15-150

Principles of Functional Programming

Michael Erdmann

Brandon Bohrer, Kevin Burg, Sandra Chen,

Sanaa Garg, Felipe Gomez-Frittelli, Zack Greenberg,

Bill Maynes, Rob Murcek,

Rebecca Paren, Jasmine Peterson, Sri Raghavan,

Nikki Ray, Luna Ruan,

Corey Sobel, Sannidhi Srinivasan,

Akshay Nanavati, Todd Nowacki, Lars Wander

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
$\downarrow$		$\downarrow$
• executed		• evaluated
• has an effect		• 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

Parallelism

/\

< 1, 0, 0, 1, 1 > → 3,

< 1, 0, 1, 1, 0 > → 3,

< 1, 1, 1, 0, 1 > → 4,

< 0, 1, 1, 0, 0 > → 2,

\/

↓

12

Parallelism

sum : int sequence $\rightarrow$ int

type row = int sequence

type room = row sequence

```
fun count (class : room) : int =
        sum (map sum class)
```

Parallelism

- Work:
  - Sequential Computation
  - Total sequential time;
    number of operations

- Span:
  - Parallel Computation
  - How long would it take if one could have as many processors as one wants;
    length of longest critical path

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*

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:

- $e \xRightarrow{1} e'$ *e reduces to* e' *in one step*
- $e \xRightarrow{k} e'$ *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:

$$
\begin{aligned}
& (3+4)*2 \\
\overset{1}{\Longrightarrow}\ & 7*2 \\
\overset{1}{\Longrightarrow}\ & 14
\end{aligned}
$$

$$
\begin{aligned}
& (3+4)*(2+1) \\
\overset{3}{\Longrightarrow}\ & 21
\end{aligned}
$$

Notation Recap

$e : t$	"e has type t"
$e \Rightarrow e'$	"e reduces to e'"
$e \hookrightarrow v$	"e evaluates to v"

Equivalence

- Functional programs are *referentially transparent*
  - The *value* of an expression depends only on the *values* of its sub-expressions
  - The *type* of an expression depends only on the *types* of its sub-expressions
- Expressions are *extensionally equivalent if they both reduce to the same value, or both raise the same exception, or both loop forever.*
- Functions are *extensionally equivalent* if they map equivalent arguments to equivalent results.
  - *safe substitution* for equivalent code
- Examples:
  - 21 + 21 is *equivalent* to 42
  - [2,7,6] is *equivalent* to [1+1, 2+5, 3+3]
  - (fn x => x+x) is *equivalent* to (fn y => 2*y)
- In proofs, will use $\cong$ for “equivalent”, e.g., [2, 7] $\cong$ [1+1, 2+5].

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

- $e_1 + e_2 \stackrel{1}{\Longrightarrow} e_1' + e_2 \quad \text{if } e_1 \stackrel{1}{\Longrightarrow} e_1'$

- $\overline{n_1} + e_2 \stackrel{1}{\Longrightarrow} \overline{n_1} + e_2' \quad \text{if } e_2 \stackrel{1}{\Longrightarrow} e_2'$

- $\overline{n_1} + \overline{n_2} \stackrel{1}{\Longrightarrow} \overline{n_1 + n_2}$

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), $\underbrace{\#1\ e, \quad \#2\ e}_{\text{usually bad style}}$

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

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

SML/NJ

- From Andrew:

  `/usr/local/bin/sml`

- Personal copies available at:

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

(Further details underneath the course webpage.)

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;
val it = 8 : int

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

-
```

← keyboard

← keyboard (load file)

Next time ...

Functions

Course Tasks

• Assignments	40%
• Labs	10%
• Quizzes	5%
• Midterm	18%
• Final	27%

Roughly one assignment per week, one lab per week.

Quizzes will occur at irregular and unannounced times.
10-15 minutes, during lecture.
Roughly half a dozen. You may drop lowest score.