Nick Krawczeniuk and Gianna Sorrentino

Homework on SML

To do alone or in a team of 2 students.

SML

1. What are the types of the following expressions?
   - [(1,5), (2,3), (5,6)]; **A list of tuples**
   - fun f(x:real) = true; **A real number function storing the value "true"**
   - map f; **A curried map function**

2. Provide expressions of the following types:
   - int * bool **(3, true)**
   - int list * bool **([4,5,6], true)**
   - int * real -> bool list **(3, 7.0) → [true,false]**

3. Write the following SML functions:

Write a recursive function that computes 2^n for $n \geq 0$.

fun f(x:int) = if x=0 then 1 else x*f(x-1);

```
fun fact n = if n=0 then 1
                    else n * fact(n-1);

fun new_if (a,b,c) = if a then b else c;
```

Using **new_if**, write a function **new_fact** that is supposed to compute *fact*.
Explain why **new_fact** does not compute the factorial.

Note: How are recursive functions evaluated in SML?

fun new_fact(n) = new_if(0,1, n*new_fact(n-1));

This does not work because the type of new_if is boolean and the type of new_fact is int. The types do not agree.

Define a function *circumference* (resp. a function *area*) that computes the circumference (resp. the area) of a circle with respect to its radius. Use *pi* from the Math library.

```
open Math;
```

In math, C = 2 π r.

```
open Math;
fun circ r = 2.0 * pi * r;
```

How to use map to add 3 to each elements of a list

```
fun addThree(x) = x + 3 ;
map addThree[91, 22, 8, 66];
map addThree L;
```

Write a function *move* that transforms a list $[a_1, ..., a_n]$ into a list $[a_2, ..., a_n, a_1]$.

```
L = [4,5,6]
fun move(L) = if L = nil then nil else tl(L) @ [hd(L)];
```

4. Implement the datatype BinaryTree and all the functions that are provided in the lecture notes: lookup, **inorder, preorder, postorder, left_subtree, right_subtree** and label. Provide screenshots to show that your code is correct. Provide 2 tests for each function.

Trees:

```
- val Tree2 = bt(3,btempty,
=                bt(5,btempty,
=                  bt(10,bt(8,bt(6,btempty,btempty),
=                                 btempty),
=                           bt(8,btempty,btempty))
=                )
=       );
val Tree2 = bt (3,btempty,bt (5,btempty,bt #)) : int BinaryTree
```

```
- datatype 'a BinaryTree = btempty | bt of 'a * 'a BinaryTree * 'a BinaryTree;
datatype 'a BinaryTree = bt of 'a * 'a BinaryTree * 'a BinaryTree | btempty
- val Tree = bt(5, btempty,
= bt(7,btempty,
= bt(10,bt(4,bt(1,btempty,btempty),
= btempty),
= bt(0,btempty,btempty))
= )
= );
val Tree = bt (5,btempty,bt (7,btempty,bt #)) : int BinaryTree
-
```

Lookup tests and code:

```
- lookup(btempty, 1);
val it = false : bool
```

```
- lookup(btempty, 9);
val it = false : bool
-
```

```
- fun lookup (btempty,_) = false
=   |     lookup(bt(root:int,left,right),x:int) =
=         if (x = root) then true
=         else (if (x <= root) then lookup(left,x)
=                   else lookup(right,x) );
val lookup = fn : int BinaryTree * int -> bool
-
```

Order functions and code:

```
- fun inorder (btempty) = []
= |     inorder(bt(root:'a,left,right)) =
= inorder(left) @ (root :: inorder(right));
val inorder = fn : 'a BinaryTree -> 'a list
- fun preorder (btempty) = []
= |     preorder(bt(root:'a,left,right)) =
= root :: (preorder(left) @ preorder(right));
val preorder = fn : 'a BinaryTree -> 'a list
- fun postorder (btempty) = []
= |     postorder(bt(root:'a,left,right)) =
= (postorder(left) @ postorder(right)) @ (root :: []);
val postorder = fn : 'a BinaryTree -> 'a list
-
```

```
- inorder(Tree);
val it = [5,7,1,4,10,0] : int list
-
```

```
- inorder(Tree2);
val it = [3,5,6,8,10,8] : int list
```

```
- preorder(Tree);
val it = [5,7,10,4,1,0] : int list
-
```

```
- preorder(Tree2);
val it = [3,5,10,8,6,8] : int list
```

```
- postorder(Tree);
val it = [1,4,0,10,7,5] : int list
-
```

```
- postorder(Tree2);
val it = [6,8,8,10,5,3] : int list
```

Subtrees:

```
- fun left_subtree btempty = btempty
= | left_subtree(bt(_,left,_)) = left;
val left_subtree = fn : 'a BinaryTree -> 'a BinaryTree
```

```
- left_subtree(Tree2);
val it = btempty : int BinaryTree
```

```
- left_subtree(Tree);
val it = btempty : int BinaryTree
```

```
- fun right_subtree btempty = btempty
= | right_subtree(bt(_,_,right)) = right
= ;
val right_subtree = fn : 'a BinaryTree -> 'a BinaryTree
```

```
- right_subtree(Tree2);
val it = bt (5,btempty,bt (10,bt #,bt #)) : int BinaryTree
```

```
- right_subtree(Tree);
val it = bt (7,btempty,bt (10,bt #,bt #)) : int BinaryTree
-
```

Label:

```
exception label_has_nil_argument
- fun label btempty = raise label_has_nil_argument
= | label(bt(value,_,_)) = value;
val label = fn : 'a BinaryTree -> 'a
-
```

```
- label(Tree2);
val it = 3 : int
```

```
- label(Tree);
val it = 5 : int
```

PROLOG

1. Let us consider the following set of facts that describe the mother predicate.

mother(linda, paul).

mother(cathy, andrew).

mother(cathy, laura)

- Define a predicate female(X) which holds iff X is a female

female(cathy)
female(linda)
female(laura)

- Define a predicate sister(X,Y) which holds iff X and Y are sisters

F(X) /\ M(Z,Y) /\ M(Z,X) → Sister(X,Y)

- Implement female and sister in PROLOG

- Provide screenshots

2. Implement the function g such that g(x) = x+5.