LECTURE 2
Types, expressions and declarations
Make a plan

- **Class, Labs** (remote)
- **Study** lecture material
- **Homework**
  - Start as early as possible, end on time
  - Don’t cheat — ask us if you need advice
- **Office hours** (remote)
Today

• Types, expressions and values
• Declarations, binding and scope
• Introduction to ML syntax
• Some example programs
Types

\[ t ::= \text{int} \mid \text{real} \mid \text{bool} \]
\[ \mid t_1 \ast t_2 \ast \cdots \ast t_k \]
\[ \mid t_1 \rightarrow t_2 \]
\[ \mid t_1 \text{ list} \]

integers, reals, truth values
tuples
functions
lists

There are syntax rules for well-typed expressions

Only well-typed expressions can be evaluated
Expressions

\[ e ::= x \mid n \mid e_1 + e_2 \mid \text{true} \mid \text{false} \mid e_1 \text{ andalso } e_2 \mid \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \mid (e_1, \ldots, e_k) \mid \text{fn } (x:t_1): t_2 \Rightarrow e_2 \mid e_1 \ e_2 \]

variables
numerals
arithmetic ops
truth values
logical ops
conditional
tuples
functions
application

+ lists, reals, …
+ declarations
list expressions

\[ e ::= \text{nil} \quad | \quad e_1 :: e_2 \quad | \quad e_1 \ @\ e_2 \quad | \quad [e_1, \ldots, e_k] \]
declarations

\[
\text{d ::= } \begin{align*}
& \text{val } x = e \\
& \text{fun } f(x:t_1):t_2 = e \\
& d_1; d_2 \\
& d_1 \text{ and } d_2
\end{align*}
\]

\[
\text{e ::= } \begin{align*}
& \text{let } d \text{ in } e_1 \text{ end} \\
& \text{local } d_1 \text{ in } d_2 \text{ end}
\end{align*}
\]

val
recursive function
sequential
simultaneous
scoped use
Values

• For each type $t$ there is a set of (syntactic) values

• An expression of type $t$ evaluates to a value of type $t$ (or fails to terminate)
<table>
<thead>
<tr>
<th>TYPE</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer numerals 42, ~42</td>
</tr>
<tr>
<td>real</td>
<td>real numbers 4.2, ~4.2</td>
</tr>
<tr>
<td>bool</td>
<td>truth values true, false</td>
</tr>
<tr>
<td>$t_1 \to t_2$</td>
<td>functions from $t_1$ to $t_2$</td>
</tr>
<tr>
<td>$t_1 \times \ldots \times t_k$</td>
<td>tuples of values of type $t_1$ ... $t_k$</td>
</tr>
<tr>
<td>$t_1$ list</td>
<td>lists of values of type $t_1$</td>
</tr>
</tbody>
</table>
Functions are values

A function value of type \( t_1 \rightarrow t_2 \)

is a syntactic form

\[
\text{fn} \ (x : t_1) : t_2 \Rightarrow e
\]

where, if \( x \) has type \( t_1 \), \( e \) has type \( t_2 \)

A function value of type \( t_1 \rightarrow t_2 \)

denotes

a partial function from values of type \( t_1 \)
to values of type \( t_2 \)
# Examples

<table>
<thead>
<tr>
<th>expression</th>
<th>value : type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(3 + 4) \times 6$</td>
<td>42 : int</td>
</tr>
<tr>
<td>$(3.0 + 4.0) \times 6.0$</td>
<td>42.0 : real</td>
</tr>
<tr>
<td>$(21+21, 2+3)$</td>
<td>$(42, 5) : \text{int} \times \text{int}$</td>
</tr>
<tr>
<td>$\text{fn } x \Rightarrow x+42$</td>
<td>$\text{fn } x \Rightarrow x+42 : \text{int} \rightarrow \text{int}$</td>
</tr>
<tr>
<td>$\text{fn } x \Rightarrow 2+2$</td>
<td>$\text{fn } x \Rightarrow 2+2 : \text{int} \rightarrow \text{int}$</td>
</tr>
</tbody>
</table>
Examples

• A function value of type \texttt{int \to int} denotes a partial function from $\mathbb{Z}$ to $\mathbb{Z}$

\begin{verbatim}
fun even(x:int):int = if x=0 then 0 else even(x-2)
\end{verbatim}

even denotes $\{(v, 0) \mid v \geq 0 \& v \mod 2 = 0\}$

even 42 evaluates to 0

even 41 loops forever
ML system

• You enter an expression
  • The system checks it’s well-typed…
  • … and evaluates, to a syntactic value.

• You enter a declaration
  • The system checks it’s well-typed…
  • … and produces bindings, of names to syntactic values.
Standard ML of New Jersey [...]  
- 225 + 193 ;  
val it = 418 : int

Don’t forget the semi-colon.  
ML reports the type and value.

225 + 193 = 418  
225 + 193 \implies 418  
\text{runtime behavior consistent with math}
Standard ML of New Jersey [...] 

- \texttt{fn} (x:int) => 2+2; 

\texttt{val it = fn - : int -> int}

ML says “it’s a function value of type int -> int”

The actual value is \texttt{fn x:int => 2+2}

The 2+2 doesn’t get evaluated (yet)

- it 99; 

\texttt{val it = 4 : int}
## Examples

<table>
<thead>
<tr>
<th>expression</th>
<th>ML says value : type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fn (x:int):int =&gt; x + 1</code></td>
<td><code>fn - : int -&gt; int</code></td>
</tr>
<tr>
<td><code>fn (x:real):real =&gt; x + 1.0</code></td>
<td><code>fn - : real -&gt; real</code></td>
</tr>
</tbody>
</table>
Declarations

**fun** double(x:int) : int = x + x

- val double = fn - : int -> int

  binds double
to the value

**fn** (x:int) : int => x + x

In the *scope* of this declaration,

**double(double 3)**

evaluates to 12
Scope

• Bindings have \textbf{static} (syntax-based) \textit{scope}

\begin{align*}
\text{val} & \quad \text{pi} : \text{real} = 3.14; \\
\text{fun} & \quad \text{area(x:real):real = pi*x*x}
\end{align*}

\begin{align*}
\text{let} & \quad \text{val} \quad \text{pi} : \text{real} = 3.14 \\
& \quad \text{in} \\
& \quad 2.0 \times \text{pi} \\
\text{end}
\end{align*}

\begin{align*}
\text{local} & \quad \text{val} \quad \text{pi} : \text{real} = 3.14 \\
& \quad \text{in} \\
& \quad \text{fun} \quad \text{area(x:real):real = pi*x*x} \\
\text{end}
\end{align*}
Design issues

\[
\text{fun circ(r:real):real} = 2.0 \times \pi \times r
\]

\[
\text{fun circ(r:real):real} = \\
\quad \text{let} \\
\quad \quad \text{val pi2:real} = 2.0 \times \pi \\
\quad \text{in} \\
\quad \quad \pi2 \times r \\
\quad \text{end}
\]

\[
\text{local} \\
\quad \text{val pi2:real} = 2.0 \times \pi \\
\quad \text{in} \\
\quad \text{fun circ(r:real):real} = \pi2 \times r \\
\quad \text{end}
\]

2.0\pi only gets evaluated once
Summary

• An expression of type $t$ can be evaluated
• If it terminates, we get a value of type $t$
• ML reports the type and value
  • val it = 3 : int
  • val it = fn - : int -> int
• Declarations produce bindings
• Bindings are statically scoped

Use well scoped declarations to avoid re-evaluating code repeatedly
List expressions

\[
e ::= \text{nil} \mid e_1::e_2 \mid [e_1, \ldots, e_k] \mid e_1@e_2
\]

All items in a list must have the same type

- \text{nil} has type \text{t list}
- \text{e}_1::\text{e}_2 \text{ has type } \text{t list} if \text{e}_1 : \text{t} and \text{e}_2 : \text{t list}
- \text{[e}_1, \ldots, \text{e}_k\text{]} \text{ has type } \text{t list} if each \text{e}_i \text{ has type } \text{t}
- \text{e}_1@\text{e}_2 \text{ has type } \text{t list} if \text{e}_1 \text{ and } \text{e}_2 \text{ have type } \text{t list}
Examples

- \([1, 3, 2, 1, 21+21]\) : int list
- \([\text{true}, \text{false}, \text{true}]\) : bool list
- \([[[1]], [2, 3]]\) : (int list) list
- \([\ ], \[\ ]\) : int list, bool list, ......
- \(1::[2, 3], 1::(2::[3]), 1::2::[3], 1::2::3::\text{nil}\)
- \([1, 2]@[3, 4]\)
- \(\text{nil} = [\ ]\)
Examples

• To finish, some ML functions to solve a simple problem.
• Introduces ML syntax (it’s fun!)
• Don’t worry if you aren’t familiar with ML.
• The examples are easy to follow (we hope).
Math background

• Every *non-negative* integer $n$ has an integer square root, the unique non-negative integer $m$ such that $m^2 \leq n < (m+1)^2$

• The integer square root of 6 is 2

**How could we write an ML function to compute integer square roots?**

- should have type `int -> int`
- needs to work for *non-negative* arguments
Finding integer square root

isqrt_0 : int -> int

fun isqrt_0 (n : int) : int =
  let
    fun loop (i : int) : int =
      if n < i*i then i-1 else loop (i+1)
  in
    loop 1
  end

• isqrt_0 n uses a localized recursive function
  loop : int -> int

• loop 1 finds smallest positive integer i such that n < i^2

• returns the value of i-1
Finding integer square root

\textbf{isqrt\_1} : \texttt{int -> int}

- isqrt\_1 is a recursive function
- For \( n > 0 \), isqrt\_1 \( n \) calls isqrt\_1(\( n-1 \))
- Uses a \texttt{let}-binding to avoid recalculation (\( r \) is used multiple times)
- Relies on arithmetic facts

\begin{verbatim}
fun isqrt_1 (n:int) : int = 
  if n=0 then 0 else
  let
    val r = isqrt_1 (n-1) + 1
  in
    if n < r * r then r - 1 else r
  end
\end{verbatim}
LEMMA
If $n > 0$ and $k$ is the integer square root of $n-1$, then either $k$ or $k+1$ is the integer square root of $n$.

Proof? Do the math!

Can show that

$k$ is the square root of $n$, if $n < (k+1)^2$

and $k+1$ is the square root of $n$, if $n \geq (k+1)^2$

This is why we wrote the code!
Finding integer square root

isqrt_2 : int -> int

• A recursive function definition
  
  • For \( n > 0 \), \( \text{isqrt}_2 \ n \) calls \( \text{isqrt}_2 \ (n \div 4) \)

• Relies on (different) arithmetic facts

…which facts?
Results

• All three functions compute integer square root correctly
• Try them out on larger and larger integer arguments….
• Can you see any differences?
• Why?
Let’s try it

Start up the ML runtime system.
Enter the function definitions for

\texttt{isqrt\_0},
\texttt{isqrt\_1},
\texttt{isqrt\_2},
as given above.

1. Find the value of \texttt{isqrt\_0 2020}

2. What happens when you evaluate \texttt{isqrt\_1 123456789}?

3. What happens when you evaluate \texttt{isqrt\_2 123456789}?
Questions

• Are the functions `isqrt_0`, `isqrt_1` and `isqrt_2` equivalent?

• If so, how could you prove it?

• If not, how could you show it?
covid testing

- Population size $N$
- Tests assumed accurate
- Naive testing algorithm: take a sample from each person and test it
  - needs a total of $N$ tests

We can do better... with fewer tests!

The Detection of Defective Members of Large Populations
Robert Dorfman, Annals of Math Stats, 1947
Let $p$ be probability that a test is positive

Split population of $N$ into groups of size $n$

Test the *grouped samples*

  • prob that a group test is negative is $(1-p)^n$

For each *positive* group, test its members

The total *expected* number of tests is

$$(N \div n) + P \times (N \div n) \times n,$$

where $P$ is $1-(1-p)^n$

if $n$ divides $N$, simplifies to

$$(N \div n) + \lceil P \rceil \times N$$

(a smarter algorithm?)
fun \( \text{exp}(r:\text{real}, n:\text{int}) : \text{real} = \)
  \text{if } n=0 \text{ then } 1.0 \text{ else } r \times \text{exp}(r, n-1) \\

fun \text{cost}(N:\text{int}, n:\text{int}, p:\text{real}) : \text{int} =
  \text{let}
  \quad \text{val } P : \text{real} = 1.0 - \text{exp}(1.0 - p, n)
  \text{in}
  \quad (N \text{ div } n) + \text{ceil}((\text{real } N) \times P)
  \text{end} \\
- \text{cost}(150, 10, 0.01); \quad \text{val } \text{it} = 30 : \text{int} \\

150 \text{ people, when } p = 1\%, \text{ can be assessed with just 30 tests}
TBD

• Given N and p, what’s the optimal n?
• the cheapest method