1. Administrative Issues

2. Course Goals and Outline

3. Types and Evaluation
Teaching Staff

- **Lectures:** Professor Michael Erdmann

- **Sec E,** 11:30–12:20 in Gates 4215:
  
  **Mark Hahnenberg**

- **Sec A,** 12:30–1:20 in Wean Hall 5302:
  
  **Ian Voysey**

- **Sec B,** 1:30–2:20 in Wean Hall 5302:
  
  **Pablo Chavez**

- **Sec C,** 2:30–3:20 in Porter Hall 226C:
  
  **Roger Su**

- **Sec D,** 3:30–4:20 in Wean Hall 5312:
  
  **Nathan Herzing**
Course Philosophy

Computation is Functional.

Programming is an explanatory linguistic process.
Computation is Functional

values : types

expressions

Functions map values to values
**Imperative** vs. **Functional**

**Command**
- executed
- has an effect
  
  \[ x := 5 \]  
  (state)

**Expression**
- evaluated
- no effect
  
  \[ 3 + 4 \]  
  (value)
Programming as Explanation

Problem statement

- high expectation to explain precisely & concisely
- invariants
- specifications
- proofs of correctness
- code

Analyze, Decompose & Fit, Prove
Textbooks


Links

- Web: http://www.cs.cmu.edu/~me/212/

- Directory: /afs/andrew/course/15/212sp/

- Bulletin Boards:

  academic.cs.15-212.discuss

  academic.cs.15-212.announce
SML Implementation for the course

SML/NJ

- From Andrew:

  /usr/local/bin/sml

- Personal copies available at:

  http://www.smlnj.org/index.html

(Further details underneath the course webpage.)
Course Requirements

- Attend lectures and recitations

- Do (and hand in!) homework assignments (50%)

- Midterm (20%), in class (Feb 17)

- Final (30%), 3 hours
Assignments

• 6 assignments (roughly 2 weeks each, 
  \[ 50 + 5 \times 100 = 550 \] points)

• Handed out mid-week, electronically

• Written and programming parts

• Programs due Wednesday at 2:12AM

• A program is like an essay!
Course Goals

• 211: Data structures and algorithms

• 212: Advanced programming techniques
  – Decomposing problems
  – Combining solutions
  – Reason about program correctness
Concepts

• Functional Programming
• Induction and recursion, loop invariants
• Data abstraction
  – data types
  – representation invariants
• Control abstraction
  – higher-order functions
  – exceptions and continuations
• Types and modularity
• Mutable data structures, objects
• Streams, demand-driven programming
Further Topics

• Symbolic computation

• Game search

• Grammars and parsing

• Type-checking and evaluation

• Computability
The ML Language

- Carrier for concepts

- Supports (in the language):
  - recursion
  - user-declared data types
  - concrete and abstract types
  - higher-order functions
  - exceptions and continuations

- Advanced module system
Characteristics of ML

- Statically typed

- “Well-typed programs cannot go wrong”

- Mathematically defined via evaluation of expressions to values

- Much later and infrequent: effects

- Computation with symbolic values via pattern matching
Interacting with ML

• You present ML with an expression.

→ The ML compiler typechecks the expression.

• The ML compiler evaluates the expression.

• The ML compiler prints the resulting value.
Standard ML of New Jersey, Version 110.
[CM; autoload enabled]

- 3 + 5;
val it = 8 : int

- use "sample.sml";
  [opening sample.sml]
val it = 8 : int
val it = () : unit

-
Defining ML (Effect-Free Fragment)

- Types $t$
- Expressions $e$
- Values $v$ (subset of expressions)
Expressions

Every well-formed ML expression $e$

- has a type $t$, written as $e : t$
- may have a value $v$, written as $e \rightarrow v$.
- may have an effect (not for our effect-free fragment)

Example: $$(3 + 4) \times 2 : \text{int}$$

$$(3 + 4) \times 2 \rightarrow 14$$
Every well-formed ML expression $e$

- has a type $t$, written as $e : t$
- may have a value $v$, written as $e \rightarrow v$.
- may have an effect (not for our effect-free fragment)

Evaluating Expressions:

- $e \xrightarrow{1} e'$ \hspace{1cm} $e$ reduces to $e'$ in one step
- $e \xrightarrow{k} e'$ \hspace{1cm} $e$ reduces to $e'$ in $k$ steps
- $e \Rightarrow e'$ \hspace{1cm} $e$ reduces to $e'$ in 0 or more steps
- $e \leftarrow v$ \hspace{1cm} $e$ evaluates to $v$
Examples:

\[(3 + 4) \times 2\]
\[7 \times 2\]
\[14\]

\[(3 + 4) \times (2 + 1)\]
\[21\]
Notation Recap

e : t  "e has type t"

\( e \Rightarrow e' \)  "e reduces to e'"

\( e \leftrightarrow v \)  "e evaluates to v"
Types in ML

Basic types:
int, real, bool, char, string

Constructed types:
product types
function types
user-defined types
Integers, Expressions

Types int

Values \ldots, -1, 0, 1, \ldots,

that is, \( n \) for every integer \( n \).

Expressions \( e_1 + e_2, \ e_1 - e_2, \ e_1 \ast e_2, \)

\( e_1 \div e_2, \ e_1 \mod e_2, \ etc. \)

Example: \( \sim 4 \ast 3 \)
Integers, Typing

Typing Rules

- $\overline{n} : \text{int}$
- $e_1 + e_2 : \text{int}$ if $e_1 : \text{int}$ and $e_2 : \text{int}$

similar for other operations.

Example:

$$(3 + 4) \times 2 : \text{int}$$

Why?

$3 + 4 : \text{int}$ and $2 : \text{int}$

Why?

$3 : \text{int}$ and $4 : \text{int}$
Integers, Evaluation

Evaluation Rules

- \( e_1 + e_2 \xrightarrow{1} e'_1 + e_2 \) if \( e_1 \xrightarrow{1} e'_1 \)

- \( \overline{n_1} + e_2 \xrightarrow{1} \overline{n_1} + e'_2 \) if \( e_2 \xrightarrow{1} e'_2 \)

- \( \overline{n_1} + \overline{n_2} \xrightarrow{1} \overline{n_1 + n_2} \)
Booleans, Typing

Types bool

Values true and false

Expressions \( \text{if } e_1 \text{ then } e_2 \text{ else } e_3, e_1 > e_2, \ldots \)

Typing Rules

\( \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : t \)

\( \text{if } e_1 : \text{bool} \)

and \( e_2 : t \)

and \( e_3 : t \)

Example: \( \text{if } (3 > 4) \text{ then } "foo" \text{ else } "bar" : \text{string} \)
Evaluation Rules

\[ \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \]
\[ \overset{1}{\Rightarrow} \text{ if } e'_1 \text{ then } e_2 \text{ else } e_3 \]
\[ \text{if } e_1 \overset{1}{\Rightarrow} e'_1 \]

\[ \text{if true then } e_2 \text{ else } e_3 \overset{1}{\Rightarrow} e_2 \]

\[ \text{if false then } e_2 \text{ else } e_3 \overset{1}{\Rightarrow} e_3 \]
Products, Expressions

Types \( t_1 \times t_2 \) for any type \( t_1 \) and \( t_2 \).

Values \( (v_1, v_2) \) for values \( v_1 \) and \( v_2 \).

Expressions \( (e_1, e_2) \), \( \#1 e, \#2 e \)

Examples: \( (3 + 4, \text{true}) \)

\( (1.0, \sim 15.6) \)

\( (8, 5, \text{false}, \sim 2) \)
Products, Typing

Typing Rules

- \((e_1, e_2) : t_1 \times t_2\)
  - if \(e_1 : t_1\)
  - and \(e_2 : t_2\)

- \#1 e : t_1
  - if \(e : t_1 \times t_2\) for some \(t_2\).

- \#2 e : t_2
  - if \(e : t_1 \times t_2\) for some \(t_1\).

Example: \((3+4, \text{true}) : \text{int} \times \text{bool}\)
Evaluation Rules

- \((e_1, e_2) \xrightarrow{1} (e'_1, e_2)\) if \(e_1 \xrightarrow{1} e'_1\)

- \((v_1, e_2) \xrightarrow{1} (v_1, e'_2)\) if \(e_2 \xrightarrow{1} e'_2\)

- \(#1 e \xrightarrow{1} #1 e'\) if \(e \xrightarrow{1} e'\)

- \(#1 (v_1, v_2) \xrightarrow{1} v_1\)

- \(#2 e \xrightarrow{1} #2 e'\) if \(e \xrightarrow{1} e'\)

- \(#2 (v_1, v_2) \xrightarrow{1} v_2\)
Declarations
Environments
Scope
Concrete

Type

Definitions
Next time ...

Functions