Assignment 1

August 15, 2006

1 A Brief Tutorial

At the `sunfire` command prompt, type `sml`. Standard ML of New Jersey toplevel prints out some information and waits for input at the `-` prompt. Inputs can now be entered for evaluation. `it` can be used to retrieve the result of the last expression evaluated at the `-` prompt.

```
nguyenh2@sf3:~/ta[507]$ sml
Standard ML of New Jersey, Version 0.93, February 15, 1993
val it = () : unit
- 2;
val it = 2 : int
- it+2;
val it = 4 : int
- it+3;
val it = 7 : int
-
```

If you store your program in a file named, for example, "samples.ml", the file can be loaded to SML toplevel by using `use ''samples.ml''`. The file will be loaded and the expressions contained in the file evaluated.

If you like to install your own system, you can download Standard ML of New Jersey from `http://www.smlnj.org/`.

Sample Functions

Below is a function that computes the square of an integer. The function is then invoked with 10 and 20 as its arguments.

```
- fun square (x : int) = x * x;
val square = fn : int -> int
- square 10;
val it = 100 : int
- square 20;
val it = 400 : int
-
```

Let's try a slightly more elaborated example. If the argument is a negative number, an exception will be raised. The exception has been declared with

`exception Neg`. Otherwise the square root of the arguments and its negation will be returned as a pair. The call `mysqrt 16.0` returns pair `(4.0, 4.0)`. The call `mysqrt 1.0` raises an exception. Note that passing `1` as argument will cause an type error.

```
- exception Neg;
exception Neg
- fun mysqrt(x : real) =
  if x < 0.0 then raise Neg
  else
    let
      val tmp = sqrt x
    in
      (tmp, ~tmp)
    end;
val mysqrt = fn : real -> real * real
- mysqrt 16.0;
val it = (4.0,~4.0) : real * real
- mysqrt ~1.0;

uncaught exception Neg
- mysqrt 1;
std_in:13.1-13.8 Error: operator and operand don't agree (tycon mismatch)
  operator domain: real
  operand:         int
  in expression:
    mysqrt 1
-
```

The following function computes the sum of a list of integers. The function uses pattern-matching to distinguish between an empty list and a non-empty one. If the list is empty, 0 is returned as result. Otherwise it adds the list's head element to the sum of the tail.

```
- fun sum nil = 0
    | sum  (h :: t) = h + (sum t);
val sum = fn : int list -> int
- sum [1,2,3];
val it = 6 : int
- sum [];
val it = 0 : int
-
```

2 Questions

1. Write the factorial function
2. Write function `map2 : ('a -> 'b -> 'c) -> 'a list -> 'b list -> 'c list`. `map2` takes a function `f` that accepts two arguments and two lists `a` and `b`. It then applies the function `f` to elements from `a` and `b` and

produce a list of results. Return an error if the two lists are not of the same length.
For example `map2 (fn a => fn b => a+b) [1,2,3] [4,5,6]` returns `[5,7,9]`.

3. Propositional formula evaluator:

   Propositions are defined using:

   - boolean variables (variables whose values are true or false)
   - operators `AND &, OR |, IMPLY =>` and `NOT !`

   (a) Define data type to capture propositions

   (b) An environment maps boolean variables to boolean values (`true, false`). Write a function that takes an environment and a proposition and returns the value of the proposition under the environment.

   Hint: an environment can be implemented as an association list. Signal an error (by throwing an exception) if the environment is not complete, i.e. there are variables that are not associated with a value.

   (c) Write a function that takes a proposition and returns all the boolean variables appearing in the proposition.

   (d) Write a function that generates all possible environments for a list of variables.

   (e) Write a function that takes a proposition and decides whether it is valid (evaluates to true under all environments) or unsatisfiable (evaluates to false under all environments) or satisfiable.

   (f) Write a function that takes a proposition and simplifies it.