CS 331 – Assignment 3

Due: 11:59pm Monday, March 10

Your functions for all six problems should be submitted in one file named *assign3.sml* to the dropbox for Assignment 3 on D2L. Here is precisely how we will load your file into SML on the Linux network.

```
$ sml
- use "assign3.sml";
[opening assign3.sml]
                                   ... Assorted information regarding the types of your functions will appear ...
val it = () : unit
-                                                                                  We will then load our test cases
```

When we do this, your file must not produce any syntax errors, or you will receive a "zero" on this assignment. All of your functions must be named *exactly* as specified in the problem statements – otherwise points will be deducted. Obviously you can choose names for helper functions as you wish. Helper functions that aren't solutions to other problems should be nested inside the functions they help. If you have code that does produce syntax errors, comment it out so we can read it without having it interfere with loading your file.

1. In this problem, you will work with the following data type that we discussed in class:

```
datatype 'ingredient pizza =
    Bottom
    | Topping of ('ingredient * ('ingredient pizza));
```

In class we wrote a particular function called *rem_anchovy* that would remove anchovy toppings from a fish pizza. Recall sample usage of that function was:

```
- Control.Print.printDepth:=60;                                                      Controls ML's printing depth
val it = () : unit
- my_pizza;
val it = Topping (Tuna,Topping (Shark,Topping (Anchovy,Bottom))) : fish pizza
- rem_anchovy my_pizza;
val it = Topping (Tuna,Topping (Shark,Bottom)) : fish pizza
```

Now write a more general function

```
- rem_ingredient;
val it = fn : ''a -> ''a pizza -> ''a pizza
```

It is given two arguments – the first represents an arbitrary ingredient for a pizza and the second is the pizza. It will remove that ingredient from the pizza. When rem_ingredient is given just one argument representing an ingredient, it should return a function that will in turn remove the specified ingredient from any pizza it is given. Sample usage:

```
- rem_ingredient Tuna my_pizza;
val it = Topping (Shark,Topping (Anchovy,Bottom)) : fish pizza
- val rem_tuna = rem_ingredient Tuna;
val rem_tuna = fn : fish pizza -> fish pizza
- rem_tuna my_pizza;
val it = Topping (Shark,Topping (Anchovy,Bottom)) : fish pizza
```

Be sure that your function is not limited to only working with fish pizzas.

2. ML is a *statically strongly typed* language. Recall this means that, if you pass the wrong type of data to a function, this error is caught as a syntax error at compile-time. In a *dynamically strongly typed* language, this type checking is postponed until run time. In this problem, you will demonstrate that dynamic type checking can be simulated in a statically typed language such as ML. This will be done by introducing the following tagged data type:

```
(* The type of tagged values. *)
datatype tagged =
   Int of int
   | Real of real
   | Bool of bool
   | String of string;
```

You will then write the following functions that accept tagged data items:

```
exception RunTimeTypeError;            The lookup function we wrote in class on Feb. 24 showed how to raise an exception.

(* If dynamic_checked_add gets two Ints or two Reals, it returns
    their sum. Otherwise it raises a RunTimeTypeError *)
fun dynamic_checked_add (m:tagged, n:tagged):tagged

(* If dynamic_checked_and gets two Bools, it returns the logical
   and of their values.  Otherwise it raises a RunTimeTypeError *)
fun dynamic_checked_and (m:tagged, n:tagged):tagged

(* If dynamic_checked_concatenate get two Strings, it returns their
   concatenation.  Otherwise it raises a RunTimeTypeError *)
fun dynamic_checked_concatenate (m:tagged, n:tagged):tagged
```

Sample usage of these functions is as follows:

```
- dynamic_checked_add((Int 5), (Int 7));
val it = Int 12 : tagged
- dynamic_checked_add((Real 5.2), (Real 4.7));
val it = Real 9.9 : tagged
- dynamic_checked_add((String "foo"), (String "bar"));
uncaught exception RunTimeTypeError
  raised at: stdIn:411.25-411.41
- dynamic_checked_and((Bool (2<3)), (Bool (3.4<8.9)));
val it = Bool true : tagged
- dynamic_checked_and((Bool (3<2)), (Bool (3.4<8.9)));
val it = Bool false : tagged
- dynamic_checked_and((String "foo"), (Bool (3.4<8.9)));
uncaught exception RunTimeTypeError
  raised at: stdIn:433.25-433.41
- dynamic_checked_concatenate((String "foo"), (String "bar"));
val it = String "foobar" : tagged
- dynamic_checked_concatenate((Bool (3<2)), (Real 3.4));
uncaught exception RunTimeTypeError
  raised at: stdIn:438.25-438.41
- dynamic_checked_concatenate((String "foo"), (Bool (3.4<8.9)));
uncaught exception RunTimeTypeError
  raised at: stdIn:438.25-438.41
```

3. Write a function called `power_set_helper` (you'll see why it's called this) that takes a two-tuple consisting of an element *a* and a list *lst* of lists of elements of the same type as *a*. `power_set_helper` inserts *a* onto the front of each of the lists in the list *lst*. For example, if *a* is 1 and *lst* is `[[2,3],[4,5,6],nil]`, then `power_set_helper` returns `[[1,2,3],[1,4,5,6],[1]]`
4. The *power set* of a set S is the set of all subsets of S. For example, if S is $\{1, 2\}$, then the power set of S is $\{\phi, \{1\}, \{2\}, \{1, 2\}\}$, where ϕ is the empty set.

   Suppose that we represent sets in ML as a list of their elements. Write a function called `power_set` that takes a list representing a set as an argument and returns the list of lists that is the power set of the argument. Hence `power_set [1,2]` returns `[nil, [1], [2], [1, 2]]`. Hint: You might want to use *power_set_helper*.
5. In the language of *propositional logic*, statements are represented by *propositional variables*, which we may think of as "atomic" identifiers. More complex logical expressions can then be built from propositional variables by applying the logical operators AND, OR, NOT. We can use the following FORMULA datatype to represent logical expressions in ML.

```
datatype FORMULA =
     ATOMIC of string
     | AND of FORMULA * FORMULA
     | OR of FORMULA * FORMULA
     | NOT of FORMULA;
```

   For this problem, you need to write a function *eval(E, L)* that takes a logical expression E as a FORMULA and a list L of propositional variables that should be to have the value *true* assigned to them. Your *eval* function should then return the truth value of E on the assumption that the propositional variables are true and all other variables are false. Hence a sample usage of the *eval* function looks like:

```
- Control.Print.printDepth:=60;
val it = () : unit
- val my_formula = AND(ATOMIC("x"), OR(ATOMIC("y"),NOT(ATOMIC("y"))));
val my_formula = AND (ATOMIC "x",OR (ATOMIC "y",NOT (ATOMIC "y"))) : FORMULA
- eval(AND(ATOMIC("x"), OR(ATOMIC("y"),NOT(ATOMIC("y")))), ["x"]);
val it = true : bool
```

6. Continuation of the previous problem. Write the function *check_formula* that takes a FORMULA and returns:
   - the string "tautology" if the FORMULA is a tautology, that is, a logical formula that is always true regardless of the truth value of its constituent propositional variables
   - the string "contradiction" if the FORMULA is a contradiction, that is, a logical formula that is always false regardless of the truth value of its constituent propositional variables
   - the string "neither" if the FORMULA is neither a tautology nor a contradiction

   Sample usage:

```
- check_formula(OR(ATOMIC("y"),NOT(ATOMIC("y"))));
val it = "tautology" : string
- check_formula(AND(ATOMIC("x"),NOT(ATOMIC("x"))));
val it = "contradiction" : string
- check_formula(ATOMIC("z"));
val it = "neither" : string
```