Contact Information

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

Course Grading

- Homeworks – 30%
  
  - Two kinds: Handwritten and Machine Processed
  - Handwritten turned in as pdfs
  - MPs turned in as plain text files
  - Both submitted 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
  - Transition Semantics
  - CHAM and K
  - denotational semantics
  - axiomatic semantics

Course Objectives

- Learn to specify different language features
  - Imperative Features
  - Functional 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: \( \{ \text{Precondition} \} \text{Program} \{ \text{Postcondition} \} \)
- 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 Gilles Kahn
- Provide value for a program by rules and derivations
- Rule conclusions look like
  
  \[
  (C, m) \Downarrow m' \\
  \text{or} \\
  (E, m) \Downarrow v
  \]
- Type derivation rules often take very similar shape

Simple Imperative Programming Language #1

\[
I \in \text{Identifiers} \\
N \in \text{Numerals} \\
E ::= N \mid I \mid E + E \mid E \times E \mid E - E \\
B ::= \text{true} \mid \text{false} \mid B \& B \mid B \text{ or } B \mid \text{not } B \\
| \ E < E \mid E = E \\
C ::= \text{skip} \mid C ; C \mid \{ C \} \mid I ::= E \\
| \text{if } B \text{ then } C \text{ else } C \text{ fi} \\
| \text{while } B \text{ do } C \text{ od}
\]

Natural Semantics of Atomic Expressions

Let \( m : \text{Identifiers} \rightarrow \text{Values} \) 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

\[
(B, m) \downarrow \text{false} \quad (B', m) \downarrow \text{true} \\
(B \& B', m) \downarrow \text{false} \quad (B \& B', m) \downarrow b \\
(B, m) \downarrow \text{true} \quad (B', m) \downarrow \text{false} \\
(B \text{ or } B', m) \downarrow \text{true} \quad (B \text{ or } B', m) \downarrow b \\
(B, m) \downarrow \text{true} \quad (not B, m) \downarrow \text{false} \\
(not B, m) \downarrow \text{true}
\]

### Relations

\[
(E, m) \downarrow U \\
(E', m) \downarrow V \\
U \sim V = b
\]

- 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

\[
(E, m) \downarrow U \\
(E', m) \downarrow V \\
U \oplus V = N
\]

where \(N\) is the specified value for \(U \oplus V\)

### Commands

**Skip:**

\[
(skip, m) \downarrow m
\]

**Assignment:**

\[
(I ::= E, m) \downarrow m'[I \leftarrow V]
\]

**Sequencing:**

\[
(C, m) \downarrow m' \\
(C', m') \downarrow m''
\]

\[
(C, m) \downarrow m'
\]

**Block:**

\[
\{C, m\} \downarrow m'
\]

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

### If Then Else Command

\[
(B, m) \downarrow \text{true} \\
(C, m) \downarrow m'
\]

\[
(B, m) \downarrow \text{false} \\
(C', m) \downarrow m'
\]

\[
(\text{if } B \text{ then } C \text{ else } C', m) \downarrow m'
\]

### While Command

\[
(B, m) \downarrow \text{false} \\
(\text{while } B \text{ do } \text{C} \text{ od }, m) \downarrow m
\]

\[
(B, m) \downarrow \text{true} \\
(C, m) \downarrow m' \\
(\text{while } B \text{ do } \text{C} \text{ od }, m') \downarrow m''
\]

\[
(\text{while } B \text{ do } \text{C} \text{ od }, m) \downarrow m''
\]
Simple Imperative Programming Language #2

$I \in \text{Identifiers}$

$N \in \text{Numerals}$

$E ::= N | I | E + E | E \times E | E - E | I ::= E$

$B ::= \text{true} | \text{false} | B \& B | B \text{ or } B | \text{not } B$

$C ::= \text{skip} | C ; C | \{ C \} | E$

| if $B$ then $C$ else $C$ fi
| while $B$ do $C$ od