Announcements

- Homework 2 is due today
- Thanks for the feedback and keep it coming!
  - MW class, reading posts after class

Questions?

Review of Denotational Semantics

Denotational Semantics of Arithmetic Expressions

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

- \[ A[n] \sigma = \text{the integer denoted by literal } n \]
- \[ A[x] \sigma = \sigma(x) \]
- \[ A[e_1+e_2] \sigma = A[e_1]\sigma + A[e_2]\sigma \]
- \[ A[e_1-e_2] \sigma = A[e_1]\sigma - A[e_2]\sigma \]
- \[ A[e_1\cdot e_2] \sigma = A[e_1]\sigma \cdot A[e_2]\sigma \]

- This is a total function (= defined for all expressions)
Denotational Semantics of Boolean Expressions

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

- \( B[\text{true}] \sigma = \text{true} \)
- \( B[\text{false}] \sigma = \text{false} \)
- \( B[b_1 \land b_2] \sigma = B[b_1] \sigma \land B[b_2] \sigma \)
- \( B[e_1 = e_2] \sigma = \text{if } A[e_1] \sigma = A[e_2] \sigma \text{ then true else false} \)

Seems Easy So Far

\[ [\text{carrot}] = \text{carrot} \]
\[ [\text{bowling pin}] = \text{bowling pin} \]

Denotational Semantics of Commands

- We try:
  \( C[] : \text{Com} \rightarrow (\Sigma \rightarrow \Sigma_\perp) \)

- \( C[\text{skip}] \sigma = \sigma \)
- \( C[x := e] \sigma = \sigma[x := A[e] \sigma] \)
- \( C[c_1; c_2] \sigma = C[c_2] (C[c_1] \sigma) \)
- \( C[\text{if } b \text{ then } c_1 \text{ else } c_2] \sigma = \text{if } B[b] \sigma \text{ then } C[c_1] \sigma \text{ else } C[c_2][\sigma] \)

Denotational Semantics of Commands

- Can’t use the same tricks as in operational semantics (directly)

Start from the Unwinding Equation

- What’s the unwinding equation?
  \( W(\sigma) = C[\text{while } b \text{ do } c] \sigma \)
  \( W(\sigma) = \text{if } B[b] \sigma \text{ then } W(C[c] \sigma) \text{ else } \sigma \)

DS of While: Unwinding

- Notation: \( W = C[\text{while } b \text{ do } c] \)
- Idea: rely on the equivalence (justification?)
  \( \text{while } b \text{ do } c = \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ else skip} \)
- Try
  \( W(\sigma) = \text{if } B[b] \sigma \text{ then } W(C[c] \sigma) \text{ else } \sigma \)

- This is called the unwinding equation
- It is not a good denotation 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
### Preview: while k-steps Semantics

- Define $W_k : \Sigma \rightarrow \Sigma_\perp$ (for $k \in \mathbb{N}$) such that
  
  \[
  W_k(\sigma) = \begin{cases} 
  \sigma' & \text{if "while } b \text{ do } c \text{" in state } \sigma \text{ terminates in fewer than } k \text{ iterations in state } \sigma' \\
  \perp & \text{otherwise}
  \end{cases}
  \]

- We can define the $W_k$ functions as follows:
  
  \[
  W_0(\sigma) = \perp
  \]
  
  \[
  W_k(\sigma) = \begin{cases} 
  W_{k-1}(\text{C}[c];\sigma) & \text{if } B[b];\sigma \text{ for } k \geq 1 \\
  \sigma & \text{otherwise}
  \end{cases}
  \]

### while Semantics

- 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 **compositional definition** of $W$
  - Depends only on $C[c]$ and $B[b]$

### It Works, But ...

- This solution is _not quite satisfactory_ because
  - It has an _operational flavor_ (= "run the loop")
  - It _does not generalize_ easily to more complicated semantics (e.g., higher-order functions)

### End of Review

**On to Domain Theory**

### Recall: Denotational Game Plan

- Since while is _recursive_
  - always have something like: $W(\sigma) = F(W(\sigma))$
- Admits _many possible values_ for $W(\sigma)$
- We will _order_ them
  - With respect to non-termination = "least"
- And then find the _least fixed point_

\[
\text{LFP } W(\sigma) = F(W(\sigma)) \Rightarrow \text{meaning of "while"}
\]

### Simple Domain Theory

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

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

- Take C[while true do skip]
  - Unwinding equation reduces to $W(x) = W(x)$
  - Any function satisfies the unwinding equation
  - Desired solution is $W(x) = \bot$

- Take C[while $x \neq 0$ do $x := x - 2$]
  - Unwinding equation:
    $W(x) = \begin{cases} W(x - 2) & \text{if } x \neq 0 \\ x & \text{if } x \text{ even} \land x \geq 0 \\ \bot & \text{else} \end{cases}$
  - Solutions (for all values $n, m \in \mathbb{Z}$):
    $W(x) = \begin{cases} \bot & \text{if } x \geq 0 \\ n & \text{if } x \text{ even} \\ m & \text{else} \end{cases}$
  - Desired solution: $W(x) = \begin{cases} 0 & \text{if } x \text{ even} \\ \bot & \text{else} \end{cases}$

What is the Desired Solution?

- Returns or $\bot$ value if loop terminates
- Eliminate arbitrary return values when non-terminating

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 $f, g : \mathbb{Z} \rightarrow \mathbb{Z}$
  - Define $f \sqsubseteq g$ as
    $\forall x \in \mathbb{Z}, f(x) = \bot \lor f(x) = g(x)$
  - A “smaller” function terminates at most as often, and when it terminates it produces the same result

Alternative Views of Function Ordering

- A semantic function $f : \mathbb{Z} \rightarrow \mathbb{Z}$ can be written as $S_f \subseteq \mathbb{Z} \times \mathbb{Z}$ as follows:
  $S_f = \{(x, y) : x \in \mathbb{Z}, f(x) = y \neq \bot\}$
  - set of “terminating” values for the function
- If $f \sqsubseteq g$ then
  - $S_f \subseteq S_g$ (refinement)
  - 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 $x \neq 0$ do $x := x - 2$]
  - Unwinding equation:
    $W(x) = \begin{cases} W(x - 2) & \text{if } x \neq 0 \\ x & \text{if } x \text{ even} \land x \geq 0 \\ \bot & \text{else} \end{cases}$
- Not all solutions are comparable:
  $W(x) = \begin{cases} \bot & \text{if } x \geq 0 \land x \text{ even} \\ 1 & \text{else 1} \\ 2 & \text{else 2} \end{cases}$
  $W(x) = \begin{cases} \bot & \text{if } x \geq 0 \land x \text{ even} \\ \bot & \text{else \bot} \end{cases}$
  $W(x) = \begin{cases} \bot & \text{if } x \geq 0 \land x \text{ even} \\ \bot & \text{else \bot} \end{cases}$
  (last one is least and best)

Any questions that come to mind?

- Fixed points?
- How do get $f, g$?
Any questions that come to mind?

- Is there always a least solution?
- How do we find it?
- If only we had a general framework for answering these questions …

A Recursive Labyrinth

...Okay, we get it. You win.

Fixed-Point Equations

- Consider the general unwinding equation for while
  
  \[
  \text{while } b \text{ do } c = \text{if } b \text{ then } c \text{; while } b \text{ do } c \text{ else skip}
  \]
- We define a context \( K \) (command with a hole)
  
  \[
  K = \text{if } b \text{ then } c \text{; else skip}
  \]
- The grammar for \( K \) does not contain "while b do c"

Fixed-Point Equations

- We can find such a (recursive) context for any looping construct
  
  \[
  \text{Consider: } \text{fact } n = \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{ fact } (n - 1)
  \]
  
  \[
  K(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n * (n - 1)
  \]
  
  \[
  \text{fact } = \text{K[fact ]}
  \]

Fixed-Point Equations

- We can find such a (recursive) context for any looping construct
  
  \[
  \text{Consider: } \text{fact } n = \text{if } n = 0 \text{ then } 1 \text{ else } n * \text{ fact } (n - 1)
  \]
  
  \[
  K(n) = \text{if } n = 0 \text{ then } 1 \text{ else } n * (n - 1)
  \]
  
  \[
  \text{fact } = \text{K[fact ]}
  \]

Fixed-Point Equations

- The meaning of a context is a semantic functional
  \[
  F : (\mathbb{Z} \to \mathbb{Z}) \to (\mathbb{Z} \to \mathbb{Z}) \text{ such that }
  \]
  
  \[
  F[K[w]] = F[w]
  \]

- For "while": \( K = \text{if } b \text{ then } c \text{; else skip} \)
  \[
  F[w] x = \text{if } b[x] \text{ then } w[(c) x] \text{ else } x
  \]
  
  \[
  - F \text{ depends only on } [c] \text{ and } [b].
  - We can rewrite the unwinding equation for while
    \[
    W(x) = \text{if } b[x] \text{ then } W(c) x \text{ else } x
    \]
    
    \[
    - \text{or, } W x = F W x \text{ for all } x,
    \]
    
    \[
    - \text{or, } W = F W \text{ (by function equality)}
    \]
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
  - 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: $F w x = \text{if } w x = \perp \text{ then 0 else } \perp$
- Assume $W$ is a fixed point
  - Pick an $x$, then if $W x = \perp$ then $W x = 0$ else $W x = \perp$
- Contradiction. This $F$ has no fixed point!

Can We Solve This?

- Good news: the functions $F$ that correspond to contexts in our language have least fixed points!
- The only way $F w x$ uses $w$ is by invoking it
- If any such invocation diverges, then $F w x$ diverges!
- It turns out: $F$ is monotonic, continuous
  - Not shown here!

New Notation: $\lambda$

- $\lambda x. e$
  - an anonymous function with body $e$ and argument $x$
- Example: $\text{double}(x) = x + x$
  $\text{double} = \lambda x. x + x$
- Example: $\text{allFalse}(x) = \text{false}$
  $\text{allFalse} = \lambda x. \text{false}$
- Example: $\text{multiply}(x,y) = x * y$
  $\text{multiply} = \lambda x. \lambda y. x * y$

The Fixed-Point Theorem

- If $F$ is a semantic function corresponding to a context in our language
  - $F$ is monotonic and continuous (we assert)
  - 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 (not detailed in the lecture):
  1. 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
        $F(\sqcup_k F^k(\lambda x. \perp)) = \sqcup_k F^{k+1}(\lambda x. \perp)$
  by continuity

Denotational Semantics for while

- We can use the fixed-point theorem to write the denotational semantics of while:
  $\text{while } b \text{ do } c = \sqcup_k F^k(\lambda x. \perp)$
  where $F f x = \text{if } b \text{ then } f (\sqcup_k F^k(\lambda x. \perp)) \text{ else } x$

Examples: DS for while

- $\text{while } b \text{ do } c = \sqcup_k F^k(\lambda x. \perp)$
  where $F f x = \text{if } [b] x \text{ then } f (\sqcup_k F^k(\lambda x. \perp)) \text{ else } x$
- $\text{while } b \text{ do } \text{skip} = \langle x, \perp \rangle$
- $\text{while } x \neq 0 \text{ then } x \leftarrow x - 1$
  $F (\lambda x. \perp) x = \langle x = 0 \text{ then } x \leftarrow x - 1 \text{ else } \perp \rangle$
  $F^2 (\lambda x. \perp) x = \langle x = 0 \text{ then } x \leftarrow x - 1 \text{ else } \perp \rangle$
  $F^3 (\lambda x. \perp) x = \langle x = 0 \text{ then } x \leftarrow x - 1 \text{ else } \perp \rangle$
  $\text{LFP}_F = \langle x = 0 \text{ then } 0 \text{ else } \perp \rangle$
- Not easy to find the closed form for general LFPs!
Examples: DS for while

\[ [\text{while } b \text{ do } c] = \bot_k F^k (\lambda x. \bot) \]
where \( F f x = \text{if } [b] x \text{ then } f ([c] x) \text{ else } x \)

- \([\text{while true do skip}] = \lambda x. \bot\]
- \([\text{while } x \neq 0 \text{ then } x := x - 1] \]
  - \( F^1 (\lambda x. \bot) x = \text{if } x = 0 \text{ then } x \text{ else } \bot \)
  - \( F^2 (\lambda x. \bot) x = \text{if } x = 0 \text{ then } x \text{ else } \)
    \[ \begin{cases} x \text{ if } x \leq 1 \text{ else } \bot \end{cases} \]
  - \( F^3 (\lambda x. \bot) x = \text{if } 2 \geq x \geq 0 \text{ then } 0 \text{ else } \bot \)
  - \( \text{LFP}_F = \text{if } x \geq 0 \text{ then } 0 \text{ else } \bot \)
- Not easy to find the closed form for general LFPs!

Discussion: Denotational Semantics

- 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 \([H] 0 \neq \bot\)
- We have derived this for programs with one variable
- 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 domain 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

Congratulations!

You just survived the hardest lectures in 5535.
It's all downhill from here.

Recall: Learning Goals

- Key features of denotational semantics
  - Compositional
  - Meaning is a "math object"
  - When to use DS?
- DS uses \( \bot \) ("bottom") to mean non-termination
- DS uses fixed points and domains to handle while
  - This is the tricky bit
  - Remember SLAM and BLAST?

For Next Time

- Homework 3 out tonight, due Mon Feb 16
**Bonus: Monotonicity, Continuity, and Domains**

### Monotonicity

- 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 \( F f_1 x = n \) of \( F f_0 x \) 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 \( F f_0 x = f_0 x \), \( f_0(0) = \perp \) 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 **monotone**

### Monotonicity of Contexts

- 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

### Continuity

- Consider \( F \) corresponding to a context in our language
- Consider a chain \( g_0 \sqsubseteq \ldots \sqsubseteq g_k \) with \( \bigvee k g_k = G \)
  - Note that \( F g_k \) form a chain also, because \( F \) is monotonic
- We'll show that, for any \( x \), \( F G x = (\bigvee k (F g_k)) x \)
  - 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 \( \max j \) be the maximum such \( j \) for all invocations
  - Thus \( F g_{\max j} x = n \), and \( (\bigvee k (F g_k)) x = n \)
- Similar reasoning for \( F G x = \perp \)

### Monotonicity and Continuity

- A function \( f : D_1 \rightarrow D_2 \) is **monotone** 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 \( x \), in \( D_1 : F (\bigvee i x_i) = \bigvee i (F x_i) \)
- 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

### Example of Domains

- Example: \( D = \mathbb{Z} \rightarrow \mathbb{Z} \)
  - \( f \sqsubseteq g \) iff for all \( n \in \mathbb{Z} : f n = \perp \text{ or } f n = g n \)
  - \( \perp = \lambda n. \perp \)
  - For a chain \( f \), the LUB = \( \lambda n. \text{if } \exists k : f_k x = m \text{ then } m \text{ else } \perp \)
- Example: Take a set \( A \) and a special element \( \perp \), then \( A \sqsubseteq A \cup \{ \perp \} \) is a **flat domain**:
  - \( a \sqsubseteq b \text{ iff } a = \perp \text{ or } a = b \)
  - For a chain \( a_n \), 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 \( D_1 \times D_2 \)