CS 611

Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 1: Introduction
25 Aug 00

Goals

- Deeper understanding of PL's
- Broader exposure to PL's
- *Not* a survey course

Why study PL?

- Elegant math, practical impact
  - a study of expressive power
  - caveat: comfort with logic, proofs, Ch. 1
- Better language design
  - how to specify
  - how to prove correct
  - embarrassing questions to ask
- Better language implementation
  - efficient implementation (more in CS 412)
  - correct implementation
- Better programmer
  - understand your tools (and which ones to use)

Schedule

• Operational semantics	5	
• Inductive proofs	3	
• Lambda calculus	3	*dynamic*
• Denotational semantics	4	
• Interesting language features	8	
• Type systems	4	*static*
• Interesting types	8	
• Miscellaneous topics	3	

Workload

- Sign-up sheet
- Readings (see course schedule)
- 6 homeworks (about half with programming component, in ML)
- Scribe 3-4 lectures (in pairs)
  - we will provide TeX template
  - meet with me for feedback
- Prelim: tentatively Oct. 26, 7-9:30PM
- Final exam: Dec. 7, 12-2:30PM

Course Staff

- Lecturer: Andrew Myers
  andru@cs.cornell.edu
  Upson 4124
  Office hours: Wed 3-4PM
- TA: Matthew Fluet
  Email: cs611@cs.cornell.edu
  Upson 4162
  Office hours: TBA

Web site: courses.cs.cornell.edu/cs611

Texts

- Required:
  - Winskel, *The Formal Semantics of Programming Languages*
- Recommended:
  - Gunter, *Semantics of Programming Languages*
  - Mitchell, *Foundations of Programming Languages*
  - Gifford (will be placed on-line; may be used only for this course)

IMP

- Winskel, Ch. 2
- Simple imperative language (vs. functional)
- IMP program is a *command*
  - **skip**
  - $X := a$
  - $c_0; c_1$
  - **if** b **then** c_0 **else** c_1
  - **while** b **do** c
- Variables (X) take integer values
- Arithmetic exprs a, boolean expressions b

Example: GCD

while x ≠ y **do**
  if x < y **then**
    y := y – x
  else
    x := x – y
  end

- Turing-complete (barely): no functions, data structures

Issues

- What is a legal program in IMP?
  - defined by abstract syntax
- What is a legal program execution in IMP?
  - structural operational semantics
- Other properties of interest
  - expressions terminate, commands may not
  - programs never "crash"
  - evaluation is deterministic

Defining Syntax

- Three *syntactic sets*:
  - **Aexp**: set of legal arithmetic expressions a
  - **Bexp**: legal boolean expressions b
  - **Com**: legal commands c
- Define legal programs inductively using Backus-Naur form (BNF):

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

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

$c ::=$ **skip** $\mid X := a \mid c_0 ; c_1 \mid$ **if** b **then** c_0 **else** $c_1 \mid$ **while** b **do** c

$X \in$ **Loc**, $n \in$ **Z**

Abstract Syntax

- This course: not about parsing
- Elements of syntactic set are *parse trees*, not concrete syntax

3 + 4 * x = 

≠ "3+4*x"

- But…will write expressions that look concrete
  - parentheses used to disambiguate parsing when necessary: (3+4)*5 vs. 3 +(4*5)
  - not part of abstract syntax

Operational Semantics

- Any element of **Com** is a legal program. How does it evaluate?
- Defining process of program evaluation: *operational semantics*
- Java language reference manual: verbose, long operational semantics
- Structural operational semantics: legal executions correspond to proofs
  - compact
  - convenient for proving properties of language

Configurations

- A *configuration* : what we need to know about a running program to define how it executes
- Input to program is a *state* or *memory* mapping variables onto integers

$$\sigma : \mathbf{Loc} \rightarrow \mathbf{Z}$$

- Output from program: state σ'
- Command configuration: $\langle c, \sigma \rangle$

Large-step evaluation

- Large-step semantics define complete evaluation of a program or subexpression

$\langle c, \sigma \rangle \Downarrow \sigma'$ = "Command c starting in state σ terminates in state σ'"

$\langle a, \sigma \rangle \Downarrow n$ = "expression a evaluates to n"

$\langle b, \sigma \rangle \Downarrow t$ = "expression b evaluates to t"

Some evaluations

$\langle X, \sigma \rangle \Downarrow \sigma(X)$ (for any σ, X)

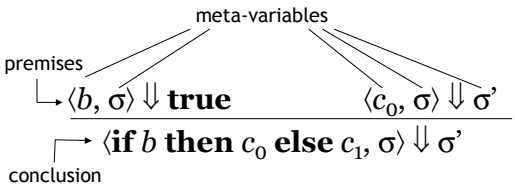
$\langle n, \sigma \rangle \Downarrow n$ (for any σ, X)

$\langle n_0 + n_1, \sigma \rangle \Downarrow n_2$ (for any n_0, n_1, n_2, X where n_2 is sum of n_0, n_1)

$\langle \mathbf{skip}, \sigma \rangle \Downarrow \sigma$ (for any σ, X)

$\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \Downarrow \sigma'$ if $\langle b, \sigma \rangle \Downarrow \mathbf{true}$ and $\langle c_0, \sigma \rangle \Downarrow \sigma'$ (for any ...)

As inference rules



$$\frac{}{\langle n, \sigma \rangle \Downarrow n}\ \text{axiom}$$

Execution as proof

- Legal executions = those that can be proved correct inductively
- Proof = *proof tree* where every step is application of an inference rule
- Execution = depth-first walk of proof tree
- Collection of inference rules: *proof system*

$$\frac{\dfrac{\langle x, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow 1 \quad \langle y, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow 2}{\langle x < y, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow \mathbf{true}} \quad \frac{\langle 0, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow 0}{\langle x := 0, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow [x\mapsto 0, y\mapsto 2]}}{\langle \mathbf{if}\ x < y\ \mathbf{then}\ x := 0\ \mathbf{else}\ \mathbf{skip}, [x\mapsto 1, y\mapsto 2] \rangle \Downarrow [x\mapsto 0, y\mapsto 2]}$$

Applying rules

- Inference rule represents a large (infinite) set of *rule instances* in which meta-variables are consistently substituted

$$\frac{}{\langle n, \sigma\rangle \Downarrow n} \Rightarrow \frac{}{\langle 0, \sigma\rangle \Downarrow 0} \quad , \quad \frac{}{\langle 1, \sigma\rangle \Downarrow 1}, \ldots$$

$$\frac{\langle b, \sigma\rangle \Downarrow \textbf{true} \quad \langle c_0, \sigma\rangle \Downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \Downarrow \sigma'} \Rightarrow ? \quad \frac{\langle 0{=}1, \sigma\rangle \Downarrow \textbf{true} \quad \langle \textbf{skip}, \sigma\rangle \Downarrow \sigma}{\langle \textbf{if } 0{=}1 \textbf{ then skip else } \ldots, \sigma\rangle \Downarrow \sigma}$$