Introduction to Denotational Semantics

CS263

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

Plan

- Define the denotational semantics of IMP
  - First attempt, runs into difficulties with "while"
  - Second attempt, introduce a restricted form of "while" in the language; then generalize to real "while"
- Later (after we see lambda calculus)
  - A more general form of denotational semantics
  - Introduction to domain theory
  - Denotational semantics of lambda calculus

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:

  $$A[\![\cdot]\!] : \text{Aexp} \to (\Sigma \to \mathbb{Z})$$

  - The $(\Sigma \to \mathbb{Z})$ is a <u>partial</u> function (uninitialized variables)
- The meaning of boolean expressions is defined in a similar way

  $$B[\![\cdot]\!] : \text{Bexp} \to (\Sigma \to \{\text{true, false}\})$$

Denotational Semantics of Arithmetic Expressions

- We define inductively a function

$$A[\![\cdot]\!] : Aexp \to (\Sigma \to \mathbb{Z})$$

$A[\![n]\!]\ \sigma$ = the integer denoted by literal n
$A[\![x]\!]\ \sigma = \sigma(x)$ if $x \in Dom(\sigma)$
$A[\![e_1+e_2]\!]\ \sigma = A[\![e_1]\!]\sigma + A[\![e_2]\!]\sigma$
$A[\![e_1*e_2]\!]\sigma = A[\![e_1]\!]\sigma \times A[\![e_2]\!]\sigma$

Denotational Semantics of Boolean Expressions

- We define inductively a function

$$B[\![\cdot]\!] : Bexp \to (\Sigma \to \{true, false\})$$

$B[\![true]\!]\sigma = true$
$B[\![false]\!]\sigma = false$
$B[\![b_1 \wedge b_2]\!]\sigma = B[\![b_1]\!]\ \sigma \wedge B[\![b_2]\!]\ \sigma$
$B[\![e_1 = e_2]\!]\sigma$ = if $A[\![e_1]\!]\ \sigma = A[\![e_2]\!]\ \sigma$ then true else false

Denotational Semantics for Commands

- Running a command c starting from a state σ yields another state σ'
- We try to define $C[\![c]\!]$ as a function that maps σ to σ'

$$C[\![\cdot]\!] : Comm \to (\Sigma \to \Sigma)$$

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

Denotational Semantics of Commands

- We introduce the special element $\perp$ (called bottom) to denote non-termination
- For any set X, we write $X_\perp$ to denote $X \cup \{\perp\}$

Denotational Semantics of Commands

- We try: $C[\![\cdot]\!] : Comm \to (\Sigma \to \Sigma_\perp)$

$C[\![skip]\!]\ \sigma = \sigma$
$C[\![x := e]\!]\ \sigma = \sigma[x := A[\![e]\!]\ \sigma]$
$C[\![c_1; c_2]\!]\ \sigma = C[\![c_2]\!]\ (C[\![c_1]\!]\ \sigma)$
Convention:
whenever $f \in X \to Y_\perp$ we extend f to $X_\perp \to Y_\perp$ so that $f(\perp) = \perp$
- This is called <u>strictness</u>
- Thus: $C[\![c]\!]\ \perp = \perp$ for any command

$C[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!]\ \sigma = \text{if } B[\![b]\!]\sigma \text{ then } C[\![c_1]\!]\sigma \text{ else } C[\![c_2]\!]\sigma$
$C[\![\text{while } b \text{ do } c]\!]\ \sigma = ?$

Examples

- $C[\![x := 2; x := 1]\!]\ \sigma = \sigma[x := 1]$
- $C[\![\text{if true then } x := 2; x := 1 \text{ else } \ldots]\!]\ \sigma = \sigma[x := 1]$
- The semantics does not care of the intermediate states
- We didn't need $\perp$ yet

Denotational Semantics of WHILE

- Notation: $W = C[\![\text{while } b \text{ do } c]\!]$
- One idea: rely on the equivalence (as in op. sem.)
  while b do c = if b then c; while b do c else skip
- This gives:
  $W(\sigma) = \text{if } B[\![b]\!]\sigma \text{ then } W(C[\![c]\!]\sigma) \text{ else } \sigma$
- This is the <u>unwinding equation</u>
- But it is not an acceptable <u>definition</u> of W because:
  - It defines W in terms of itself
    - It is not <u>compositional</u> (defined based on semantics of sub-expressions)
  - It is not evident that such a W exists
  - It may not describe W uniquely

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 W !
- Take $C[\![\text{while } x \neq 0 \text{ do } x := x - 2]\!]$
  - The following solution satisfies the equation

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

(for any σ')

New Attempt for WHILE

- Idea: introduce an approximation of "while" that has a finite unrolling
- Introduce two new language constructs to IMP:

  c ::= … | while_k b do c | forever
- while_k b do c   (with k a natural number *constant*)
  - A bounded "while"
  - Execute at most k – 1 iterations of the loop body; loop forever if more iterations would be needed

  "while_0 b do c" behaves like "forever"

  "while_{k+1} b do c" behaves like "if b then c; while_k b do c else skip"
- Original "while" is like while_∞

Denotational Semantics of WHILE

- Let W_k be shorthand for $C[\![\text{while}_k \text{ b do c}]\!]$
- We can define the $W_k : \Sigma \to \Sigma_\perp$ (for $k \in \mathbb{N}$) :
  - $W_0(\sigma) = \perp$
  - For k > 0, $W_k(\sigma)$ = if $B[\![b]\!]\sigma$ then $W_{k-1}(C[\![c]\!]\sigma)$ else σ
    - With the usual constraint that $W_k(C[\![c]\!]\sigma) = \perp$ if $C[\![c]\!]\sigma = \perp$

- Another view:

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

- Intuitively, we are looking for the "limit" W_∞

Denotational Semantics of WHILE

- How do we get W from W_k?

$$W(\sigma) = \begin{cases} \perp & \text{if } \forall k.\ W_k(\sigma) = \perp \\ W_k(\sigma) & \text{if k smallest such that } W_k(\sigma) \neq \perp \end{cases}$$

- This is a valid compositional definition of W
  - Depends only on $C[\![c]\!]$ and $B[\![b]\!]$
- Try the examples again:
  - For $C[\![\text{while true do skip}]\!]$

    $W_k(\sigma) = \perp$ for all k, thus $W(\sigma) = \perp$
  - For $C[\![\text{while } x \neq 0 \text{ do } x := x - 2]\!]$

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

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)

- The domain theory builds the mathematical tools necessary to generalize this result
  - We will learn some domain theory when we do denotational semantics for lambda calculus

- However, precisely due to the operational flavor this solution is easy to prove sound w.r.t operational semantics

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 $C[\![c]\!]\sigma = \sigma'$ and $\sigma' \neq \bot$
- Each of these proofs has two directions
- The case of arithmetic and boolean expressions are easy by structural induction on expressions
- The case for commands is more interesting

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
  - $C[\![w]\!] = W$
- Must 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

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 $B[\![b]\!]\sigma$ = false
  - This means that $W_1(\sigma) = \sigma$
  - Therefore $W(\sigma) = \sigma$

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 $B[\![b]\!]\sigma$ = true
- By IH on D_2 we get that $C[\![c]\!]\sigma = \sigma_1 \neq \bot$
- By IH on D_3 we get that $W(\sigma_1) = \sigma' \neq \bot$
  - There is k smallest such that $W_k(\sigma_1) = \sigma'$
- But for any j, $W_{j+1}(\sigma) = W_j(C[\![c]\!]\sigma) = W_j(\sigma_1)$
- Then k + 1 is smallest such that $W_{k+1}(\sigma) = \sigma'$
  - K + 1 is smallest because $W_k(\sigma) = W_{k-1}(\sigma_1) = \bot$
- Thus $W(\sigma) = \sigma'$ (q.e.d.)

Equivalence Proof (IV)

$\Leftarrow$ If $C[\![c]\!]\sigma = \sigma'$ and $\sigma' \neq \bot$
then there exists $D : \langle c, \sigma\rangle \Downarrow \sigma'$

- Proof by <u>induction on the structure of the command c</u>
- We do only the case for WHILE
- We know that exists smallest k s.t. $W_k(\sigma) = \sigma'$
- Sufficient to prove
  $\forall k\in\mathbb{N}.\ \forall\sigma.$ (if k smallest s.t. $W_k(\sigma) = \sigma' \neq \bot$
  then there exists $D : \langle c, \sigma\rangle \Downarrow \sigma'$)
- This can be proved by <u>mathematical induction on k</u>
  - Note that this is nested induction !

Equivalence Proof (V)

- Base: k = 0. Vacuously true.
- Inductive case, subcase k = 1.
  - Pick σ, $W_1(\sigma) = \sigma' \neq \bot$
  - Therefore $B[\![b]\!]\sigma$ = false and $\sigma = \sigma'$
  - Thus there is $D_1 :: \langle b, \sigma\rangle \Downarrow$ false
  - We construct D as follows:

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

Equivalence Proof (VI)

- Still in inductive case (k > 1)
- Pick arbitrary σ s.t. k smallest with $W_k(\sigma) = \sigma' \neq \bot$
  - Since $W_1(\sigma) = \bot$ we have $B[\![b]\!]\sigma$ = true
  - Thus $D_1 :: \langle b, \sigma\rangle \Downarrow$ true exists
- Since $\sigma' \neq \bot$ we have $\sigma_1 = C[\![c]\!]\sigma \neq \bot$
  - By IH (struct. induction) on c there is $D_2 :: \langle c, \sigma\rangle \Downarrow \sigma_1$
- We have for all j : $W_j(\sigma) = W_{j-1}(\sigma_1)$
- Then k – 1 is smallest s.t. $W_{k-1}(\sigma_1) \neq \bot$
  - By IH (math. induction) there exists $D_3 :: \langle w, \sigma_1\rangle \Downarrow \sigma'$

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

Comments on Denotational Semantics

- Denotational definitions are not necessarily better than operational semantics, and they usually require more mathematical work
- The mathematics may pay off
  - 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)
- We skipped the domain theory here
  - We'll revisit some concepts later for lambda calculus

Introduction to Domain Theory

Supplement

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

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

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 $\in \mathbb{Z}_\perp$):

    $W(x) = \text{if } x \geq 0 \text{ then}$

    $\quad\quad \text{if } x \text{ even then } 0 \text{ else } n$

    $\quad \text{else } m$

    Is this correct?
  - Desired solution: $W(x) = \text{if } x \geq 0 \wedge \text{even}(x) \text{ then } 0 \text{ else } \perp$

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 $f, g \in \mathbb{Z} \to \mathbb{Z}_\perp$
- Define $f \sqsubseteq g$ as

  $\forall x \in \mathbb{Z}.\ f(x) = \perp \text{ or } f(x) = g(x)$
  - A "larger" function is obtained by replacing some $\perp$ in the "smaller" function with actual values
  - Idea: semantic functions "grow" if you grow your time horizon for running the program

Alternative Views of Function Ordering

- A semantic function $f \in \mathbb{Z} \to \mathbb{Z}_\perp$ can be written as $S_f \subseteq \mathbb{Z} \times \mathbb{Z}$ as follows:

  S_f = { (x, y) | $x \in \mathbb{Z}$, f(x) = y $\neq \perp$ }
  - A list of the "terminating" input-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

The "Best" Solution

- Consider again C⟦while x ≠ 0 do x := x – 2⟧
  - Unwinding equation:

    W(x) = if x ≠ 0 then W(x – 2) else x
- Not all solutions are comparable:

  W(x) = if x ≥ 0 then if x even then 0 else 1 else 2

  W(x) = if x ≥ 0 then if x even then 0 else ⊥ else 3

  W(x) = if x ≥ 0 then if x even then 0 else ⊥ else ⊥ (least, best)

- Is there always a least solution ?
- How do we find it ?
- General framework for answering these questions

Fixed-Point Equations

- Consider the general unwinding equation for while

  while b do c ≡ if b then c; while b do c else skip

- We define a context C (command with a hole)

  C = if b then c; • else skip

  while b do c ≡ C[while b do c]
  - C does not contain "while b do c"

- We can find such a context for any looping construct
  - Consider: fact n = if n = 0 then 1 else n * fact (n – 1)
  - C = λn. if n = 0 then 1 else n * • (n – 1)
  - fact = C [fact]

Fixed-Point Equations

- The meaning of a context is a semantic functional $F : (\mathbb{Z} \to \mathbb{Z}_\perp) \to (\mathbb{Z} \to \mathbb{Z}_\perp)$ such that

  ⟦C[w]⟧ = F ⟦w⟧

- For "while": C = if b then c; • else skip

  F w x = if ⟦b⟧ x then w (⟦c⟧ x) else x
  - F depends only on ⟦c⟧ and ⟦b⟧

- We can rewrite the unwinding equation for while
  - W(x) = if ⟦b⟧ x then W(⟦c⟧ x) else x
  - or, W x = (F W) x for all x,
  - or, W = F W (function equality)

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 (most non-termination of all possible solutions)
  - 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 = if w x = ⊥ then 0 else ⊥
  - Assume W is a fixed point
  - F W x = W x = if W x = ⊥ then 0 else ⊥
  - Pick an x, then if W x = ⊥ then W x = 0 else W x = ⊥
  - Contradiction. This F has no fixed point !

Monotonicity

- Good news: the functions F that *correspond to contexts in all reasonable languages* 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 !

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 ≠ ⊥. 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 (F is a function)
- If F f_0 x = ⊥ , it could be that F f_1 x ≠ ⊥
  - Take F f x = f x, f_0 (0) = ⊥ 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

Monotonicity (Cont.)

- If we replace the sub-command with one that terminates more often, the whole command will terminate more often
- The following F is not monotonic:
  F w x = if w x = ⊥ then 0 else ⊥
  - 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

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)

  $w_k\ x = \text{if } 0 \leq x \leq k \text{ then } 0 \text{ else } \perp$
  - 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 <u>chain of approximations</u> of the true semantics
  - We say that W is an <u>upper bound</u> for the chain w_k

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, $U = \text{if } 0 \leq x \text{ then } 0 \text{ else } 5$
- We see that $W \sqsubseteq U$
  - W is the <u>least upper bound</u> of the chain w_k (written $\sqcup_k w_k$)
- Compute the least upper bound for a chain in $\mathbb{Z} \to \mathbb{Z}_\perp$:
  - for each x, we construct the sequence $f_1\ x$, $f_2\ x$, …
  - 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 $W = \sqcup_k w_k$

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 $x := x - 1; \bullet$ else skip
- Thus, $W = LFP_F = \sqcup_k F^k (\lambda x.\ \perp)$
- Is this true for all functions F ?

Continuity

- Consider F **corresponding to a context** in our language
- Consider a chain $g_0 \sqsubseteq \ldots \sqsubseteq g_k$ 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 <u>continuous</u>
- 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 $F\ G\ x = \perp$

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 $k \in \mathbb{N}$
    $F^k(\lambda x.\bot) \sqsubseteq G$
  - The least of all fixed points is
    $\sqcup_k F^k(\lambda x.\bot)$
- Proof:
  1. by mathematical induction on k.
     Base: $F^0(\lambda x.\bot) = \lambda x.\bot \sqsubseteq G$
     Inductive: $F^{k+1}(\lambda x.\bot) = F(F^k(\lambda x.\bot)) \sqsubseteq F(G) = G$
  2. Suffices to show that $\sqcup_k F^k(\lambda x.\bot)$ is a fixed-point
     $F(\sqcup_k F^k(\lambda x.\bot)) = \sqcup_k F^{k+1}(\lambda x.\bot) = \sqcup_k F^k(\lambda x.\bot)$

Denotational Semantics For WHILE

- We can use the fixed-point theorem to write the denotational semantics of while:
  $[\![\text{while } b \text{ do } c]\!] = \sqcup_k F^k(\lambda x.\bot)$
  where $F\ f\ x = \text{if } [\![b]\!]\ x \text{ then } f\ ([\![c]\!]\ x) \text{ else } x$
- Example: $[\![\text{while true do skip}]\!] = \lambda x.\bot$
- Example: $[\![\text{while } x \neq 0 \text{ then } x := x - 1]\!]$
  - $F(\lambda x.\bot)\ x = \text{if } x = 0 \text{ then } 0 \text{ else } \bot$
  - $F^2(\lambda x.\bot)\ x = \text{if } x = 0 \text{ then } 0 \text{ else if } x - 1 = 0 \text{ then } 0 \text{ else } \bot$
    $= \text{if } 1 \geq x \geq 0 \text{ then } 0 \text{ else } \bot$
  - $F^3(\lambda x.\bot)\ x = \text{if } 2 \geq x \geq 0 \text{ then } 0 \text{ else } \bot$
  - $LFP_F = \text{if } x \geq 0 \text{ then } 0 \text{ else } \bot$
- In general, it is not easy to find the closed form for LFP !

Discussion

- We can express the denotational semantics but we cannot always compute it.
  - Otherwise, we could decide the halting problem
  - H is halting for input 0 iff $[\![H]\!]\ 0 \neq \bot$
- 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

Domain Theory

- A set D is a <u>domain</u> if
  - It has a partial order $x \sqsubseteq y$
    - Reflexive, transitive, and anti-symmetric
  - There is a least element $\bot$ called bottom
  - Any chain $x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$ 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

Example of Domains

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

Monotonicity and Continuity

- A function $f : D_1 \to D_2$ is monotonic iff for all $x, y \in D$: $x \sqsubseteq y \Rightarrow f\ x \sqsubseteq f\ y$
- A function $F : D_1 \to D_2$ 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

Least Fixed-Point Theorem

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

Denotation Semantics
Supplemental Material

The Function Domain

- We are interested only in those semantic functions that are monotonic and continuous
  - Notation: $[D \to E]$ the set of continuous functions from D to E.
  - Theorem: If D and E are domains, then $[D \to E]$ is a domain
- Proof:
  - Define the (induced) partial order on $[D \to E]$
    $f \sqsubseteq_{[D \to E]} g \quad \text{iff} \quad \forall x \in D.\ f(x) \sqsubseteq_E g(x)$
    - This is the <u>pointwise ordering</u>
  - Define the bottom of $[D \to E]$
    $\perp_{[D \to E]} \quad =_{def} \quad \lambda x \in D. \perp_E$
  - Define least upper bounds
    $\sqcup_{[D \to E]} \langle f_i \rangle =_{def} \lambda x \in D. \sqcup_E \langle f_i(x) \rangle$

The Function-Space Domain

- Prove completeness of $\sqsubseteq_{[D \to E]}$
1. lubs exist for all chains. Easy
2. lubs are continuous, hence in $[D \to E]$
   - 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: $F(\sqcup_j \langle x_j \rangle) = \sqcup_j F(x_j)$

The Function-Space Domain (Cont.)

- To show: $F(\sqcup_j x_j) = \sqcup F(x_j)$
- But
  $F(\sqcup x_j) = (\sqcup_i f_i)(\sqcup_i x_j)$
  $= \sqcup_i (\sqcup_j f_i(x_j))$
- 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 !

Proof Techniques for Domains

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

1. 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
2. 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 <u>least</u> upper bound
3. 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

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 poweful
  - 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

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

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

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

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

  $x \sqsubseteq y$ iff $x = y$ 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

Some Continuous Functions

- Function application: app $=_{def} \lambda f\in[D\rightarrow E].\ \lambda x\in D.f\ (x)$
- Function composition:

  comp $=_{def} \lambda f\in[E\rightarrow F].\lambda g\in[D\rightarrow E].\lambda x\in D.f(g(x))$
- Pairing: mkPair $=_{def} \lambda x\in D.\lambda y\in E.(x, y)$
- Projection: proj $=_{def} \lambda(x, y) \in D \times E.\ x$
- Case analysis:

  case $=_{def} \lambda b\in bool_\perp.\lambda t\in D.\lambda f\in D.$if b then t else f
- Proofs of these in Winskel, Chapter 8