Recitation - 07

Jahnavi - jp5867@nyu.edu

ML features:

- Functional Language
- Automatic garbage collection
- Applicative order evaluation - function's arguments are evaluated completely before the function is applied
- Polymorphism
- Pattern Matching
- Abstract datatypes
- Strongly Typed
- Type inference
- Advanced module system

Basic Expressions:

```
(* Integers *)
1
~3 (*Negative 3*)

(* Reals *)
3.14
~3.2E3

(* Booleans *)
true
false

(* Strings *)
" This is a string "
```

1. The operator +, - can be between reals or integers but not both.
2. Division / applies only to reals.

```
3 + 4
3 + 4.0 (* WRONG ✘! SML does not support convertion *)
real 3 + 4.0 (* or 3.0 + 4.0 *)
3 <> 4 (* not equal, val it = true : bool *)
```

Control Flow Expressions:

If else statement:

```
(* Format:
        if <cond> then <exp> else <exp>
*)

if true then 3 else false (* WRONG ✘! types of then branch and else branch should be the same *)
```

Multiple selection:

```
(* Format:
       case <exp1> of
       <pattern_1> => <exp>
       | <pattern_2> => <exp>
       | . . . . . .
       | <pattern_n> => <exp>
*)
fun compare (x, y) =
       if x < y then ~1
       else if x > y then 1
       else 0;

case compare(3, 4) of
        1 => "Greater"
       | ~1 => "Less"
       | 0 => "Equal"
       | _ => "Error input!" ;
```

Type Inference:

- ML has a strong type system, but types don't need to be declared by the programmer
- Interpreter uses a type inference system to deduce the types of functions and variables
- Functions must always return the same type:

❖ If-then-else:

- The expression following if must have a Boolean type.
- The expression following 'then' and 'else' can be of any one type but they must be of the same type.

Type Inference:

- Sometimes it is just easy to see: f a = a + 1 . This function has a type of int -> int
- However, what about the function that returns its arguments? => f g=g
- To solve this problem, ml uses a type variable. They are denoted by a quotation mark (') followed by a letter
- Usually,the first letter in the alphabet not being used by another type variable

EX: 'a

- Denotes that the value is"of type 'a"

Going back to our example, the type of "f g = g" is:

- 'a -> 'a
- Takes in a parameter of type 'a and returns a value of type 'a

Type error or not?

```
fun bigger a b = if a then b - a else a - b

fun compose f g = fn x => f (g x)

fun reverse l = case l of
    [] => []
  | (h::t) => (reverse t) :: h
```

Type error or not?

1. bigger : Type error on variable a. The condition expression expects variable a has boolean type. However, a - b refers that a should be of int.
2. Compose : No error. **val compose = fn : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b**
3. reverse: Error on second match case (reverse t) :: h

   In the first matching case [] => [], we can infer that the reverse function returns 'a list as return. But, in the second case, reverse t will result in a list and appending h will create list of lists which will cause type result circularity.

Tuples:

- Collection of 2 or more expressions of any types (heterogeneous)
- Example: val t=(4, 5.0,"hello world")
- Tuples are used for function arguments when there is more than 1 parameter.
- The type of a tuple is the product type using (*).
- A product type is formed from two or more types T1, T2, ... Tn by putting * between them like T1 * T2 * ..* Tn.
- Example: (1, 3.0, "Hi") is of type int * real * string
- Values of this type are tuples with k components, the first if which is of type T1, second is of type T2 and so on.

Data Types:

- Used for defining new types.
- Type is defined as the set of values and operators that operate on those values.
- Data type definition involves two kinds of identifiers:
- An identifier which is the name of the data type.
- One or more data constructors used as operators to build values belonging to new data type.

Examples:

- datatype fruit=Apple | Pear | Grape;
- datatype ('a, 'b) element = pair of 'a * 'b | Null

Recursive data types:Type whose values may incorporate elements of that type.

- datatype 'a list = Nil | Cons of 'a * (list 'a)
- val myList = Cons(5, Cons (2 , Cons (3 , Nil)));

Example:

- Parse tree for simple arithmetic statements:
- datatype op = PLUS | MINUS | DIVIDE | MULT
- datatype leaf = int
- datatype (leaf, op) tree = leaf | Node of tree * op * tree

Example:

- Node(Node(2,PLUS,5),MULT,Node(10,DIVIDE,2)) => (2+5) * (10/2)
- (2+(5*10))/2 => Node (Node(2, PLUS, Node (5, MULT, 10)), DIVIDE, 2)

Lists:

- Elements are of the same type (homogenous). Comma separated, surrounded by square brackets.
- Example:

- [1,2,3]
- val it=[1,2,3] : int list
- []; => valit=[]:'a list

- Head and tail of the list: Except for the empty list, every list has a head (First element of the list) and tail (List of all elements but the first). You access them using the functions 'hd' and 'tl'. Similar to car and cdr in Scheme.

- hd [1,2,3] -> 1
- tl [1,2,3] -> [2,3]

Lists:

- Concatenation (@): Two lists can be concatenated using @ operator.
  - [1,2] @ [3,4] -> [1,2,3,4]
- Cons: The cons operator in ML is represented as (::), takes an element (head) and a list of elements of the same type as the head and produces a single list.
  - 2::[3,4] -> [2,3,4]
- Empty list: The type of an empty list is 'a list

How ML deduces types:

1. Types of operands and result of arithmetic operations must all agree.

2. Whenarithmeticcomparisonismade,youcanbesurethatthe operands are of the same type and the result is Boolean.

3. In a conditional expression, the sub-expressions following the then and else must be of the same type.

4. If a variable or expression used as an argument of a function is of a known type, then the corresponding parameter of the function must be of that type.

5. Return type of same function must be consistent (e.g.in pattern matching, or if the same function is called multiple times.)

6. If there is no way to determine the type of a particular use of an overloaded operator exists, then the type of that operator is defined to be the default for that operator, normally integer.

    a. An operator is overloaded if it can apply to 2 or more different types. For example +.

7. Types that cannot be specified are given type variables.

Polymorphism:

While ML is strictly typed, it is possible to write functions that are applicable to more than a single type. The types of these parameters are assigned type variables by the type-checker.

- Ex. - fun nonNegative f a = if f a > 0 then f a else 0;

val nonNegative = fn : ('a -> int) -> 'a -> int

- This is an example of polymorphism, the use of a single operation with multiple types.

Datatypes can also be polymorphic.

- The above function will work correctly with different types of inputs:

nonNegative length [0, 1, 2];

val it = 3 : int

nonNegative abs (-4);

val it = 4 : int

Pattern Matching:

- ML makes heavy use of pattern matching to deal with different cases of inputs.
- Constants will only match their value, while variables will match any value and bind a name to it.
- Pattern matching is the standard way to write recursive functions
- Example:

```
fun fib 0 = 1
        | fib 1=1
        | fib n = fib (n - 1) + fib (n - 2);
val fib = fn : int -> int
```

- This is especially common when dealing with lists, records, datatypes

Pattern Matching:

- This is especially common when dealing with lists, records, datatypes.
  - Example using the tree we defined earlier:

```
fun get_leaves Leaf i = [i]
        | get_leaves Nodes (left, op, right) = (get_leaves left) @ (get_leaves right)
val get_leaves = fn : tree -> leaf list
```

- The '_' pattern is a wildcard. It matches any value but doesn't bind a name
  - Example: fun summap _ [] = 0

```
                | summap f (x::xs) = f x + summap f xs;
val summap = fn: ('a->int) -> 'a list -> int
```

Other notes about Pattern Matching:

- When run, a value will match the first pattern it can match with, and ignore the rest.
- ML will warn you if your patterns are non-exhaustive.

Type Inference examples:

```
fun is_large x =
    if x > 37 then true
    else false
```

val is_large = fn : int -> bool

```
fun append l1 l2 =
  case l1 of
    [ ] => l2
  | h1::t1 => h1 :: append t1 l2
```

val append = fn : 'a list -> 'a list -> 'a list

In ML, only one argument can be passed to a function. If you want to pass multiple arguments, use a tuple of those arguments. The example above uses currying.

Examples:

3. fun foo (op <) f g (x,y) z = if f(x,y) < g x then z + 1 else z - 1

Type: ('a * 'b -> bool) -> ('c * 'd -> 'a) -> ('c -> 'b) -> 'c * 'd -> int -> int

4. fun f (x,y) [] = []

   | f (x,y) (z::zs) = if length y = 1 then y else z

Type: 'a * 'b list -> 'b list list -> 'b list

5. fun bar w x (y,z) =
   let val (b::bs) = w y
       val d = x z
       fun f a b = length a + b
   in f b d
   end

('a -> 'b list list) -> ('c -> int) -> 'a * 'c -> int

Example 5: How ML interpreter does type inference:

1. y is unconstrained, so call its type 'a. Therefore, the input type of w is 'a.
2. The type of z is unconstrained, so call its type 'c. Therefore, the input type of x is 'c.
3. The first parameter a to f can be a list of any type, so call its type 'b list. The second parameter b to f must be int, due to the use of +.
4. Thus, the type of f is 'b list -> int -> int.
5. The return type of f is int, therefore the return type of bar is int.
6. Since b and d are passed as the parameters to f, the type of b is 'b list and the type of d is int.
7. The output type of w is 'b list list, since the result of the call to w is (b::bs).
8. The output type of x is int, because d is an int.
9. Since the parameters to bar are w, x, and (y,z) and the result type is int,
10. the type of bar is ('a -> 'b list list) -> ('c -> int) -> 'a * 'c -> int.

SML installation guide:

Installation

- You can use this website to download Standard ML.
- Please follow the instructions to download SML.
- Once installed, type sml in your terminal to use SML compiler.
- You can also write a file of your assignment by using .sml extension. After named SML source file, open the compiler and type this:

– use "<file_name>.sml";

Mid-term logistics:

→ Format of examination:

- ❑ Single and more than one correct MCQs.
- ❑ Matching questions
- ❑ Short answer questions
- ❑ Fill in the blanks
- ❑ Short essay type questions