# CS 611: Lecture 11

Denotational Semantics

September 22, 1999

Cornell University Computer Science Department
Andrew Myers

## Limits of operational semantics

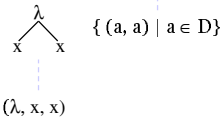
- Operational semantics
  - meaning of program defined by *actions*
  - structural operational semantics: how to write a recursive-descent interpreter
  - translates language terms into other language terms
  - $(\lambda\ x\ x) = (\lambda\ x\ x)$
- Denotational semantics
  - meaning of program defined in terms of underlying semantic domain
  - translates program into a mathematical function
  - how to write a *compiler*
  - $(\lambda\ x\ x) = \lambda\ x\ .\ x$



## Functions

- Denotational semantics operates on expressions to produce functions that are the meaning of the expression

$C[\![ (\lambda\ x:\ x) ]\!] = \lambda\ x\ .\ x$






## Back to IMP

- Recall IMP has three kinds of expressions:

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

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

$c ::= X := a_0 \mid \text{skip} \mid \text{if } b_0 \text{ then } c_0 \text{ else } c_1 \mid \text{while } b_0 \text{ do } c_0$

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

What is the meaning of these three syntactic categories?

Does an element from $a$ mean an integer?



## Semantic functions for IMP

- Expression $a$ denotes an integer given a particular store $\sigma$
- Expression $b$ denotes an truth value ($\mathbf{T}$) given a particular store $\sigma$
- Command $c$ maps one store into another

$\mathcal{A}[\![a]\!]\sigma = n$ $\quad\quad$ $\mathcal{A}$ : Aexp $\rightarrow(\Sigma\rightarrow\mathbf{N})$

$\mathcal{B}[\![b]\!]\sigma = t$ $\quad\quad$ $\mathcal{B}$: Bexp $\rightarrow (\Sigma\rightarrow\mathbf{T})$

$\mathcal{C}[\![c]\!]\sigma = \sigma'$ $\quad\quad$ $\mathcal{C}$: Com $\rightarrow (\Sigma\rightarrow \Sigma)$



## Lambda notation

- 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 $\forall, \exists$)
  $\lambda x\lambda y\lambda z\ .\ x\ \lambda w.\ w = \lambda x(\lambda y(\lambda z\ .\ (x\ (\lambda w.\ w))))$
  $not\ ((\lambda x(\lambda y(\lambda z\ .\ x)))\ (\lambda w.\ w))$
- Application left-associates:
  $xyz = (xy)z = x(y,z)$
  $f = \lambda xyz\ .\ e \quad\quad\quad\quad fabc = f(a,b,c)$

## Typed functions

- For mathematical functions, we will usually write the types of the arguments

$$PLUS = \lambda x{:}Z\,.\,\lambda y{:}Z.\; x+y = \lambda x,y{:}Z\,.\; x+y$$

$$PLUS\;:\; Z \to (Z \to Z)$$

- Type $(T_1 \to T_2)$ is a function that maps elements from domain $T_1$ to domain $T_2$
- Application associates to the left $\Rightarrow$ function constructor $(\to)$ associates to right

$$Z \to (Z \to Z) = Z \to Z \to Z$$



## Denotations

Semantic (meaning) Functions:

$\mathcal{A}[\![a]\!]\,\sigma = n$ $\quad\quad$ $\mathcal{A}$: Aexp $\to\Sigma\to Z$

$\mathcal{B}[\![b]\!]\,\sigma = t$ $\quad\quad$ $\mathcal{B}$: Bexp $\to\Sigma\to T$

$\mathcal{C}[\![c]\!]\,\sigma = \sigma$ $\quad\quad$ $\mathcal{C}$: Com $\to\Sigma\to\Sigma$

$\mathcal{A}[\![a]\!] : \Sigma\to N$, $\mathcal{B}[\![b]\!] : \Sigma\to T$, $\mathcal{C}[\![c]\!] : \Sigma\to\Sigma$

$\mathcal{A}[\![a]\!]$ is the **denotation** of the arithmetic expression $a$

The expression $a$ **denotes** the function from states to integers $\mathcal{A}[\![a]\!]$



## Arithmetic denotations

- The function $\mathcal{A}$ : Aexp $\to\Sigma\to Z$ is defined using structural induction:

$$\mathcal{A}[\![n]\!] = \lambda\sigma:\Sigma\,.\,n$$

$$\mathcal{A}[\![X]\!] = \lambda\sigma:\Sigma\,.\,\sigma X$$

$$\mathcal{A}[\![a_0 + a_1]\!] = \lambda\sigma:\Sigma\,.\,\mathcal{A}[\![a_0]\!]\;\sigma + \mathcal{A}[\![a_1]\!]\;\sigma$$

$$\mathcal{A}[\![a_0 - a_1]\!] = \lambda\sigma:\Sigma\,.\,\mathcal{A}[\![a_0]\!]\;\sigma - \mathcal{A}[\![a_1]\!]\;\sigma$$

$$\mathcal{A}[\![a_0 \times a_1]\!] = \lambda\sigma:\Sigma\,.\,\mathcal{A}[\![a_0]\!]\;\sigma \cdot \mathcal{A}[\![a_1]\!]\;\sigma$$



## Semantic function as a set

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

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

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



## Boolean denotations

$\mathcal{B}$ : Bexp $\to\Sigma\to T$

$$\mathcal{B}[\![a_0 = a_1]\!] = \lambda\sigma:\Sigma\,.\,\text{if } \mathcal{A}[\![a_0]\!]\,\sigma = \mathcal{A}[\![a_1]\!]\sigma \text{ then } \textbf{true} \text{ else } \textbf{false}$$

$$\mathcal{B}[\![a_0 \le a_1]\!] = \lambda\sigma:\Sigma\,.\,\text{if } \mathcal{A}[\![a_0]\!]\;\sigma \le \mathcal{A}[\![a_1]\!]\sigma \text{ then } \textbf{true} \text{ else } \textbf{false}$$

$$\mathcal{B}[\![b_0 \wedge b_1]\!] = \lambda\sigma:\Sigma\,.\,\text{if } \mathcal{B}[\![b_0]\!]\,\sigma \text{ and } \mathcal{B}[\![b_1]\!]\sigma \text{ then } \textbf{true} \text{ else } \textbf{false}$$

$$\mathcal{B}[\![b_0 \vee b_1]\!] = \lambda\sigma:\Sigma\,.\,\text{if } \mathcal{B}[\![b_0]\!]\,\sigma \text{ or } \mathcal{B}[\![b_1]\!]\sigma \text{ then } \textbf{true} \text{ else } \textbf{false}$$



## Command denotations

- Commands are partial functions from states to states ($\mathcal{C}$ : Com $\to\Sigma\to\Sigma$ )

$$\mathcal{C}[\![\text{skip}]\!]\;\sigma = \sigma$$

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

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

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

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

## while command

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

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

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



## While denotation as a fixed point

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

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

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

$f = \Gamma(f)$

Denotation of while is fixed point of $\Gamma$



## Finding the fixed point

- Denotation for while is defined as *least fixed point* of the equation $f = \Gamma(f)$
- Least fixed point is function that maps fewest states : while command diverges whenever it can while satisfying $f = \Gamma(f)$
- Can set up definition of f as inference rules:

$$\frac{\mathcal{B}[\![b]\!]\ \sigma = \text{true}}{(\sigma,\sigma) \in f} \qquad \frac{\mathcal{B}[\![b]\!]\ \sigma = \text{false} \quad (\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \quad (\sigma'',\sigma') \in f}{(\sigma,\sigma') \in f}$$

- Rule induction: least fixed point exists



## Denotation of while

$\mathcal{C}[\![\text{while } b \text{ do } c]\!]\ \sigma =$
$\quad fix\ (\lambda f.\ \text{if } \mathcal{B}[\![b]\!]\ \sigma \text{ then } \sigma \text{ else } f\ (\mathcal{C}[\![c]\!]\ \sigma))$