Z}, f(x) = y \neq \perp \}$
 - A list of the "terminating" values for the function
- If $f \sqsubseteq g$ then
 - $S_f \subseteq S_g$
 - We say that g **refines** f
 - We say that f **approximates** g
 - We say that g provides more information than f

CS 263

20

The "Best" Solution

- Consider again $C[\text{while } x \neq 0 \text{ do } x := x - 2]$
 - Unwinding equation:
 - $W(x) = \text{if } x \neq 0 \text{ then } W(x - 2) \text{ else } x$
- Not all solutions are comparable:
 - $W(x) = \text{if } x \geq 0 \text{ then if } x \text{ even then } 0 \text{ else } 1 \text{ else } 2$
 - $W(x) = \text{if } x \geq 0 \text{ then if } x \text{ even then } 0 \text{ else } \perp \text{ else } 3$
 - $W(x) = \text{if } x \geq 0 \text{ then if } x \text{ even then } 0 \text{ else } \perp \text{ else } \perp$ (least, best)
- Is there always a least solution?
- How do we find it?
- General framework for answering these questions

CS 263

21

Fixed-Point Equations

- Consider the general unwinding equation for **while**
 - $\text{while } b \text{ do } c \equiv \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ else skip}$
- We define a context C (command with a hole)
 - $C = \text{if } b \text{ then } c; \bullet \text{ else skip}$
 - $\text{while } b \text{ do } c \equiv C[\text{while } b \text{ do } c]$
 - C does not contain "while b do c"
- We can find such a context for any looping construct
 - Consider: $\text{fact } n = \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{fact } (n - 1)$
 - $C = \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n * \bullet (n - 1)$
 - $\text{fact} = C[\text{fact}]$

CS 263

22

Fixed-Point Equations

- The meaning of a context is a semantic functional
 - $F : (\mathbb{Z} \rightarrow \mathbb{Z}_\perp) \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z}_\perp)$ such that
 - $\llbracket C[w] \rrbracket = F \llbracket w \rrbracket$
- For "while": $C = \text{if } b \text{ then } c; \bullet \text{ else skip}$
 - $F w x = \text{if } \llbracket b \rrbracket x \text{ then } w(\llbracket c \rrbracket x) \text{ else } x$
 - F depends only on $\llbracket c \rrbracket$ and $\llbracket b \rrbracket$
- We can rewrite the unwinding equation for while
 - $W(x) = \text{if } \llbracket b \rrbracket x \text{ then } W(\llbracket c \rrbracket x) \text{ else } x$
 - or, $W x = F W x$ for all x ,
 - or, $W = F W$ (function equality)

CS 263

23

Fixed-Point Equations

- The meaning of "while" is a solution for $W = F W$
- Such a W is called a fixed point of F
- We want the **least fixed point**
 - We need a general way to find least fixed points
- Whether such a least fixed point exists depends on the properties of function F
 - Counterexample: $F w x = \text{if } w x = \perp \text{ then } 0 \text{ else } \perp$
 - Assume W is a fixed point
 - $F W x = W x = \text{if } W x = \perp \text{ then } 0 \text{ else } \perp$
 - Pick an x , then if $W x = \perp$ then $W x = 0$ else $W x = \perp$
 - Contradiction. This F has no fixed point!

CS 263

24

Monotonicity

- Good news: the functions F that *correspond to contexts in our language* have least fixed points!
- The only way $F f x$ uses f is by invoking it
- If any such invocation diverges, then $F f x$ diverges!

CS 263

25

Monotonicity (Cont.)

- Consider $f_0 \sqsubseteq f_1$. What can we say about the relationship between $F f_0 x$ and $F f_1 x$, for any x ?
- Assume $F f_0 x = n \neq \perp$. Show that $F f_1 x = n$
 - In computing $F f_0 x$, f_0 is invoked a finite number of times
 - All those invocations terminate with some values
 - The value of f_0 at other points does not matter!
 - But f_1 terminates with same results everywhere f_0 terminates
 - Thus $F f_1 x = n$ (determinism of F)
- If $F f_0 x = \perp$, it could be that $F f_1 x \neq \perp$
 - Take $F f x = f x$, $f_0(0) = \perp$ and $f_1(0) = 0$
- In general, if $f_0 \sqsubseteq f_1$ then $F f_0 \sqsubseteq F f_1$
- We say that F must be monotonic

CS 263

26

Monotonicity (Cont.)

- If we replace the sub-command with one that terminates more often, the host command will terminate more often
- The following F is not monotonic:
 $F w x = \text{if } w x = \perp \text{ then } 0 \text{ else } \perp$
 - This function does not correspond to a computable context
- The semantics of computable contexts are monotonic
 - Can be proved by induction on the structure of context

CS 263

27

Chains of Approximations

- Consider the command `while $x \neq 0$ then $x := x - 1$`
- Semantics: $W x = \text{if } 0 \leq x \text{ then } 0 \text{ else } \perp$
- Try the following approximations (for arbitrary k)
 $w_k x = \text{if } 0 \leq x < k \text{ then } 0 \text{ else } \perp$
 - w_k is the semantics if we allow at most k iterations
- Show that $w_k \sqsubseteq W$
 - All w_k approximate W
- Also, $w_k \sqsubseteq w_{k+1}$
 - We get more information if we allow more iterations
 - w_k form a chain of approximations of the true semantics
 - We say that W is an upper bound for the chain w_k

CS 263

28

Least Upper Bounds

- Recall: $W x = \text{if } 0 \leq x \text{ then } 0 \text{ else } \perp$
 $w_k x = \text{if } 0 \leq x < k \text{ then } 0 \text{ else } \perp$
- Pick any other upper bound for chain w_k
 - e.g. $U = \text{if } 0 \leq x \text{ then } 0 \text{ else } n$
- We see that $W \sqsubseteq U$
 - W is the least upper bound of the chain w_k
- Compute the least upper bound for a chain in $\mathbb{Z} \rightarrow \mathbb{Z}_\perp$:
 - for each x , we construct the sequence $f_1 x, f_2 x, \dots$
 - Thus: $(\sqcup_k f_k) x = \text{if } \exists k. f_k x = n \neq \perp \text{ then } n \text{ else } \perp$
 - We can verify that $W = \sqcup_k w_k$

CS 263

29

Solving Fixed Point Equations

- Thus $W = \sqcup w_k$
- Note that $w_0 = \lambda x. \perp$
- Note also that $w_{k+1} = F w_k$, where F is the meaning of context `if $x \neq 0$ then $x := x - 1$; • else skip`
- Thus, $W = \text{LFP}_F = \sqcup_k F^k (\lambda x. \perp)$ (*)
- Is this true for all functions F ?

CS 263

30

Continuity

- Consider F corresponding to a context in our language
- Consider a chain $g_0 \sqsubseteq \dots \sqsubseteq g_k$ with $\sqcup_k g_k = G$
 - Note that $F g_k$ form a chain also, because F is monotonic
- We'll show that, for any x , $F G x = (\sqcup_k (F g_k)) x$
 - We say that such functions F are continuous
- If $F G x = n \neq \perp$, then G was invoked a finite number of times, and terminated each time
 - For each such invocation, there is a j , such that g_j terminates with the same result
 - Let \max be the maximum such j , for all invocations
 - Thus, $F g_{\max} x = n$, and $(\sqcup_k (F g_k)) x = n$
- Similar reasoning for $F G x = \perp$

CS 263

31

The Fixed-Point Theorem

- If F is a semantic functional corresponding to a context in our language
 - F is monotonic and continuous
 - For any fixed-point G of F and $k \in \mathbb{N}$

$$F^k(\lambda x. \perp) \sqsubseteq G$$
 - The least of all fixed points is

$$\sqcup_k F^k(\lambda x. \perp)$$
- Proof:
 - by mathematical induction on k .
 - Base: $F^0(\lambda x. \perp) = \lambda x. \perp \sqsubseteq G$
 - Inductive: $F^{k+1}(\lambda x. \perp) = F(F^k(\lambda x. \perp)) \sqsubseteq F(G) = G$
 - Suffices to show that $\sqcup_k F^k(\lambda x. \perp)$ is a fixed-point

$$F(\sqcup_k F^k(\lambda x. \perp)) = \sqcup_k F^{k+1}(\lambda x. \perp) = \sqcup_k F^k(\lambda x. \perp)$$

CS 263

32

Denotational Semantics For WHILE

- We can use the fixed-point theorem to write the denotational semantics of while:

$$\llbracket \text{while } b \text{ do } c \rrbracket = \sqcup_k F^k(\lambda x. \perp)$$

where $F f x = \text{if } \llbracket b \rrbracket x \text{ then } f(\llbracket c \rrbracket x) \text{ else } x$
- Example: $\llbracket \text{while true do skip} \rrbracket = \lambda x. \perp$
- Example: $\llbracket \text{while } x \neq 0 \text{ then } x := x - 1 \rrbracket$
 - $F(\lambda x. \perp) x = \text{if } x = 0 \text{ then } x \text{ else } \perp$
 - $F^2(\lambda x. \perp) x = \text{if } x = 0 \text{ then } x \text{ else if } x - 1 = 0 \text{ then } x - 1 \text{ else } \perp$
 $= \text{if } 1 \geq x \geq 0 \text{ then } 0 \text{ else } \perp$
 - $F^3(\lambda x. \perp) x = \text{if } 2 \geq x \geq 0 \text{ then } 0 \text{ else } \perp$
 - $LFP_F = \text{if } x \geq 0 \text{ then } 0 \text{ else } \perp$
- In general, it is not easy to find the closed form for LFP!

CS 263

33

Discussion

- We can write the denotational semantics but we cannot always compute it.
 - Otherwise, we could decide the halting problem
 - H is halting for input 0 iff $\llbracket H \rrbracket 0 \neq \perp$
- We have derived this for programs with one variable
- We can generalize to multiple variables, even to variables ranging over richer data types, even higher-order functions
 - Domain theory

CS 263

34

Domain Theory

- A set D is a domain if
 - It has a partial order $x \sqsubseteq y$
 - Reflexive, transitive, and anti-symmetric
 - There is a least element \perp called bottom
 - Any chain $x_1 \sqsubseteq \dots \sqsubseteq x_n \sqsubseteq \dots$ has a least-upper bound $\sqcup_i x_i$
 - For all i , $x_i \sqsubseteq \sqcup_i x_i$ (is an upper bound)
 - For any y such that $(\forall i. x_i \sqsubseteq y)$, we have $\sqcup_i x_i \sqsubseteq y$ (least upper bound)
- Usual sets of semantic values are domains

CS 263

35

Example of Domains

- Example: $D = \mathbb{Z} \rightarrow \mathbb{Z}_\perp$
 - $f \sqsubseteq g$ iff for all $n \in \mathbb{Z}$, $f n = \perp$ or $f n = g n$
 - $\perp_b = \lambda n. \perp$
 - For a chain f_i , the LUB = $\lambda n. \text{if } \exists k. f_k x = m \neq \perp \text{ then } m \text{ else } \perp$
- Example: Take a set A and a special element \perp , then $A_\perp = A \cup \{\perp\}$ is a flat domain:
 - $a \sqsubseteq b$ iff $a = \perp$ or $a = b$
 - For a chain a_i , LUB = $\text{if } \exists k. a_k \neq \perp \text{ then } a_k \text{ else } \perp$
- Exercise: If D_1 and D_2 are domains, then $D_1 \rightarrow D_2$ is a domain, and so is $D_1 \times D_2$

CS 263

36

Monotonicity and Continuity

- A function $f : D_1 \rightarrow D_2$ is monotonic iff for all $x, y \in D_1$: $x \sqsubseteq y \Rightarrow f x \sqsubseteq f y$
- A function $F : D_1 \rightarrow D_2$ is continuous iff for all chains x_i in D_1 : $F(\sqcup_i x_i) = \sqcup_i (F x_i)$
- We can show that functions corresponding to usual language constructs are monotonic and continuous
 - Show that $F f x = f(f_0 x)$ is monotonic and continuous, for any f_0 that is monotonic and continuous

CS 263

37

Least Fixed-Point Theorem

- If D is a domain, and $F : D \rightarrow D$ is a continuous function then
 - $\perp, F \perp, F(F \perp), \dots$ form a chain in D
 - $\sqcup_i (F^i \perp)$ is the least fixed point of F

CS 263

38

Equivalence of Denotational and Operational Semantics

CS 263

39

Equivalence with Operational Semantics

- Statement:
 - $\langle e, \sigma \rangle \Downarrow n$ iff $A[e] \sigma = n$
 - $\langle b, \sigma \rangle \Downarrow t$ iff $B[b] \sigma = t$
 - $\langle c, \sigma \rangle \Downarrow \sigma'$ iff $C[c] \sigma = \sigma'$
- Each of these proofs has two directions
- The case of arithmetic and boolean expressions are easy by structural induction
- The case for commands is more interesting

CS 263

40

Equivalence Proof (I)

- \Rightarrow If we have a derivation $D :: \langle c, \sigma \rangle \Downarrow \sigma'$ then $C[c] \sigma = \sigma'$
- The proof is by induction on the structure of D
 - Notation:
 - while b do c = w
 - $C[w] = W$
 - To prove that $W(\sigma) = \sigma'$
 - We consider only the cases when at the root of D we have either while-true or while-false
 - The other cases are easier

CS 263

41

Equivalence Proof (II)

- Case: the rule at root is while-false
$$D :: \frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma}$$
 - σ' must be σ
 - From D_1 and using the equivalence for booleans we have that $B[b] \sigma = \text{false}$
 - From the unwinding equation we get that $W(\sigma) = \sigma$

CS 263

42

Equivalence Proof (III)

- Case: the rule at the root of D is while-true
- $$D :: \frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{true} \quad D_2 :: \langle c, \sigma \rangle \Downarrow \sigma_1 \quad D_3 :: \langle \text{while } b \text{ do } c, \sigma_1 \rangle \Downarrow \sigma'}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma'}$$
- From D_1 we get that $B[b]\sigma = \text{true}$
 - By IH on D_2 we get that $C[c]\sigma = \sigma_1$
 - By IH on D_3 we get that $W(\sigma_1) = \sigma'$
 - From the unwinding equation we get

$$W(\sigma) = W(C[c]\sigma) = W(\sigma_1) = \sigma'$$
 - Note that any solution to the unwinding equation has the same property! Why?

CS 263

43

Equivalence Proof (IV)

- \Leftarrow If $C[c]\sigma = \sigma'$ and $\sigma' \neq \perp$ then there exists $D : \langle c, \sigma \rangle \Downarrow \sigma'$
- Proof by induction on the structure of the command c
- We do only the case for WHILE
- We know that $\sqcup F^k(\lambda x. \perp)(\sigma) = \sigma' \neq \perp$
- From the definition of \sqcup on the flat domain Σ_\perp

$$\exists k. F^k(\lambda x. \perp)\sigma = \sigma' \neq \perp$$
- Sufficient to prove

$$\forall k \in \mathbb{N}. (F^k(\lambda x. \perp)\sigma = \sigma' \neq \perp \Rightarrow D : \langle c, \sigma \rangle \Downarrow \sigma')$$
- This can be proved by mathematical induction on k
 - Note that this is nested induction!

CS 263

44

Equivalence Proof (V)

- Base: $k = 0$. Vacuously true.
- Inductive step: $k \geq 1$
 - If $B[b]\sigma = \text{false}$.
 - Check that $F^k(\lambda x. \perp)\sigma = \sigma$, thus $\sigma = \sigma'$
 - We know that $D_1 :: \langle b, \sigma \rangle \Downarrow \text{false}$ exists
 - We construct D as follows:

$$D :: \frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma}$$

CS 263

45

Equivalence Proof (VI)

- Still in inductive case, $k \geq 1$ and $B[b]\sigma = \text{true}$
- Thus $D_1 :: \langle b, \sigma \rangle \Downarrow \text{true}$ exists
- Check that $\sigma' = F^k(\lambda x. \perp)\sigma = F^{k-1}(\lambda x. \perp)(C[c]\sigma)$
- Since $\sigma' \neq \perp$ we have $\sigma_1 = C[c]\sigma \neq \perp$
- By IH (struct. induction) on c exists $D_2 :: \langle c, \sigma \rangle \Downarrow \sigma_1$
- By IH (math. induction) exists $D_3 :: \langle w, \sigma_1 \rangle \Downarrow \sigma'$
- We can build the desired D:

$$D :: \frac{D_1 :: \langle b, \sigma \rangle \Downarrow \text{true} \quad D_2 :: \langle c, \sigma \rangle \Downarrow \sigma_1 \quad D_3 :: \langle \text{while } b \text{ do } c, \sigma_1 \rangle \Downarrow \sigma'}{\langle \text{while } b \text{ do } c, \sigma \rangle \Downarrow \sigma'}$$

CS 263

46

Comments on Denotational Semantics

- 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
 - e.g. The proof that the set of continuous functions is a domain
- The mathematics may pay off
 - It is trivial to prove that

$$\text{"If } B[b_1] = B[b_2] \text{ and } C[c_1] = C[c_2] \text{ then } C[\text{while } b_1 \text{ do } c_1] = C[\text{while } b_2 \text{ do } c_2]\text{"}$$
 (compare with the operational semantics)

CS 263

47

Denotation Semantics Supplemental Material

CS 263

48

The Function Domain

- We are interested only in those semantic functions that are monotonic and continuous
 - Notation: $[D \rightarrow E]$ the set of continuous functions from D to E .
 - Theorem: If D and E are domains, then $[D \rightarrow E]$ is a domain
- Proof:
 - Define the (induced) partial order on $[D \rightarrow E]$

$$f \sqsubseteq_{[D \rightarrow E]} g \text{ iff } \forall x \in D. f(x) \sqsubseteq_E g(x)$$
 - This is the pointwise ordering
 - Define the bottom of $[D \rightarrow E]$

$$\perp_{[D \rightarrow E]} =_{\text{def}} \lambda x \in D. \perp_E$$
 - Define least upper bounds

$$\sqcup_{[D \rightarrow E]} \langle f_i \rangle =_{\text{def}} \lambda x \in D. \sqcup_E \langle f_i(x) \rangle$$

CS 263

49

The Function-Space Domain

- Prove completeness of $\sqsubseteq_{[D \rightarrow E]}$
 - lubs exist for all chains. Easy
 - lubs are continuous, hence in $[D \rightarrow E]$
 - Let $\langle f_i \rangle$ be a chain with lub F :

$$F = \sqcup_i \langle f_i \rangle = \lambda x. \sqcup_i \langle f_i(x) \rangle$$
 - Pick $\langle x_j \rangle$ a chain in D
 - To show: $F(\sqcup_j \langle x_j \rangle) = \sqcup_j F(x_j)$

CS 263

50

The Function-Space Domain (Cont.)

- To show: $F(\sqcup_j \langle x_j \rangle) = \sqcup F(x_j)$
- But

$$F(\sqcup_j \langle x_j \rangle) = (\sqcup_i f_i)(\sqcup_j \langle x_j \rangle)$$

$$= \sqcup_i (\sqcup_j f_i(x_j))$$
- and

$$\sqcup_j F(x_j) = \sqcup_j (\sqcup_i f_i(x_j))$$
- Is it the case that $\sqcup_i (\sqcup_j f_i(x_j)) = \sqcup_j (\sqcup_i f_i(x_j))$?
 - It happens to be so in this case, but we must prove it
 - This only holds because f_i are continuous!

CS 263

51

Proof Techniques for Domains

- We must prove $\sqcup_i (\sqcup_j f_i(x_j)) = \sqcup_j (\sqcup_i f_i(x_j))$

 - How do we prove that $x = y$ for some $x, y \in E$?
 - One method: prove $x \sqsubseteq y$ and $y \sqsubseteq x$
 - Then by anti-symmetry of \sqsubseteq we get the equality
 - How do we prove $\sqcup \langle x_i \rangle \sqsubseteq y$?
 - One method: prove $\forall i \in \mathbb{N}. x_i \sqsubseteq y$
 - Then use the fact that \sqcup is the least upper bound
 - How do we prove $x \sqsubseteq \sqcup \langle y_i \rangle$?
 - One method: prove $\exists i \in \mathbb{N}. x \sqsubseteq y_i$
 - Then use the fact that \sqcup is an upper bound

CS 263

52

Proof Techniques for Domains (Example)

- We must prove $\sqcup_i (\sqcup_j f_i(x_j)) \sqsubseteq \sqcup_n (\sqcup_m f_m(x_n))$
 - We could try either proof trick #2 or #3
 - Trick #2 is generally more powerful
 - Trick #2 works here
- To show (for an arbitrary i) $\sqcup_j f_i(x_j) \sqsubseteq \sqcup_n (\sqcup_m f_m(x_n))$
 - Trick #2 again
- To show (for arbitrary i and j) $f_i(x_j) \sqsubseteq \sqcup_n (\sqcup_m f_m(x_n))$
 - Trick #3 twice
- To show $\forall i \forall j. \exists m \exists n. f_i(x_j) \sqsubseteq f_m(x_n)$
 - Easy: pick $m = i$ and $n = j$
- The other direction works in a similar manner

CS 263

53

More Domains

- So, $[D \rightarrow E]$ is a domain if D and E are domains
- $D \times E$ is a domain if D and E are domains

$$(x, y) \sqsubseteq_{D \times E} (x', y') \text{ iff } x \sqsubseteq_D x' \text{ and } y \sqsubseteq_E y'$$

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

$$\sqcup (x_i, y_i) =_{\text{def}} (\sqcup_D x_i, \sqcup_E y_i)$$
 - Convince yourself that these definitions are well-formed
- A set $D_\perp = D \cup \{\perp\}$ with the ordering

$$x \sqsubseteq y \text{ iff } x = y \text{ or } x = \perp$$
 is a domain
 - How do chains look in such a domain?
 - What is \sqcup ?
 - Such a domain is called a flat domain

CS 263

54

Some Continuous Functions

- Function application: $\text{app} =_{\text{def}} \lambda f \in [D \rightarrow E]. \lambda x \in D. f(x)$
- Function composition:
 $\text{comp} =_{\text{def}} \lambda f \in [E \rightarrow F]. \lambda g \in [D \rightarrow E]. \lambda x \in D. f(g(x))$
- Pairing: $\text{mkPair} =_{\text{def}} \lambda x \in D. \lambda y \in E. (x, y)$
- Projection: $\text{proj} =_{\text{def}} \lambda (x, y) \in D \times E. x$
- Case analysis:
 $\text{case} =_{\text{def}} \lambda b \in \text{bool}. \lambda t \in D. \lambda f \in D. \text{if } b \text{ then } t \text{ else } f$
- Proofs of these in Winskel, Chapter 8