Programming Language Design (CS 422)

Elsa L Gunter
2112 Siebel Center, UIUC
egunter@illinois.edu

http://courses.engr.illinois.edu/cs422/sp2016

Slides based in part on previous lectures by Grigore Roșu

January 20, 2016

Contact Information

- Office: 2112 Siebel Center
- Office hours:
  - Wednesday 12:30pm – 1:45pm
  - Thursday 9:00am – 9:50am
  - Also by appointment
- Email: eguntere@illinois.edu

Course Website

- **main** page - summary of news items
- **policy** - rules governing the course
- **lectures** - syllabus, slides and example code
- **mps** - Information about homework
- **unit projects** - for 4 credit students
- **resourses** - papers, tools, and helpful info
- **faq** - answers to some general questions about the course and course resources

Some Course References

- **No Required Textbook**

- Lecture Notes of Grigore Rosu, found in Resources
- *Essentials of Programming Languages (2nd Edition)* by Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes, MIT Press 2001
- *The Formal Semantics of Programming Languages: An Introduction* by Glynn Winskel. MIT Press, 1993.
- *Concrete Semantics With Isabelle/HOL*, by Tobias Nipkow and Gerwin Klein. Springer, 2014.

Course Grading

- Homeworks – 30%
  - Two kinds: Handwritten and Machine Processed
  - Handwritten turned in as pdfs
  - MPs turned in as plain text files
  - Both subimtted via course svn student directories
- Midterm – 30%
- Final – 40%
- Unit Project
  - Only for 4-credit graduate students
  - Worth 25%, with all other parts scaled down accordingly

Collaboration on Assignments

- You may discuss homeworks and their solutions with others
- You may work in groups, but you must list members with whom you worked
- Each student must turn in their own solution separately
- You may look at examples from class and other similar examples from any source
- Note: University policy on plagiarism still holds
- Problems from homework may appear verbatim, or with some modification on exams

Default Unit Project

- Design, formalize and create an interpreter for a new language with specified features.
- Will be an extension of previously describe language.
- Students may develop alternate projects with instructor approval.

Course Objectives

- Learn different methods of specifying the meaning of language features and how to reason about them
  - Structural Operational Semantics
  - Tranistion Semamtics
  - CHAM and K
  - denotational semantics
  - axiomatic semantics

Courxe Objectives

- Learn to specify different language features
  - Imperative Features
  - Funtional Features
  - Type Systems
  - Object Oriented Features

Semantics

- Expresses the meaning of syntax
- Static semantics
  - Meaning based only on the form of the expression without executing it
  - Usually restricted to type checking / type inference
- Dynamic semantics
  - Method of describing meaning of executing a program
  - Used for formal reasoning about programs and languages
  - Several different types:
    - Operational Semantics
    - Axiomatic Semantics
    - Denotational Semantics

Dynamic Semantics

- Different languages better suited to different types of semantics
- Different types of semantics serve different purposes

Operational Semantics

- Start with a simple notion of machine
- Describe how to execute (implement) programs of language on virtual machine, by describing how to execute each program statement (ie, following the structure of the program)
- Meaning of program is how its execution changes the state of the machine
- Useful as basis for implementations

Axiomatic Semantics

- Also called Floyd-Hoare Logic
- Based on formal logic (first order predicate calculus)
- Axiomatic Semantics is a logical system built from axioms and inference rules
- Mainly suited to simple imperative programming languages

Axiomatic Semantics

- Used to formally prove a property (*post-condition*) of the state (the values of the program variables) after the execution of program, assuming another property (*pre-condition*) of the state before execution
- Written :

$$\{Precondition\}Program\{Postcondition\}$$

- Source of idea of loop invariant

Denotational Semantics

- Construct a function M assigning a mathematical meaning to each program construct
- Lambda calculus often used as the range of the meaning function
- Meaning function is compositional: meaning of construct built from meaning of parts
- Mainly used for proving properties of programs

Natural Semantics

- Aka "Big Step Semantics"
- Originally introduced by Giles Kahn
- Provide value for a program by rules and derivations
- Rule conclusions look like

$$(C, m) \Downarrow m'$$

or

$$(E, m) \Downarrow v$$

- Type derivation rules often take very similar shape

Simple Imperative Programming Language #1

$$
\begin{array}{lll}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
 & & \mid E < E \mid E = E \\
C & ::= & \text{skip} \mid C; C \mid \{C\} \mid I ::= E \\
 & & \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
 & & \mid \text{while } B \text{ do } C \text{ od}
\end{array}
$$

Natural Semantics of Atomic Expressions

Let $m : \text{Identifiers} \rightharpoonup \text{Values}$ be a partial function supplying values for program variable names

Identifiers:	$(I, m) \Downarrow m(I)$
Numerals are values:	$(N, m) \Downarrow N$
Booleans:	$(\text{true}, m) \Downarrow \text{true}$
	$(\text{false}, m) \Downarrow \text{false}$

Boolean Expressions

$$\frac{(B, m) \Downarrow \text{false}}{(B \& B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \& B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{true}} \qquad \frac{(B, m) \Downarrow \text{false} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(\text{not } B, m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}}$$

Relations

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b}$$

- By $U \sim V = b$, we mean does (the meaning of) the relation $\sim$ hold on the meaning of U and V
- May be specified by a mathematical expression/equation or rules matching U and V

Arithmetic Expressions

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \oplus V = N}{(E \oplus E', m) \Downarrow N}$$

where N is the specified value for $U \oplus V$

Commands

Skip: $(skip, m) \Downarrow m$

Assignment: $$\frac{(E, m) \Downarrow V}{(I ::= E, m) \Downarrow m[I \leftarrow V]}$$

Sequencing: $$\frac{(C, m) \Downarrow m' \quad (C', m') \Downarrow m''}{(C; C', m) \Downarrow m''}$$

Block: $$\frac{(C, m) \Downarrow m'}{(\{C\}, m) \Downarrow m'}$$

where $m[I \leftarrow V](J) = \begin{cases} V & \text{if } J = I \\ m(J) & \text{otherwise} \end{cases}$

If Then Else Command

$$\frac{(B, m) \Downarrow \text{true} \quad (C, m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

$$\frac{(B, m) \Downarrow \text{false} \quad (C', m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

While Command

$$\frac{(B, m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od }, m) \Downarrow m}$$

$$\frac{(B, m) \Downarrow \text{true} \quad (C, m) \Downarrow m' \quad (\text{while } B \text{ do } C \text{ od }, m') \Downarrow m''}{(\text{while } B \text{ do } C \text{ od }, m) \Downarrow m''}$$

Simple Imperative Programming Language #2

$$
\begin{array}{lcl}
I & \in & \mathit{Identifiers} \\
N & \in & \mathit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \mid I ::= E \\
B & ::= & \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
 & & \mid E < E \mid E = E \\
C & ::= & \text{skip} \mid C ; C \mid \{C\} \mid E \\
 & & \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
 & & \mid \text{while } B \text{ do } C \text{ od}
\end{array}
$$