15-212

Spring 2011

`http://www.cs.cmu.edu/∼me/212/`

Lecture 1

Michael Erdmann

January 11, 2011

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		Expression
↓		↓
• executed • has an effect		• evaluated • no effect
x := 5		3 + 4
(state)		(value)

Programming as Explanation

Problem statement

↓

high expectation to explain precisely & concisely:

- invariants
- specifications
- proofs of correctness
- code

Analyze, Decompose & Fit, Prove

Textbooks

- **Required:** M. R. Hansen and H. Rischel, *Introduction to Programming Using SML*, Addison-Wesley, 1999.
- **Web:** Robert Harper, *Programming in Standard ML*, on-line, 2007.
- **Supplementary:** Lawrence C. Paulson, *ML for the Working Programmer*, 2nd ed, Cambridge University Press, 1996.

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, $\mathsf{50} + \mathsf{5} * \mathsf{100} = \mathsf{550}$ 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**.

```
% /afs/andrew/course/15/212sp/bin/smlnj

Standard ML of New Jersey, Version 110.
[CM; autoload enabled]

- 3 + 5;                    ← keyboard
val it = 8 : int

- use "sample.sml";         ← keyboard
[opening sample.sml]          (load file)
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 $e \hookrightarrow v$.
- may have an effect (not for our effect-free fragment)

Example: (3+4)*2 : int

(3+4)*2 $\hookrightarrow$ 14

Expressions

Every well-formed ML expression `e`

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

Evaluating Expressions:

- $e \stackrel{1}{\Longrightarrow} e'$ *e reduces to e′ in one step*
- $e \stackrel{k}{\Longrightarrow} e'$ *e reduces to e′ in k steps*
- $e \Longrightarrow e'$ *e reduces to e′ in 0 or more steps*
- $e \hookrightarrow v$ *e evaluates to v*

Examples:

$$(3+4)*2$$

$$\overset{1}{\Longrightarrow} \quad 7 * 2$$

$$\overset{1}{\Longrightarrow} \quad 14$$

$$(3+4)*(2+1)$$

$$\overset{3}{\Longrightarrow} \quad 21$$

Notation Recap

$e : t$ "e has type t"

$e \Rightarrow e'$ "e reduces to e'"

$e \hookrightarrow v$ "e evaluates to v"

Types in ML

Basic types:

int, real, bool, char, string

Constructed types:

product types
function types
user-defined types

Types `int`

Values ..., `~1, 0, 1`, ...,

that is, $\overline{n}$ for every integer n.

Expressions e_1 `+` e_2, e_1 `-` e_2, e_1 `*` e_2,

e_1 `div` e_2, e_1 `mod` e_2, *etc.*

Example: ~4 * 3

Typing Rules

- $\overline{n}$: `int`
- e_1 `+` e_2 : `int`

  if e_1 : `int` and e_2 : `int`

similar for other operations.

Example:

(3 + 4) * 2 : int

Why?

3+4 : int and 2 : int

Why?

3 : int and 4 : int

Evaluation Rules

- $e_1 + e_2 \stackrel{1}{\Longrightarrow} e_1' + e_2$ if $e_1 \stackrel{1}{\Longrightarrow} e_1'$
- $\overline{n_1} + e_2 \stackrel{1}{\Longrightarrow} \overline{n_1} + e_2'$ if $e_2 \stackrel{1}{\Longrightarrow} e_2'$
- $\overline{n_1} + \overline{n_2} \stackrel{1}{\Longrightarrow} \overline{n_1 + n_2}$

Types `bool`

Values `true` and `false`

Expressions `if` e_1 `then` e_2 `else` e_3, $e_1 > e_2$, ...

Typing Rules

$$\frac{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : t}{\text{if } e_1 : \texttt{bool} \quad \text{and } e_2 : t \quad \text{and } e_3 : t}$$

if e_1 : `bool`

and $e_2 : t$

and $e_3 : t$

Example: if (3>4) then "foo" else "bar" : string

Evaluation Rules

- $$\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \stackrel{1}{\Longrightarrow} \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3$$

  if $e_1 \stackrel{1}{\Longrightarrow} e_1'$

- $\texttt{if true then } e_2 \texttt{ else } e_3 \stackrel{1}{\Longrightarrow} e_2$

- $\texttt{if false then } e_2 \texttt{ else } e_3 \stackrel{1}{\Longrightarrow} e_3$

Types $t_1 * t_2$ 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

usually bad style

Examples:

(3+4, true)

(1.0, ~15.6)

(8, 5, false, ~2)

Typing Rules

- $(e_1,\ e_2)\ :\ t_1 * t_2$

  if $e_1 : t_1$

  and $e_2 : t_2$

- #1 $e\ :\ t_1$

  if $e : t_1 * t_2$ for some t_2.

- #2 $e\ :\ t_2$

  if $e : t_1 * t_2$ for some t_1.

Example: (3+4 , true) : int * bool

Evaluation Rules

- $(e_1, e_2) \stackrel{1}{\Longrightarrow} (e_1', e_2)$ if $e_1 \stackrel{1}{\Longrightarrow} e_1'$
- $(v_1, e_2) \stackrel{1}{\Longrightarrow} (v_1, e_2')$ if $e_2 \stackrel{1}{\Longrightarrow} e_2'$
- $\#1\ e \stackrel{1}{\Longrightarrow} \#1\ e'$ if $e \stackrel{1}{\Longrightarrow} e'$
- $\#1\ (v_1, v_2) \stackrel{1}{\Longrightarrow} v_1$
- $\#2\ e \stackrel{1}{\Longrightarrow} \#2\ e'$ if $e \stackrel{1}{\Longrightarrow} e'$
- $\#2\ (v_1, v_2) \stackrel{1}{\Longrightarrow} v_2$

Declarations

Environments

Scope

Concrete Type Definitions

Next time ...

Functions