Theory of Programming Languages
Computer Science & Engineering
ML Project
(due on 29 May, 2019; Wednesday)

1. [20/100] (Power Set) Define a function `powerset()`, which, given a set (represented as a list), will return the set of all its subsets, using the following functions.

```
fun map f nil = nil
 | map f hd::tl = f(hd) :: map f tl;
fun cons h t = h::t;
```

2. [20/100] (Counting Coins) Consider the problem of deciding how many different ways there are of changing $1 into1, 5, 10, 25, and 50 cent coins. Suppose that we impose some order on the types of coins. Then it is clear that the following relation holds:

   Number of ways to change amount *a* using n kinds of coins =
   Number of ways to change amount *a* using all but the first kind of coin +
   Number of ways to change amount *a-d* using all *n* kinds of coins,

   where *d* is the denomination of the first kind of coin.

   This relation can be transformed into a recursive function if we specify the degenerate cases that terminate the recursion. If *a*=0, we will count this as one way to make change. If *a*<0, or n *n*=0, then there is no way to make change.

3. [20/100] (Long Integer) Implement long integers as an ML data type. Use the new datatype to write ML programs which calculate the 100'th Fibonacci number and the factorial of 40.

4. [20/100] (Permutation) Write a procedure `permutations : 'a list -> 'a list list` which provides a list of all permutations of a given list. For example,

```
- permutations [1, 2, 3];
val it = [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]] : int list list
- permutations ["a", "b", "c"];
val it = [["a", "b", "c"], ["a", "c", "b"], ["b", "a", "c"],
["b", "c", "a"], ["c", "a", "b"], ["c", "b", "a"] : string list list;
```

5. [20/100] (Tree Type; Lisp Notation) The ML datatype of (simple) tree is given by the declaration:

```
datatype 'a tree = Tree of 'a tree list | Leaf of 'a;
```

   For example,

```
val ex = Tree [Leaf "2",
               Leaf "3",
               Tree [Leaf "1",
                     Leaf "3"],
               Leaf "4",
                     Tree [Tree [Leaf "3"]],
               Leaf "7"];
```

   is a string tree.

   (a) Since this notation is hard to read it is helpful to have a function which will pretty print a string tree in Lisp notation. Write an ML procedure which does this. For example, it should handle the tree above as follows:

```
- prettyprint ex;
val it = " ( 2 3 ( 1 3 ) 4 ( ( 3 ) ) 7 ) " : string
```

   (b) Write the procedure `deepreverse : 'a tree -> 'a tree`. For example:

```
- prettyprint (deepreverse ex);
val it = " ( 7 ( ( 3 ) ) 4 ( 3 1 ) 3 2 ) " : string
```