## CS 611
## Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 10
Denotational semantics of IMP
15 Sep 00

## Operational vs. denotational

- Operational semantics
  - meaning of program defined by syntactic transitions
  - structural operational semantics: how to write a recursive-descent interpreter
  - meaning of language terms defined by other language terms

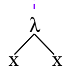
  (λ x x) = (λ x x)
- Denotational semantics
  - defines meaning of program in terms of underlying semantic domain (intrinsic meaning)
  - *semantic function* maps expressions to meanings
  - how to write a *compiler*



## Semantic Function

- Denotational semantics operates on expressions to produce objects that are the meaning of the expression (usually mathematical function)

$\mathscr{C}[\![ (\lambda\, x\ \ x) ]\!] = \lambda\, x \in D.\, x$




$\{ (a, a) \mid a \in D \}$



## Parsing λ's

- Notation for describing a mathematical function of several variables:
  $\lambda xyz\,.\,e \quad = \quad \lambda x \lambda y \lambda z\,.\,e$
- Lambda expression extends as far to the right as possible (like ∀, ∃)
  $\lambda x\lambda y\lambda z\,.\,x\,\lambda w.\,w = \lambda x(\lambda y(\lambda z\,.\,(x\,(\lambda w.\,w))))$
  *not* $((\lambda x(\lambda y(\lambda z\,.\,x)))\,(\lambda w.\,w))$
- Application left-associates:
  $xyz = (xy)z = x(y,z)$
  $f = \lambda xyz\,.\,e \qquad fabc = f(a,b,c)$



## Typed functions

- For mathematical functions, we will usually write the types of the arguments
  $PLUS = \lambda x{\in}Z\,.\,\lambda y{\in}Z.\,x{+}y = \lambda x,y{\in}Z\,.\,x{+}y$
  $PLUS \in Z \rightarrow (Z \rightarrow Z)$
- Type $(T_1 \rightarrow T_2)$ is domain of functions that maps elements from domain $T_1$ to domain $T_2$
- Application associates to the left ⇒ function constructor (→) associates to right

$$Z \rightarrow (Z \rightarrow Z) = Z \rightarrow Z \rightarrow Z$$



## Back to IMP

- Recall IMP has three kinds of expressions:

$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$

$b ::= a_0 \leq a_1 \mid a_0 = a_1 \mid b_0 \wedge b_1 \mid b_0 \vee b_1$

$c ::= X := a_0$ | **skip** | **if** $b_0$ **then** $c_0$ **else** $c_1$ | **while** $b_0$ **do** $c_0$

$a$ : **Aexp**, $b$: **Bexp**, $c$: **Com**

What is the meaning of these three syntactic categories?

Does an element from $a$ mean an integer?

## Natural semantics as functions

- Expression $a$ denotes a unique integer given a particular store $\sigma$ $\langle a,\sigma\rangle \Downarrow n$
- Expression $b$ denotes a unique truth value given a particular store $\sigma$ $\langle b,\sigma\rangle \Downarrow t$
- Command $c$ maps one store into another $\langle c,\sigma\rangle \Downarrow \sigma'$
- Deterministic evaluation $\Rightarrow$ exists functions $\mathcal{A}, \mathcal{B}, \mathcal{C}$ such that

$$\mathcal{A}[\![a]\!]\,\sigma = n \quad \Leftrightarrow \quad \langle a,\sigma\rangle \Downarrow n$$
$$\mathcal{B}[\![b]\!]\,\sigma = t \quad \Leftrightarrow \quad \langle b,\sigma\rangle \Downarrow t$$
$$\mathcal{C}[\![c]\!]\,\sigma = \sigma' \quad \Leftrightarrow \quad \langle c,\sigma\rangle \Downarrow \sigma'$$



## Semantic functions for IMP

- Meaning functions $\mathcal{A}, \mathcal{B}, \mathcal{C}$ translate syntactic expressions into meaning: mathematical functions

$\mathcal{A}[\![a]\!]\,\sigma = n$ $\qquad \mathcal{A} \in$ **Aexp** $\rightarrow (\Sigma \rightarrow \mathbf{N})$

$\mathcal{B}[\![b]\!]\,\sigma = t$ $\qquad \mathcal{B} \in$ **Bexp** $\rightarrow (\Sigma \rightarrow \mathbf{T})$

$\mathcal{C}[\![c]\!]\,\sigma = \sigma'$ $\qquad \mathcal{C} \in$ **Com** $\rightarrow (\Sigma \rightarrow \Sigma)$

- $\mathcal{A}[\![a]\!]$ is denotation of $a$ ($\mathcal{A}[\![a]\!] \in \Sigma \rightarrow \mathbf{N}$)
- $\mathcal{B}[\![b]\!]$ is *denotation* of $b$, $\mathcal{C}[\![c]\!]$ is denotation of $c$



## Arithmetic denotations

- The function $\mathcal{A}$: **Aexp** $\rightarrow \Sigma \rightarrow Z$ is defined using induction on structure of exprs:

$$\mathcal{A}[\![n]\!] = \lambda\,\sigma \in \Sigma\,.\,n$$
$$\mathcal{A}[\![X]\!] = \lambda\,\sigma \in \Sigma\,.\,\sigma\,X$$
$$\mathcal{A}[\![a_0 + a_1]\!] = \lambda\,\sigma \in \Sigma\,.\,\mathcal{A}[\![a_0]\!]\,\sigma + \mathcal{A}[\![a_1]\!]\,\sigma$$
$$\mathcal{A}[\![a_0 - a_1]\!] = \lambda\,\sigma \in \Sigma\,.\,\mathcal{A}[\![a_0]\!]\,\sigma - \mathcal{A}[\![a_1]\!]\,\sigma$$
$$\mathcal{A}[\![a_0 \times a_1]\!] = \lambda\,\sigma \in \Sigma\,.\,\mathcal{A}[\![a_0]\!]\,\sigma \cdot \mathcal{A}[\![a_1]\!]\,\sigma$$



## Semantic function as a set

$$\mathcal{A}[\![n]\!] = \lambda\,\sigma \in \Sigma\,.\,n = \{(\sigma, n) \mid \sigma \in \Sigma\}$$

$$\mathcal{A}[\![a_0 + a_1]\!] = \lambda\,\sigma \in \Sigma\,.\,\mathcal{A}[\![a_0]\!]\,\sigma + \mathcal{A}[\![a_1]\!]\,\sigma$$
$$= \{ (\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \wedge (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}$$

$$\frac{\mathcal{A}[\![a_0]\!] = f_0 \qquad \mathcal{A}[\![a_1]\!] = f_1}{\mathcal{A}[\![a_0 + a_1]\!] = \lambda\,\sigma \in \Sigma\,.\,f_0\,\sigma + f_1\,\sigma}$$



## Boolean denotations

$\mathcal{B} \in$ **Bexp** $\rightarrow \Sigma \rightarrow T$

$\mathcal{B}[\![a_0 = a_1]\!]\sigma$ = if $\mathcal{A}[\![a_0]\!]\,\sigma = \mathcal{A}[\![a_1]\!]\sigma$ then true else false

$\mathcal{B}[\![a_0 \le a_1]\!]\sigma$ = if $\mathcal{A}[\![a_0]\!]\,\sigma \le \mathcal{A}[\![a_1]\!]\sigma$ then true else false

$\mathcal{B}[\![b_0 \wedge b_1]\!]\sigma$ = if $\mathcal{B}[\![b_0]\!]\,\sigma$ and $\mathcal{B}[\![b_1]\!]\sigma$ then true else false

$\mathcal{B}[\![b_0 \vee b_1]\!]\sigma$ = if $\mathcal{B}[\![b_0]\!]\,\sigma$ or $\mathcal{B}[\![b_1]\!]\sigma$ then true else false



## Command denotations

- Some commands do not terminate ($\neg\exists\sigma'\,.\,\langle c,\sigma\rangle \Downarrow \sigma'$)
- Commands are partial functions from states to states ($\Sigma \rightharpoonup \Sigma$)
- Idea: make denotations total by adding special state to represent non-termination: $\perp$
- Domain $\Sigma_\perp$ has elements of $\Sigma \cup \{\perp\}$ (*lift* of $\Sigma$)

$$\mathcal{C} \in \mathbf{Com} \rightarrow \Sigma_\perp \rightarrow \Sigma_\perp$$

- Advantage over large-step: can specify non-terminating behavior

## Command denotations

$\mathcal{C}[\![\textbf{skip}]\!]\,\sigma = \sigma$

$\mathcal{C}[\![X := a]\!]\,\sigma = \sigma[X \mapsto \mathcal{A}[\![a]\!]\,\sigma]$

$\mathcal{C}[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]\,\sigma =$
 $\textit{if } \mathcal{B}[\![b]\!]\sigma \textit{ then } \mathcal{C}[\![c_0]\!]\,\sigma \textit{ else } \mathcal{C}[\![c_1]\!]\sigma$

$\mathcal{C}[\![c_0 ; c_1]\!]\,\sigma = \mathcal{C}[\![c_1]\!]\,(\mathcal{C}[\![c_0]\!]\,\sigma)$

Note: σ could be ⊥ ; need to wrap *if* $\sigma = \perp$ *then* ⊥ *else* ... around :=, **if**, **while** definitions



## while

- What we'd like to write:

$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]\,\sigma =$
 if $\neg\mathcal{B}[\![b]\!]\,\sigma$ then σ
 else $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]\,(\mathcal{C}[\![c]\!]\,\sigma)$

- What's wrong with this?



## while command

$\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]\,\sigma =$
 if $\mathcal{B}[\![b]\!]\,\sigma$ then σ
 else $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]\,(\mathcal{C}[\![c]\!]\,\sigma)$

- This is an *equation*, not a definition (induction is not well-founded)

$\mathcal{C}[\![\text{while } b \text{ do } c]\!] =$
 $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\,\sigma \;\&\; (\sigma, \sigma') \in \mathcal{C}[\![\text{while } b \text{ do } c]\!] \circ \mathcal{C}[\![c]\!]\}$
$\cup \{(\sigma, \sigma) \mid \neg\mathcal{B}[\![b]\!]\,\sigma\}$



## Denotation as a fixed point

$\mathcal{C}[\![\text{while } b \text{ do } c]\!] =$
 $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\,\sigma \;\&\; (\sigma, \sigma') \in \mathcal{C}[\![\text{while } b \text{ do } c]\!] \circ \mathcal{C}[\![c]\!]\}$
$\cup \{(\sigma, \sigma) \mid \neg\mathcal{B}[\![b]\!]\,\sigma\}$

Define $\Gamma(f)$ where $f$ is a command denotation

$\Gamma = \lambda f \in \Sigma_\perp \to \Sigma_\perp .\ \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\,\sigma \;\&\; (\sigma, \sigma') \in f \circ \mathcal{C}[\![c]\!]\}$
 $\cup \{(\sigma, \sigma) \mid \neg\mathcal{B}[\![b]\!]\,\sigma\}$
 $= \lambda f \in \Sigma_\perp \to \Sigma_\perp .\ \text{if } \mathcal{B}[\![b]\!]\,\sigma \text{ then } \sigma \text{ else } f(\mathcal{C}[\![c]\!]\,\sigma)$

$\mathcal{C}[\![\text{while } b \text{ do } c]\!] = \Gamma(\mathcal{C}[\![\text{while } b \text{ do } c]\!])$

Denotation of while is fixed point of Γ



## Denotation of while

$\mathcal{C}[\![\text{while } b \text{ do } c]\!]\,\sigma =$

$\textit{fix}\,(\lambda f.\ \text{if } \mathcal{B}[\![b]\!]\,\sigma \text{ then } \sigma \text{ else } f(\mathcal{C}[\![c]\!]\,\sigma))$

- Question: how do we define least fixed point operator *fix* for domain $\Sigma_\perp \to \Sigma_\perp$ ?
- Answer: next lecture