LECTURE 1

Introduction, Equivalence, Evaluation, Typing, Binding, Scope
Course staff

Instructors:  Michael Erdmann  Dilsun Kaynar

Head TA:  Jacob Neumann

Pankaj Bhojvani  Ariel Davis  Joy Gu  Malaika Handa
Leo Huang  Maxwell Johnson  Thejas Kadur  Minji Lee  Adam Lerner
Jeremy Leung  Wan Shen Lim  Crystal Lin  Tristan Marino
Shashank Ojha  Pavi Pandurangan  Cliff Ressel
Kylee Santos  Brian Scheuermann
Travis Schwartz  Mera Tegene  Elliot Toy  Michael Wagner
Jordan Widjaja  Carter Williams  Cameron Wong
Grace Yu  Erin Zhang
Course philosophy

• **Computation** is functional.

• **Programming** is an explanatory linguistic process.
Computation is functional

- values classified with respect to types
- expressions
- functions map values to values
Imperative vs Functional

command

-executed

has an effect

x := 5

(new state)

expression

evaluated

no effect

3 + 5

(new value)
Programming as explanation

• Problem statement
• Invariants
• Specifications
• Proofs of correctness
• Analyze, decompose and fit, prove

High expectation to explain precisely and concisely
Parallelism

\[ \langle 1, 0, 0, 1, 1 \rangle \rightarrow 3 \]
\[ \langle 1, 0, 1, 0, 1 \rangle \rightarrow 3 \]
\[ \langle 1, 1, 0, 1, 1 \rangle \rightarrow 4 \]
\[ \langle 0, 0, 0, 1, 1 \rangle \rightarrow 2 \]
\[ <0, 0, 0, 1, 1> \rightarrow 12 \]
\[\text{fun} \ \text{count} \ (\text{class} : \text{room}) : \text{int} = \text{sum} \ (\text{map} \ \text{sum} \ \text{class})\]
Complexity analysis

• Let $k$ be the number of rows, and $n/k$ be the size of each row.

• *If we have enough parallel processors,* the `count` operation takes time proportional to $O(k + n/k)$.

  computes each row sum, in parallel, then adds the row sums.

With $n=240$, $k=12$, $k + n/k$ is 32, almost 8-fold speedup over 240.
Work and Span

- **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
Analysis

• How could you improve the running time of \texttt{count}?

Divide and conquer
Defining ML

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

- A type is a “prediction” about the kind of value that an expression will have if it winds up having a value.
- An expression is **well-typed** if it has at least one type, and **ill-typed** otherwise.
- A well-typed expression has a type, may have a value, and may have an effect (not for our effect-free fragment).
Example

Well-typed expression with no value

5 div 0 : int
Extensional equivalence

$\equiv$

An equivalence relation on expressions of the same type
Equivalence

- Expressions of type `int` are equivalent whenever one of the following is true
  - if they evaluate to the same integer
  - if they both loop forever
  - if they both raise the same exception

Equivalence is a form of semantic equality
Equivalence

• Functions of type \texttt{int -> int} are equivalent if they map equivalent arguments to equivalent results

Equivalence is a form of semantic equality
Referential transparency

for types and values

- The type of an expression depends only on the types of its sub-expressions
- The value of an expression depends only on the values of its sub-expressions

safe substitution, compositional reasoning
Equivalence

• $21 + 21$ is equivalent to $42$ is equivalent to $7 * 6$
• $[2, 4, 6]$ is equivalent to $[1+1, 2+2, 3+3]$

We use $\equiv$ for equivalence
Functions

In math, one talks about a function $f$ being a mapping between spaces $X$ and $Y$.

$$f : X \rightarrow Y$$

In SML, we do the same with $X$ and $Y$ being types.
Totality

- Issue: Computationally a function may not always return a value. This complicates equivalence checking.

**Definition:** A function $f : X -> Y$ is *total* if $f(x)$ returns a value for all values $x$ in $X$. 
Types in ML

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

• Constructed types
  • Product types
  • Function types
  • User-defined types
Integers, Expressions

- **Type** `int`
- **Values** ..., ~1, 0, 1, ..., 
- **Expressions** $e_1 + e_2$, $e_1 - e_2$, $e_1 * e_2$, $e_1 \div e_2$, $e_1 \mod e_2$, ...
- **Example** $-4 * 3$
Integers, Typing

• Typing rules
  • \( n : \text{int} \)
  • \( e_1 + e_2 : \text{int} \) if \( e_1 : \text{int} \) and \( e_2 : \text{int} \)
  • similar for other operations
Integers, Evaluation

- $e_1 + e_2$ evaluates to $e_1' + e_2$ in one step if $e_1$ evaluates to $e_1'$ in one step
- $n_1 + e_2$ evaluates to $n_1 + e_2'$ in one step if $e_2$ evaluates to $e_2'$ in one step
- $n_1 + n_2$ evaluates to $n_1 + n_2$ in one step
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 \((e_1, e_2), \# e_1, \# e_2\)
• Example \((\sim 4 \times 3, \text{true}) \)
  \((3, 5, \text{“another example”})\)

usually bad style
Products, Typing

• Typing Rules:

• \((e_1, e_2) : t_1 \times t_2 \text{ if } e_1 : t_1 \text{ and } e_2 : t_2\)

• Example

\((~4 * 3, \text{true}) : \text{int} \times \text{bool}\)

\( (3,5, \text{“another example”}) : \text{int} \times \text{int} \times \text{string} \)
Products, Evaluation

• \((e_1, e_2)\) evaluates to \((e_1', e_2)\) in one step if \(e_1\) evaluates to \(e_1'\) in one step

• \((v_1, e_2)\) evaluates to \(v_1 + e_2'\) in one step if \(e_2\) evaluates to \(e_2'\) in one step

• \((v_1, v_2)\) evaluates to \((v_1, v_2)\) in one step
Declarations, Environments, Scope
Declaration

\texttt{val pi : real = 3.14}

Introduces binding of 3.14 to \texttt{pi}, sometimes written as \texttt{[3.14/x]}

Lexically statically scoped
Environment

\[
\begin{align*}
\text{val } x &: \text{ int } = 8 - 5 & [3/x] \\
\text{val } y &: \text{ int } = x + 1 & [4/y] \\
\text{val } x &: \text{ int } = 10 & [10/x] \\
\text{val } z &: \text{ int } = x + 1 & [11/z]
\end{align*}
\]

Second binding of \( x \) shadows first binding. First binding has been shadowed.
Local declarations

let ... in ... end

let

val m : int = 3
val n : int = m * m

end

This is an expression with type int and value 12.
Local declarations

```plaintext
val k : int = 4

let
  val k : real = 3.0
in
  k * k
end
```

Type? Value?
Local declarations

val k : int = 4

let
  val k : real = 3.0
in
  k * k
end
Concrete Type Definitions

type float = real
type point = float * float
val p : point = (1.0, 2.6)
Course tasks

• Assignments 30%
• Labs 5%
• Midterm 1 20% (Feb 15)
• Midterm 2 20% (Mar 29)
• Final 25%
Collaboration policy

• Make sure to read and understand the policy for this semester
Functions
fun square (x : int) : int = x * x
How does ML evaluate a function application $e_2 e_1$?

- Evaluate $e_2$ to a function value $\bar{f}$
- Reduce $e_1$ to a value $\nu$
- Locally extend the environment that existed at the time of the definition of $\bar{f}$ with a binding of value $\nu$ to the variable $x$
- Evaluate the body in the resulting environment